The Steiner Tree Problem for Terminals on the Boundary of a Rectilinear Polygon*

Siu-Wing Cheng
Department of Computer Science
HKUST, Clear Water Bay
Hong Kong
scheng@cs.ust.hk

September 4, 1997

Abstract

Given a simple rectilinear polygon P with k sides and n terminals on its boundary, we present an $O(k^3 n)$-time algorithm to compute the minimal rectilinear Steiner tree lying inside P interconnecting the terminals. We obtain our result by proving structural properties of a selective set of minimal Steiner trees and exploiting them in a dynamic programming algorithm.

Keywords: Rectilinear Steiner tree, homotpic routing, dynamic programming.

*Research supported partly by RGC CERG HKUST 190/93E and RGC CERG HKUST 650/95E.

1 Introduction

We study the problem of computing a minimal Steiner tree interconnecting a set of *terminals* (points) lying on the boundary of a simple rectilinear polygon. The terminals can lie at polygon vertices or on polygon sides and not all polygon vertices have to be occupied by terminals. The minimal Steiner tree desired is required to lie inside the given polygon. The application of the problem in single-layer homotopic routing [8] in VLSI design is described in [7].

If the solution is required to be a subgraph of a given underlying planar graph G, then the problem becomes the well-known Steiner tree problem for planar graphs. Note that the terminals lie on the exterior face of G. Let n be the number of terminals. Let $m \geq n$ be the number of vertices in G. For arbitrary planar graph G, Provan [10] presented an algorithm that runs in $O(m^2 n^2)$ time, and Bern [2] and Erickson *et al* [6] obtained a better running time of $O(mn^3 + (m \log m)n^2)$. (The algorithms in [2, 6] can deal with some generalized versions of the problem.) In our problem, G is actually a grid graph and every finite face has exactly four incident vertices. Kaufmann *et al* [7] exploited this fact to improve the running time further to $O(\min\{n^4 \log n, mn^2\})$. One cannot bound m as a function of n and m can be very large even when the geometry of the problem is not complicated. For example, if the n terminals are evenly distributed on the boundary of a rectangle, then G contains the arrangement of $n/2$ vertical lines and $n/2$ horizontal lines. Hence, m can be $\Theta(n^2)$ and the running time of the algorithm in [7] becomes $O(n^4)$. For our problem, it is actually not necessary to specify the underlying grid graph. Cohoon *et al.* [5] and Agarwal *et al.* [1] presented optimal $O(n)$-time algorithms for constructing a rectilinear Steiner tree for points lying on the boundary of a rectangle. Subsequently, Richards and Salowe [11], Cheng *et al* [3], and Cheng and Tang [4] solve the problem for *rectilinear convex polygons* in $O(k^4 n)$, $O(n^3)$, and $O(k^2 n)$ time respectively, where k is the number of sides of the polygon.

In this paper, we present an $O(k^3 n)$-time algorithm for any simple rectilinear polygon. We assume that the sides of the input polygon are given in cyclic order and the terminals on each side are given in sorted order. The time complexity analysis assumes that $k = O(n)$. Thus, our algorithm runs in linear time when the polygon has constant complexity. We derive our results by proving some structural properties for a selective set of *canonical* minimal Steiner trees, and developing a dynamic programming algorithm that exploits these properties. The structural properties proved in this paper are those in [11] (proved in the case of rectilinear convex polygons) extended to the case of simple rectilinear polygons. Our definition of canonical Steiner trees is not the same as that in [11]. Although the skeleton of our proofs is similar to that in [11], there are substantial differences and the new definition of canonical Steiner trees allows us to simplify the proofs (e.g., our proofs do not consist of orientation dependent case analyses).

Throughout this paper, we shall omit the word "rectilinear" for simplicity. Section 2 provides some definitions regarding the input polygon and Steiner trees. Section 3 defines a selective set of canonical minimal Steiner trees and prove some structural properties. Section 4 describes the preprocessing essential for the subsequent computation. One of the goals is to compute and store pointers so that we can jump from one vertex to another vertex in constant time. Section 5 provides the subproblem definitions for our dynamic programming algorithm and gives an overview of the algorithm. Section 6 describes the solutions of the subproblems. In Section 7, we conclude with the time complexity analysis and discuss how to handle the case where $k \gg n$.

2 Preliminaries and definitions

Polygon and vertices. A simple rectilinear polygon is a polygon without holes and whose boundary consists of vertical and horizontal sides. Given a simple rectilinear polygon P, $\mathsf{bd}\, P$ denotes its boundary and $\mathsf{int}\, P$ denotes its interior. We call the vertices of P *boundary vertices*. A boundary vertex is *reflex* if the interior angle at the vertex equals $\frac{3\pi}{2}$. A boundary vertex is *convex* if the interior angle at the vertex equals $\frac{\pi}{2}$. We assume that the polygon sides are given in cyclic order around P and the terminals on each side are given in sorted order.

Given any grid vertices a and d on $\mathsf{bd}\, P$, $\mathsf{bd}\, [a, d]$ denotes the closed polygonal curve on $\mathsf{bd}\, P$ traversed from a to d in clockwise order. We also use $\mathsf{bd}\, (a, d]$, $\mathsf{bd}\, [a, d)$, and $\mathsf{bd}\, (a, d)$ to denote the semi-open and open polygonal curves $\mathsf{bd}\, [a, d] - \{a\}$, $\mathsf{bd}\, [a, d] - \{d\}$, and $\mathsf{bd}\, [a, d] - \{a, d\}$, respectively. The *interval* $[a, d]$ denotes the union of $\{a, d\}$ and *the set of terminals* lying on $\mathsf{bd}\, [a, d]$.

Grid graph. For each terminal and reflex boundary vertex on $\mathsf{bd}\, P$, shoot vertical and horizontal rays into $\mathsf{int}\, P$ till they hit $\mathsf{bd}\, P$. The arrangement formed by these intercepted rays and $\mathsf{bd}\, P$ is the *grid graph*. The vertices and edges of the grid graph are called *grid vertices* and *grid edges*. Each maximal collinear sequence of grid edges lying in $\mathsf{int}\, P$ is a *grid line*. Since a grid line is required to be in $\mathsf{int}\, P$, a grid line may not be a maximal collinear sequence of grid edges in the grid graph.

Steiner tree. We say that a Steiner tree T lying inside P *interconnects* a set of grid vertices if these grid vertices form a subset of the vertices of T. W.l.o.g., we can assume that the two tree edges incident to any degree-2 tree vertex in $\mathsf{int}\, P$ are not collinear; otherwise, that tree vertex would be redundant. A maximal collinear sequence of tree edges in $\mathsf{int}\, P$ is called a *line*. (Since a line is required to be in $\mathsf{int}\, P$, a line may not be a maximal collinear sequence of tree edges in T.) A *complete line* is a line whose two endpoints are on $\mathsf{bd}\, P$. A degree-2 tree vertex in $\mathsf{int}\, P$ is a *corner-vertex*. A line incident to a corner-vertex is a *leg*. A degree-3 or degree-4 tree vertex in $\mathsf{int}\, P$ is a T-vertex. A *head* is a line that contains two collinear tree edges which are incident to a T-vertex. Take a T-vertex v and a head h through v. A *body* is a maximal sequence of collinear tree edges in $\mathsf{int}\, P$ that can be visited when we start at v and walk away from v in a direction perpendicular to h. Let ℓ be any collinear sequence of tree edges. We say that a tree edge is *pinned to ℓ* at a tree vertex v if the tree edge is perpendicular to ℓ and incident to v on ℓ. If v is in the interior of ℓ, then we can also say that the tree edge is pinned to the interior of ℓ. A tree edge is *pinned to the left/right* of ℓ if the tree edge is pinned to and lies on the left/right of ℓ.

3 Structure and properties of Steiner trees

Let L be the set of grid lines incident to reflex boundary vertices. Let V be the union of the following three set of points:

1. The endpoints of grid lines in L.
2. The set of grid vertices and terminals a neighboring to some endpoint u of a grid line in L, i.e., $[a, u] = \{a, u\}$ or $[u, a] = \{a, u\}$.

3. The set of grid vertices and terminals b neighboring to some boundary vertex v, i.e., $[b, v] = \{b, v\}$ or $[v, b] = \{b, v\}$.

A *purple grid line* is a grid line incident to a vertex in V. Each (convex or reflex) boundary vertex and each endpoint of a purple grid line is a *purple vertex*. There are $O(k)$ purple vertices. Define the *interior degree* of a tree vertex to be the number of incident tree edges that lie in $\mathsf{int}\, P$. Define a *non-alternating edge pair* to be a pair of tree edges pinned to the same side of a line at two adjacent tree vertices in $\mathsf{int}\, P$.

To restrict our search space, we select a subset of minimal Steiner trees interconnecting all terminals on $\mathsf{bd}\, P$ by the following criteria *in the listed order of priority*:

1. *Boundary-length criteria*: maximize the length of intersection of the tree and $\mathsf{bd}\, P$.
2. *Purple-degree criteria*: maximize the sum of interior degrees of purple vertices.
3. *Corner-number criteria*: minimize the number of corner-vertices.
4. *Terminal-degree criteria*: maximize the sum of interior degrees of terminals.
5. *Alternating criteria*: minimize the number of non-alternating edge pairs.

We call the minimal Steiner trees selected *canonical*. In the following, we prove properties about canonical trees. Two main operations to be used are *flipping corners* and *sweeping a collinear sequence of edges* in the tree. Figure 1 shows the flipping of corners. Figure 2 shows the sweeping operation. To sweep a collinear sequence ℓ of edges, both endpoints of ℓ must

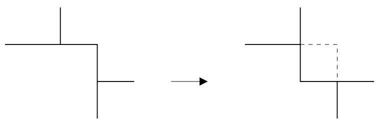


Figure 1.

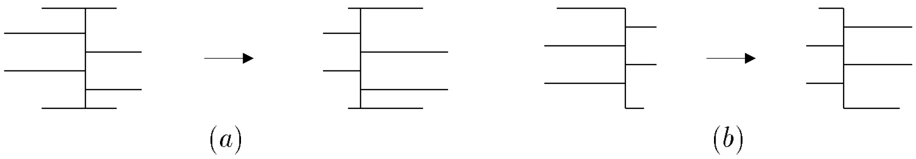


Figure 2.

have degree at least 2. During sweeping, the tree edges pinned to ℓ (at vertices in the interior of ℓ or at the endpoints of ℓ) will be lengthened or shortened accordingly. See Figure 2(a) and (b) for two examples.

Lemma 1 *There is no non-alternating edge pair.*

Proof. Assume to the contrary that two tree edges e_1 and e_2 are pinned to the same side of a line ℓ at adjacent tree vertices u and v in $\mathsf{int}\, P$. W.l.o.g., suppose that ℓ is horizontal, e_1 and e_2 are above ℓ, and e_1 is to the left of e_2. Let e_3 be the tree edge uv. Let ℓ_1 and ℓ_2 be the two lines containing e_1 and e_2 respectively. Let w_i be the upper endpoint of ℓ_i, $i = 1, 2$. Let x_i be the upper endpoint of e_i, $i = 1, 2$. Four other tree edges au, bu, cv, and dv may possibly exist as shown by dotted lines in Figure 3(a).

Claim. *Suppose that w_1 is lower than w_2. Then w_1 is a corner-vertex, w_1 is the upper endpoint of e_1, the horizontal leg incident to w_1 lies on the left of e_1, au and cv do not exist, bu and dv must exist, and x_2 is not a corner-vertex.*

Proof. No tree edge can be pinned to the right of ℓ_1 above e_3. Otherwise, sweeping e_3 upward will make the tree overlap itself, a contradiction. Also, w_1 cannot lie on bd P; otherwise, sweeping e_3 upward will make e_3 hit bd P and this contradicts the boundary-length criteria or the purple-degree criteria. Thus, w_1 must be a corner-vertex and the horizontal leg incident to w_1 lies on the left of e_1. The edge au does not exist and no tree edge is pinned to the left of ℓ_1 at a vertex above u and below w_1. Otherwise, flipping the corner at w_1 makes the tree overlap itself, a contradiction. Thus, w_1 is the upper endpoint of e_1. Similar argument to prevent self-overlapping of the tree by corner flipping shows that bu must exist, and cv or dv must exist. If cv exists, then we can sweep e_3 upward to become incident to w_1 and v cannot become a corner-vertex. So the number of corner-vertices decreases which contradicts the corner-number criteria. Thus, dv exists instead of cv. The presence of e_3 and dv forbids x_2 to be a corner-vertex; otherwise, the corner at x_2 can be flipped to make the tree overlap itself. This proves the claim.

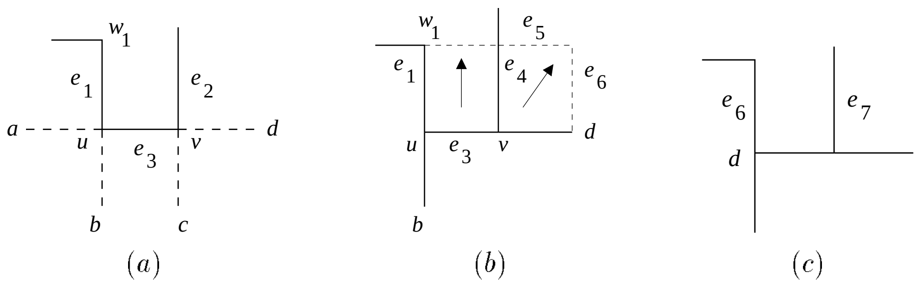


Figure 3.

W.l.o.g., assume that w_1 is lower than w_2 and we apply the claim to obtain the configuration in Figure 3(b). We sweep e_3 upward to become incident to w_1 and so w_1 is no longer a corner-vertex. Let now e_4 be the vertical edge incident to the new e_3 and v. Then we flip the corner at v which eliminates the vertical edge e_4, but creates a new horizontal edge e_5 and a new vertical edge e_6. (By the boundary-length criteria and purple-degree criteria, we can assume that the flipping does not cause new overlapping with bd P.) The vertex d cannot become a new corner-vertex unless d was a degree-2 tree vertex in int P with two collinear tree edges incident to it. But such a tree vertex can be assumed to be non-existent which implies that the number of corner-vertices is unchanged by the flipping. Let's count the number of non-alternating edge pairs. After flipping, e_1 and e_4 cease to be a non-alternating edge pair. If the new e_3 (resp. e_5) forms a non-alternating edge pair with an edge e above it, then e and the old e_3 (resp. dv) formed a non-alternating edge pair in the original tree. Thus, the new e_3 and e_5 do not increase the number of non-alternating edge pairs. It remains to analyze the effect of e_6.

If e_6 does not form a non-alternating edge pair with a vertical edge to the right of e_6, then we have a net decrease in the number of non-alternating edge pairs, which contradicts the alternating criteria. Suppose that e_6 forms a non-alternating edge pair with an edge e_7, see Figure 3(c). The upper endpoint of the line containing e_7 cannot be lower than the upper endpoint of e_6; otherwise, the symmetric version of the claim would imply that the upper endpoint of e_6 is not a corner-vertex, a contradiction. Thus, we can apply the claim to e_6 and e_7 and produce the configuration Figure 3(c) which is identical to Figure 3(b). Hence, we can repeat the sweeping and flipping operations in Figure 3(b) to Figure 3(c) again. Since the given tree is finite, this sequence of operations must terminate, namely, when no new non-alternating edge pair is produced. Subsequently, there will be a net decrease in the number of non-alternating edge pairs, contradicting the alternating criteria. □

Lemma 2 *Each body is a single tree edge with an endpoint on* bd P.

Proof. Let v be a T-vertex, let ℓ be a body incident to v, and let h be the head containing v perpendicular to ℓ. If the lemma is not true, then among the tree edges in int P pinned to ℓ, pick the one closest to v. This edge and a tree edge, which lies on h and is incident to v, form a non-alternating edge pair, contradicting Lemma 1. □

Lemma 3 *Each leg has an endpoint on* bd P.

Proof. Assume to the contrary that a leg ℓ does not have an endpoint on bd P. By Lemma 2, a leg cannot be a body. Thus, the two endpoints of ℓ are corner-vertices. W.l.o.g., let ℓ be vertical and there are two possible configurations as shown in Figure 4 (there is a symmetric case for each one). In Figure 4(a), we sweep ℓ to the right, while lengthening tree edges pinned to the

(a) (b)

Figure 4.

left of ℓ and shortening tree edges pinned to the right of ℓ. By Lemma 1, there are more tree edges shortened than those lengthened and so the tree length is reduced, a contradiction. We also sweep ℓ to the right in the case of Figure 4(b). By Lemma 1, equal numbers of tree edges must be shortened and lengthened and so the tree length remains unchanged. The sweeping will eventually cause the interior of ℓ to hit bd P, or an endpoint of ℓ to lie on bd P, or an endpoint of ℓ to become a T-vertex. In the first case, the length of intersection of the tree and bd P increases which contradicts the boundary-length criteria. In the latter two cases, the number of corner-vertices decreases which contradicts the corner-number criteria. □

Lemma 4 *Each leg or head is incident to a purple vertex.*

PROOF. Let ℓ be a leg of a corner C without any Steiner vertex in the interior of ℓ. By Lemma 3, ℓ has an endpoint v on $\mathsf{bd}\,P$. Vertex v must be a reflex boundary vertex which is purple; otherwise, the corner C can be flipped to overlap with the polygon side containing v, contradicting the boundary-length criteria.

The remaining case is that ℓ is a head. (The line ℓ may be a leg or a complete line.) Assume to the contrary that ℓ is not incident to a purple vertex and so the endpoint(s) of ℓ on $\mathsf{bd}\,P$ lie in the interior of polygon side(s). W.l.o.g., assume that ℓ is vertical, $N_r \geq 1$ tree edges are pinned to the right of ℓ at tree vertices in $\mathsf{int}\,P$, and $N_l \leq N_r$ tree edges are pinned to the left of ℓ at tree vertices in $\mathsf{int}\,P$. Given an endpoint u of ℓ on $\mathsf{bd}\,P$, define $e_l(u)$ and $e_r(u)$ be the tree edge pinned to the left and right of ℓ at u respectively. Define $e_l(u) = \emptyset$ and $e_r(u) = \emptyset$ if the corresponding edges do not exist.

Sweep ℓ to the right as long as the tree length does not increase and the length of the intersection of the tree and $\mathsf{bd}\,P$ does not decrease. We can assume that during sweeping, the tree does not overlap more with $\mathsf{bd}\,P$, the interior of ℓ does not hit $\mathsf{bd}\,P$, no endpoint of ℓ on $\mathsf{bd}\,P$ becomes incident to a reflex boundary vertex, and no corner-vertex endpoint of ℓ ceases to be a corner-vertex. Otherwise, it contradicts the boundary-length criteria, the purple-degree criteria, or the corner-number criteria. In order that the sweeping stops, $e_r(a) = \emptyset$ for an endpoint a of ℓ on $\mathsf{bd}\,P$ as $N_l \leq N_r$. Let b be the other endpoint of ℓ.

Claim. *When the sweeping stops, ℓ has a terminal endpoint u (a or b) such that $e_r(u) = \emptyset$ and $N_r \geq 1$, or $e_l(u) = \emptyset$ and $N_l \geq 1$.*

Proof. If a is a terminal, then we are done as. Suppose that a is not a terminal. Then $e_l(a) \neq \emptyset$ as $e_r(a) = \emptyset$. By Lemma 1, $N_l = N_r$ or $N_l = N_r - 1$. If b is a corner-vertex or b is a degree-3 vertex on $\mathsf{bd}\,P$, then we sweep ℓ to the right if $N_l = N_r - 1$ or sweep ℓ to the left if $N_l = N_r$ (maintain $e_r(a) = \emptyset$ as a is not a terminal). See Figure 5 for illustrations of the cases. In the first case, the tree

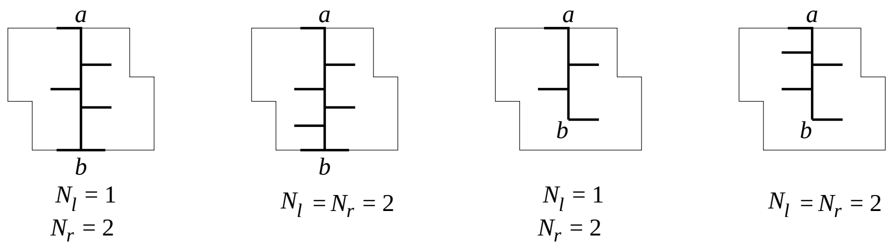


Figure 5.

length will not increase and the overlapping with $\mathsf{bd}\,P$ increases ($e_l(a)$ lengthens) which contradicts the boundary-length criteria. In the second case, the tree length decreases, contradicting the minimality of tree length. Thus, b must be a degree-2 vertex on $\mathsf{bd}\,P$ as ℓ is not a body. If b is not a terminal, then either $e_l(b) \neq \emptyset$ or $e_r(b) \neq \emptyset$. If $e_l(b) \neq \emptyset$, then we sweep ℓ to the left. $N_l \geq N_r - 1$ tree edges plus $e_l(a)$ and $e_l(b)$ shorten while only N_r tree edges lengthen. Thus, the tree length decreases, a contradiction. If $e_r(b) \neq \emptyset$, then we sweep ℓ to the right. The tree length will not increase and the overlapping with $\mathsf{bd}\,P$ will not decrease ($e_l(a)$ lengthens while $e_r(b)$ shortens), which contradicts that the sweeping of ℓ should have stopped. In all, we conclude that if a is not a terminal, then b must be a terminal with degree 2. If $e_r(b) = \emptyset$, then we know already that $N_r \geq 1$. If $e_r(b) \neq \emptyset$ and $e_l(b) = \emptyset$, then in order that we cannot sweep ℓ further to the right, $N_l = N_r \geq 1$. Hence, b satisfies the claim.

By the claim, w.l.o.g., let terminal u be the lower endpoint of ℓ, $e_r(u) = \emptyset$, and $N_r \geq 1$. Let u lie in the interior of the polygon side s. Let f be the lowest tree edge pinned to the right and interior of ℓ. Since f is a body, by Lemma 2, the right endpoint of f lies on $\mathsf{bd}\, P$ and let it be y. See Figure 6 for an illustration. The endpoint u is not a purple vertex; otherwise, we

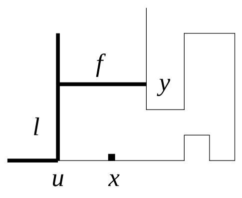


Figure 6.

have violated the purple-degree criteria as ℓ is assumed not to be incident to a purple vertex before sweeping. To prevent u from being purple, there must be a terminal x on s that lies between u and the right endpoint of s (a boundary vertex), and x lies to the left of y. Since $e_r(u) = \emptyset$, the tree path between x and y does not pass through u. Thus, we can introduce $\mathsf{bd}\,[x, u]$ to create a cycle and then remove f to break the cycle. As f is longer than $\mathsf{bd}\,[x, u]$, the tree length decreases which is a contradiction. Hence, our initial assumption that ℓ is not incident to a purple vertex is false. ❏

Lemma 5 *Each complete line not containing a Steiner vertex in its interior is aligned with some grid line. Each body is aligned with some grid line.*

PROOF. Let ℓ be a body. By Lemma 2, an endpoint v of ℓ lies on $\mathsf{bd}\, P$. Assume to the contrary that the lemma is false. Then v is not a terminal or reflex boundary vertex. Also, the degree of v is 3, otherwise, the degree of v is 2 and we can sweep ℓ to shorten the tree edge pinned to ℓ at v, contradicting the minimality of tree length. Since the degree of v is 3, we are able to sweep ℓ, while keeping the tree length fixed and the length of the intersection of the tree with $\mathsf{bd}\, P$ fixed. One of the following events will eventually happen: (1) the tree overlaps itself, (2) the interior of ℓ hits $\mathsf{bd}\, P$, or (3) ℓ becomes incident to a reflex boundary vertex or a terminal, or (4) ℓ becomes incident to the endpoint, which is a corner-vertex, of the head perpendicular to ℓ. Event (1) contradicts the minimality of tree length. Event (2) contradicts the boundary-length criteria. Event (3) contradicts the purple-degree criteria or the terminal-degree criteria. Event (4) contradicts the corner-number criteria. Similar analysis applies when ℓ is a complete line with no Steiner vertex in its interior. ❏

Corollary 1 *A canonical Steiner tree is a subgraph of the grid graph.*

PROOF. Lemma 5 says that every complete line with no Steiner vertex in its interior and every body is aligned with some grid line. Lemma 4 imply that every complete line with Steiner vertex in its interior and every leg is also aligned with some grid line. This covers all tree edges in $\mathsf{int}\, P$. ❏

4 Preprocessing

In preprocessing, we aim to compute in $O(kn)$ time the following entities:

- the grid vertices on $\mathsf{bd}\, P$ in cyclic order,
- the purple grid lines, and the set $\mathbb{P}$ of all purple grid vertices on $\mathsf{bd}\, P$ and intersections of purple grid lines.
- a queue *Interval_queue* containing all intervals $[a, d]$, where a or d is purple or ad is a grid line, in non-decreasing order of the number of grid vertices on $\mathsf{bd}\, [a, d]$. Our dynamic programming algorithm will process intervals in *Interval_queue* in order.
- for each grid vertex x on $\mathsf{bd}\, P$, the following pointers

$v(x)$	If x is incident to a vertical grid line ℓ , then $v(x)$ points to the opposite endpoint of ℓ. Otherwise, $v(x)$ is null.
$h(x)$	If x is incident to a horizontal grid line ℓ , then $h(x)$ points to the opposite endpoint of ℓ. Otherwise, $h(x)$ is null.
$g(x)$	The next grid vertex on $\mathsf{bd}\, P$ succeeding x in clockwise order.
$\bar{g}(x)$	The next grid vertex on $\mathsf{bd}\, P$ succeeding x in anticlockwise order.
$t(x)$	The next terminal succeeding x in clockwise order.
$\bar{t}(x)$	The next terminal succeeding x in anticlockwise order.
$p(x)$	The next purple grid vertex on $\mathsf{bd}\, P$ succeeding x in clockwise order.
$\bar{p}(x)$	The next purple grid vertex on $\mathsf{bd}\, P$ succeeding x in anticlockwise order.

We first compute the $O(n)$ grid lines in $O(kn)$ time by brute force. Then we plane-sweep the grid lines in rightward and upward directions to compute the cyclic ordering of grid vertices on $\mathsf{bd}\, P$. In the rightward plane sweep, we compute the grid vertices in sorted order on each horizontal polygon side. The grid vertices in sorted order on vertical polygon sides are computed in the upward plane sweep. Then we join the k sorted lists to produce the cyclic ordering. The details are as follows. Recall that the terminals are given in sorted order on each polygon side in the input. Thus, by an $O(kn)$-time merging step, we obtain a queue of boundary vertices of the input polygon and terminals sorted in non-decreasing x-coordinates. We maintain a list L of horizontal polygon sides intersected by the sweep line. The sweep line stops at every x-coordinate in the queue. At each stop, the sweep line encounters a vertical grid line whose endpoints induce two new grid vertices on two polygon sides adjacent in L. The entire sweep takes $O(n)$ time and the grid vertices on each horizontal polygon side are produced in left-to-right order. The upward plane sweep is performed similarly.

Using the grid lines, we can set up the pointers $v(x)$ and $h(x)$ for each grid vertex x on $\mathsf{bd}\, P$. By scanning the cyclic ordering of grid vertices, we can also set up the pointers $g(x)$, $\bar{g}(x)$, $t(x)$ and $\bar{t}(x)$. This takes $O(n)$ time.

Now, we compute the purple grid lines and the set $\mathbb{P}$. We first color all grid lines incident to reflex boundary vertices purple. Then using the above pointers computed for grid vertices on $\mathsf{bd}\, P$, we apply the definitions of purple grid lines to identify the remaining purple grid lines.

Then the purple vertices are identified as well. Now, by scanning the cyclic ordering of grid vertices again, we can set up the pointers $p(x)$ and $\bar{p}(x)$ for each grid vertex x on $\mathsf{bd}\, P$. This takes $O(n)$ time. The set $\mathbb{P}$ is obtained in $O(k^2)$ time simply by computing the intersections of all purple grid lines and then taking the union with the purple grid vertices on $\mathsf{bd}\, P$.

Each interval in *Interval_queue* is represented as an ordered pair of grid vertices. So it is easy to generate all the intervals in *Interval_queue* in $O(kn)$ time but *Interval_queue* needs to be sorted afterward. We first assign consecutive and increasing integer indices to the grid vertices on $\mathsf{bd}\, P$ in cyclic order. Let v be the grid vertex on $\mathsf{bd}\, P$ with index 1. By scanning the cyclic ordering of grid vertices, we can compute the number of grid vertices $N(\mathsf{bd}\,[a, v])$ on $\mathsf{bd}\,[a, v]$ for all grid vertices a in $O(n)$ time. Then for each interval $[a, d]$, the number of grid vertices on $\mathsf{bd}\,[a, d]$ is either $N(\mathsf{bd}\,[a, v]) + M - N(\mathsf{bd}\,[d, v]) + 1$ if $v \in \mathsf{bd}\,[a, d]$, or $N(\mathsf{bd}\,[a, v]) - N(\mathsf{bd}\,[d, v]) + 1$ if $v \notin \mathsf{bd}\,[a, d]$, where M is the total number of grid vertices on $\mathsf{bd}\, P$. Since $N(\mathsf{bd}\,[a, d]) \leq 3n + k$ for any interval $[a, d]$, we can bucket sort *Interval_queue* in $O(kn)$ time.

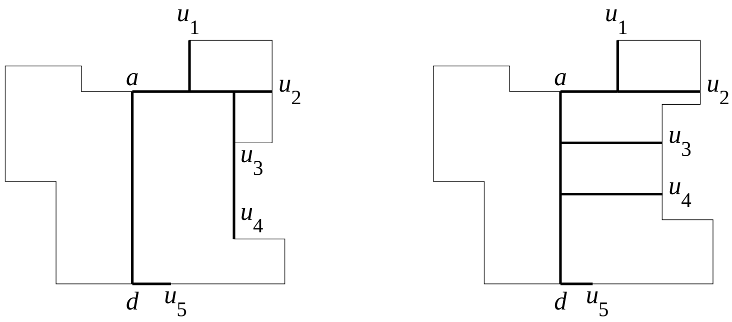
5 Subproblems and summary of the algorithm

Consider a Steiner tree T (not necessarily of minimum length) that lies inside P and interconnects *any subset of terminals* on $\mathsf{bd}\, P$. We say that T satisfies the *purple properties* if T satisfies Lemmas 2, 3, 4, and 5 and T is a subgraph of the grid graph. Results in Section 3 imply that there exists a minimal Steiner tree that satisfies the purple properties. (Note that a Steiner tree satisfying the purple properties is not necessarily canonical.) Therefore, we can focus on constructing candidate trees that satisfy the purple properties and pick the minimal one among the candidates.

5.1 Subproblem definitions

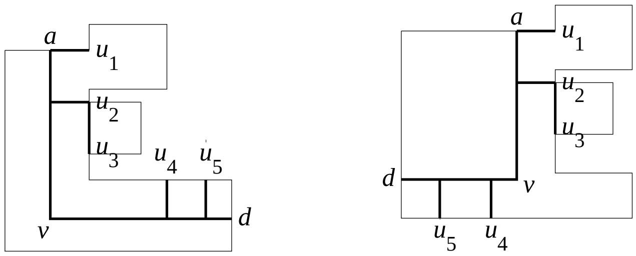
We will construct six categories of Steiner trees that interconnect subsets of terminals on $\mathsf{bd}\, P$ and satisfy purple properties. Then we compose trees from appropriate categories to form the minimal Steiner tree desired. Each tree in a category is a Steiner tree of minimum length under the *vertex constraints* and *topology constraints* specific to that category. The six categories are described below. Recall that $\mathbb{P}$ is the set of purple vertices on $\mathsf{bd}\, P$ and the intersections of purple grid lines.

- $T([a, d])$
  VERTEX CONSTRAINTS: (i) a and d are grid vertices on $\mathsf{bd}\, P$, (ii) a or d is purple or ad is a grid line.
  TOPOLOGY CONSTRAINTS: (i) interconnects $[a, d]$, (ii) satisfies the purple properties, (iii) does not intersect $\mathsf{bd}\,(d, a)$.
- $I([a, d])$ (See Figure 7)
  VERTEX CONSTRAINT: ad is a grid line.
  TOPOLOGY CONSTRAINTS: (i) interconnects $[a, d]$, (ii) satisfies the purple properties, (iii) contains ad.
- $L(v, [a, d])$ (See Figure 8)
  VERTEX CONSTRAINTS: (i) a and d are purple vertices, (ii) $v \in \mathbb{P} \cap \mathsf{int}\, P$.
  TOPOLOGY CONSTRAINTS: (i) interconnects $[a, d]$, (ii) satisfies the purple properties, (iii) contains a corner with corner-vertex v and legs av and dv.



(a) (b)

Figure 7. The figures show two examples of $I([a,d])$ interconnecting the points a, u_1, u_2, u_3, u_4, u_5, and d. In (a), ad is non-purple and so by the purple properties, no tree edge can be pinned to its interior. In (b), ad is purple and there can be tree edges pinned to its interior.



(a) (b)

Figure 8. The figures show two examples of $L(v,[a,d])$ interconnecting the points a, u_1, u_2, u_3, u_4, u_5, and d. The angle $\angle avd$ facing $\mathsf{bd}\,[a,d]$ can be $\frac{\pi}{2}$ or $\frac{3\pi}{2}$.

- This category contains two kinds of trees $J_a(v,w,[a,d])$ and $J_d(v,w,[a,d])$ with symmetric vertex and topology constraints.

  $J_a(v,w,[a,d])$ (See Figure 9)
  Vertex constraints: (i) a and d are grid vertices on $\mathsf{bd}\,P$ and a is purple, (ii) $v, w \in \mathbb{P}$ and $v \notin \mathsf{bd}\,[a,d]$, (iii) either $w = v$, or $v \in \mathsf{int}\,P$, $w \in \mathsf{bd}\,(d,a)$, and $av \perp vw$, (iv) av lies in $\mathsf{int}\,P$ and if $w \neq v$, vw lies in $\mathsf{int}\,P$.
  Topology constraints: (i) interconnects $[a,d]$, (ii) satisfies the purple properties, (iii) contains av and vw, (iv) degree of w equals 1 and degree of v is at most 2, (v) does not intersect $\mathsf{bd}\,(d,a)$ except at w when $w \in \mathsf{bd}\,(d,a)$, (vi) for all body pq pinned to av with p on av and q on $\mathsf{bd}\,[a,d]$, the angle $\angle apq$ facing $\mathsf{bd}\,[a,q]$ equals $\frac{\pi}{2}$.

  $J_d(v,w,[a,d])$
  Vertex constraints: (i) a and d are grid vertices on $\mathsf{bd}\,P$ and d is purple, (ii) $v, w \in \mathbb{P}$ and $v \notin \mathsf{bd}\,[a,d]$, (iii) either $w = v$, or $v \in \mathsf{int}\,P$, $w \in \mathsf{bd}\,(d,a)$, and $av \perp vw$, (iv) dv lies in $\mathsf{int}\,P$ and if $w \neq v$, vw lies in $\mathsf{int}\,P$.
  Topology constraints: (i) interconnects $[a,d]$, (ii) satisfies the purple properties,

(iii) contains dv and vw, (iv) degree of w equals 1 and degree of v is at most 2, (v) does not intersect $\mathsf{bd}\,(d, a)$ except at w when $w \in \mathsf{bd}\,(d, a)$, (vi) for all body pq pinned to dv with p on dv and q on $\mathsf{bd}\,[a, d]$, the angle $\angle dpq$ facing $\mathsf{bd}\,[q, d]$ equals $\frac{\pi}{2}$.

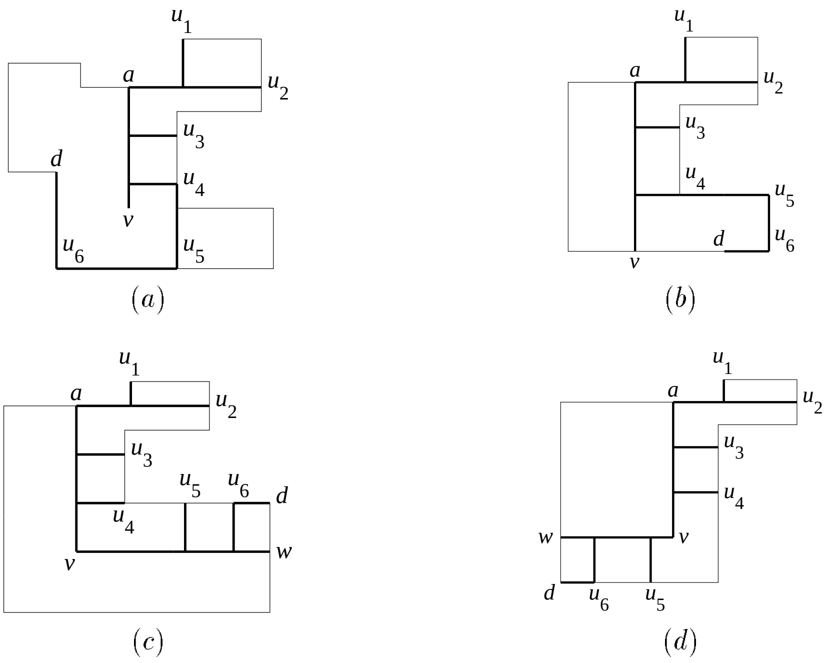


Figure 9. In (a) and (b), $J_a(v, v, [a, d])$ interconnects a, d, v, and the terminals u_i, $1 \leq i \leq 6$. Note that $w = v$, degree of v equals 1, and v is allowed to lie on $\mathsf{bd}\,(d, a)$ as shown in (b). In (c) and (d), two different configurations for $J_a(v, w, [a, d])$ when $w \neq v$ are shown. Note that w has degree 1 and lies on $\mathsf{bd}\,(d, a)$ and there may be bodies pinned to vw. In all figures, by definition, any body pinned to av must lie on the right of av to form an angle $\frac{\pi}{2}$ with av facing $\mathsf{bd}\,[a, d]$.

- $J_a^*(v, w, [a, d))$: the minimal $J_a(v, w, [a, x])$ among all grid vertex $x \in \mathsf{bd}\,[a, d)$ such that $[a, x]$ contains all the terminals in $[a, d]$.

  $J_d^*(v, w, (a, d])$: the minimal $J_d(v, w, [x, d])$ among all grid vertex $x \in \mathsf{bd}\,(a, d]$ such that $[x, d]$ contains all the terminals in $[a, d]$.

- $T^*([a, d))$: the minimal $T([a, x])$, where a is purple, among all grid vertex $x \in \mathsf{bd}\,[a, d)$ such that $[a, x]$ contains all the terminals in $[a, d]$.

  $T^*((a, d])$: the minimal $T([x, d])$, where d is purple, among all grid vertex $x \in \mathsf{bd}\,(a, d]$ such that $[x, d]$ contains all the terminals in $[a, d]$.

We will present a dynamic programming algorithm to compute the lengths of trees in the above six categories. Standard augmentation techniques can be applied to reconstruct the trees (without sacrificing the asymptotic running time) given the trace of computation. Therefore, we shall just focus on computing the tree lengths in the rest of this paper. The vertex and topology constraints determine the numbers of trees in each category. The notations for tree lengths and the number of trees in each category are tabulated below.

Trees	Tree lengths	Numbers of trees
$T([a,d])$	$\tau([a,d])$	$O(n)$, if ad is a non-purple grid line $O(kn)$, if a or d is purple
$I([a,d])$	$i([a,d])$	$O(k)$, if ad is a purple grid line $O(n)$, if ad is a non-purple grid line
$L(v,[a,d])$	$\ell(v,[a,d])$	$O(k^2)$
$J_a(v,w,[a,d])$ $J_d(v,w,[a,d])$	$j_a(v,w,[a,d])$ $j_d(v,w,[a,d])$	$O(k^3)$, if d is purple $O(k^2n)$, if d is non-purple $O(k^3)$, if a is purple $O(k^2n)$, if a is non-purple
$J_a^*(v,w,[a,d))$ $J_d^*(v,w,(a,d])$	$j_a^*(v,w,[a,d))$ $j_d^*(v,w,(a,d])$	$O(k^2n)$ $O(k^2n)$
$T^*([a,d))$ $T^*((a,d])$	$\tau^*([a,d))$ $\tau^*((a,d])$	$O(kn)$ $O(kn)$

5.2 Order of computation and dependency

Recall that *Interval_queue* is a queue containing all intervals $[a,d]$, where a or d is purple or ad is a grid line, and *Interval_queue* is sorted in non-decreasing order of number of grid vertices on $\mathsf{bd}\,[a,d]$. Our dynamic programming algorithm will process intervals in *Interval_queue* in order and for each interval $[a,d]$ in *Interval_queue*, our algorithm computes the lengths of trees in the previous section (if they are defined) in the following order:

1. $j_a^*(v,w,[a,d))$ for all grid vertices v and w such that it is defined.
   $j_d^*(v,w,(a,d])$ for all grid vertices v and w such that it is defined.
2. $\tau^*([a,d))$ and $\tau^*((a,d])$.
3. $i([a,d])$.
4. $\ell(v,[a,d])$ for all grid vertex v such that it is defined.
5. $\tau([a,d])$.
6. $j_a(v,w,[a,d])$ for all grid vertices v and w such that it is defined.
   $j_d(v,w,[a,d])$ for all grid vertices v and w such that it is defined.

The above ordering must be respected because of the dependency in the computation. The computation for $[a,d]$ may also use lengths of trees defined for subintervals on $\mathsf{bd}\,[a,d]$ in *Interval_queue*. These tree lengths are available because, by the definition of *Interval_queue*, all such subintervals precede $[a,d]$ in *Interval_queue* and should have been processed. Figure 10 shows the dependency which is clearly cycle free. The dependency can be verified by studying the computation described in Section 6.

5.3 Obtaining the optimal Steiner tree

After processing the entire queue *Interval_queue*, we have obtained $i([a,d])$ and $\ell(v,[a,d])$ for all intervals $[a,d]$ in *Interval_queue* and for all appropriate grid vertex v. In the following, we

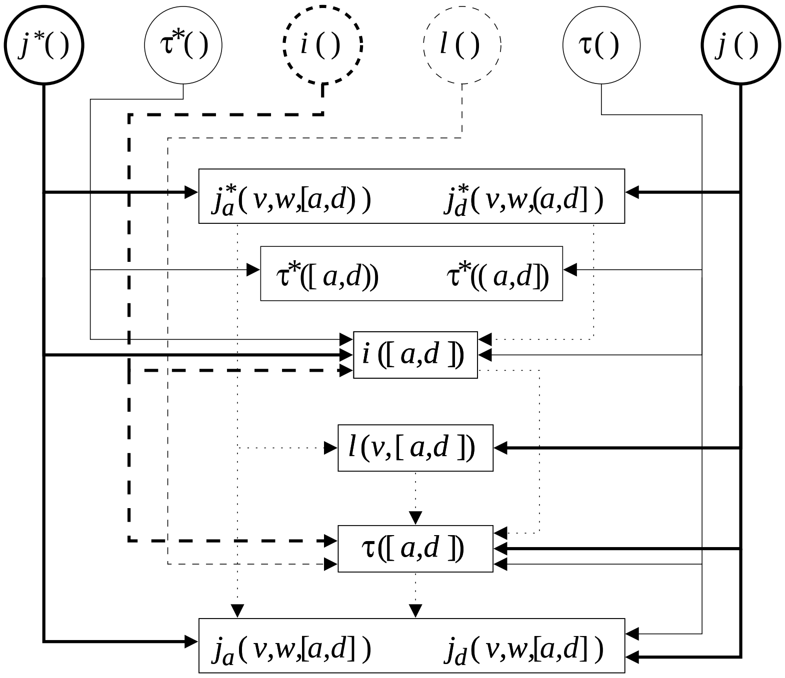


Figure 10. Each rectangle represents, as labeled, the set of tree lengths to be computed for $[a, d]$. The circles denote the sets of tree lengths, as labeled, computed for intervals preceding $[a, d]$ in *Interval_queue*. An arc going from one set of tree lengths X to another set of tree lengths Y means that the computation of a tree length in X uses a tree length in Y.

show how to obtain the length of the optimal Steiner tree interconnecting all terminals on $\mathsf{bd}\, P$. A useful observation here and in the rest of the paper is that if we hang a Steiner tree at a tree vertex v, then each subtree rooted at a child of v interconnects some interval $[a, d]$.

Lemma 6 *After processing the intervals in Interval_queue, the minimal Steiner tree interconnecting all terminals on* $\mathsf{bd}\, P$ *can be constructed in* $O(k^2 n)$ *time.*

Proof. Let T be a canonical Steiner tree. Then T can be classified as (1) T does not have any tree edge in $\mathsf{int}\, P$, (2) T contains a complete line, or (3) T contains a corner-vertex. In case (1), $|T| = \min\{\, |\mathsf{bd}\, P| - |\mathsf{bd}\, [a, t(a)]| : \text{terminal } a\,\}$. The minimization takes $O(n)$ time. In case (2), $|T| = \min\{\, i([a, d]) + i([d, a]) - |ad| : \text{grid line } ad\,\}$. The minimization takes $O(n)$ time. In case (3), suppose that T contains a corner with corner-vertex v and legs incident to purple vertices a and d on $\mathsf{bd}\, P$. Then $|T|$ equals $\ell(v, [a, d]) + \ell(v, [d, a]) - |av| - |dv|$. We simply try all $O(k^2)$ combinations of a, d, and v to return the minimum. Finally, it suffices to return the minimum among the values obtained in the above three cases. Again, standard augmentation can be applied to reconstruct the optimal Steiner tree. There are only $O(n)$ Steiner vertices and therefore the reconstruction takes $O(n)$ time. The $O(n)$ count is obtained by charging each T-vertex u to the endpoint on $\mathsf{bd}\, P$ of a body incident to u, and charging each corner-vertex v to an endpoint on $\mathsf{bd}\, P$ of a leg incident to v. ◻

In latter sections, we will show that processing the intervals in *Interval_queue* takes $O(k^3 n)$ time. Combined with Lemma 6, our claimed result of computing an optimal Steiner tree in $O(k^3 n)$ time follows. In the proof of Lemma 6, we see that only the tree lengths $i(\cdot)$ and

$\ell(\cdot,\cdot)$ will be useful ultimately. But due to the dependency in computation as shown in last subsection, we have to compute the other tree lengths as well.

6 Solutions to subproblems

In the following subsections, we will describe the computation of $j_a^*(v,w,[a,d))$, $j_d^*(v,w,(a,d])$, $i([a,d])$, $\ell(v,[a,d])$, $\tau([a,d])$, $j_a(v,w,[a,d])$, and $j_d(v,w,[a,d])$. In preprocessing, we have already obtained the pointers $v(x)$, $h(x)$, $g(x)$, $\bar{g}(x)$, $t(x)$, $\bar{t}(x)$, $p(x)$, and $\bar{p}(x)$ for each grid vertex x on $\mathsf{bd}\,P$. (See Section 4 for their definitions.)

6.1 Compute $j_a^*(v,w,[a,d))$ and $j_d^*(v,w,(a,d])$

Consider $j_a^*(v,w,[a,d))$. If $\bar{g}(d)=\bar{t}(d)$, then $j_a^*(v,w,[a,d))=j_a(v,w,[a,\bar{g}(d)])$. Otherwise, $j_a^*(v,w,[a,d))=\min\{\, j_a^*(v,w,[a,\bar{g}(d))),\, j_a(v,w,[a,\bar{g}(d)])\,\}$. This takes only $O(1)$ time. The value $j_d^*(v,w,(a,d])$ is computed similarly .

6.2 Compute $\tau^*([a,d))$ and $\tau^*((a,d])$

Consider $\tau^*([a,d))$. If $\bar{g}(d)=\bar{t}(d)$, then $\tau^*([a,d))=\tau([a,\bar{g}(d)])$. Otherwise, $\tau^*([a,d))=\min\{\,\tau^*([a,\bar{g}(d))),\,\tau([a,\bar{g}(d)])\,\}$. This takes only $O(1)$ time. The value $\tau^*((a,d])$ is computed similarly .

6.3 Compute $i([a,d])$

We establish the following result in this subsection.

Lemma 7 *Computing $i([a,d])$ takes $O(n)$ time when ad is a purple grid line, $O(n)$ time when ad is a non-purple grid line and $g(a)\,\bar{g}(d)$ is a purple grid line, and $O(1)$ time otherwise.*

Case 1 : ad is a purple grid line. Splitting $I([a,d])$ at d yields $J_a(d,d,[a,x])$ and $T([y,d])$ for some grid vertices x,y on $\mathsf{bd}\,[a,d]$, where $[a,x]$ contains the terminals in $[a,d]-[y,d]$. Thus, $i([a,d])$ equals

$$\min\{\, j_a^*(d,d,[a,y))+\tau([y,d])\ :\ \text{grid vertex } y \text{ on } \mathsf{bd}\,(a,d]\,\}. \tag{1}$$

The minimization takes $O(n)$ time.

Case 2 : ad is a non-purple grid line. By definition of purple grid lines, a and d must lie in the interior of two parallel opposite polygon sides such that $g(a)\,\bar{g}(d)$ is also a grid line parallel to ad. By the purple properties, since ad is non-purple, no tree edge is pinned to the interior of ad.

Suppose that $I([a,d])$ contains the edges $a\,g(a)$ and $\bar{g}(d)\,d$. If $g(a)\,\bar{g}(d)$ is a purple grid line, then $g(a)$ and $\bar{g}(d)$ are purple and $i([a,d])$ equals

$$\min\left\{\ \begin{array}{c}\tau^*([g(a),y))+\tau([y,\bar{g}(d)])\\ +|ad|+|a\,g(a)|+|\bar{g}(d)\,d|\end{array}\ :\text{grid vertex } y\in\mathsf{bd}\,(g(a),\bar{g}(d)]\ \right\}. \tag{2}$$

The minimization takes $O(n)$ time. If $g(a)\,\bar{g}(d)$ is a non-purple grid line, then we can construct $I([a,d])$ by adding to $I([g(a),\bar{g}(d)])$ the edges $a\,g(a)$ and $\bar{g}(d)\,d$, and then sweep the line $g(a)\,\bar{g}(d)$ to become incident to a and d. It is feasible to sweep the line $g(a)\,\bar{g}(d)$ as there is no tree edge pinned to its interior by purple properties. In this case, $i([a,d])$ equals

$$i([g(a),\bar{g}(d)]) + |a\,g(a)| + |\bar{g}(d)\,d|. \tag{3}$$

If $I([a,d])$ does not contain the edge $a\,g(a)$, then $I([a,d])$ is the union of ad and a subtree G that interconnects $[t(a),d]$. Let s_d be the polygon side containing d. Define $u_{t(a)} = v(t(a))$ or $h(t(a))$ such that $t(a)\,u_{t(a)}$ is a grid line parallel to ad and $u_{t(a)}$ lies on s_d. If such a grid line does not exist, then $u_{t(a)}$ is undefined.

Proposition 1 *G can be assumed to be either the union of the segment $u_{t(a)}\,d$ and a subtree interconnecting $[t(a),u_{t(a)}]$, or the union of the segment $\bar{p}(d)\,d$ and a subtree interconnecting $[t(a),\bar{p}(d)]$. The first possibility may happen only when $u_{t(a)}$ is defined.*

Proof. Let ρ be a rectilinear shortest path inside the polygon P directed from d to $t(a)$. We choose ρ to minimize the area of the open region bounded by ρ and $\mathsf{bd}\,[t(a),d]$. Let ρ leave s_d at a vertex x and let y be the vertex following x on ρ. If $u_{t(a)}$ is defined, then we have $x = u_{t(a)}$ and $y = t(a)$ and so ρ consists of the edges $u_{t(a)}d$ and $t(a)u_{t(a)}$. See Figure 11(a) for an illustration. If $u_{t(a)}$ is undefined, then $y \neq t(a)$. Now, either x is an endpoint of s_d (which is purple) or y must be a reflex boundary vertex which makes x a purple vertex by definition. See Figure 11(b) for an illustration. So ρ contains a segment connecting d and the purple vertex $\bar{p}(d)$.

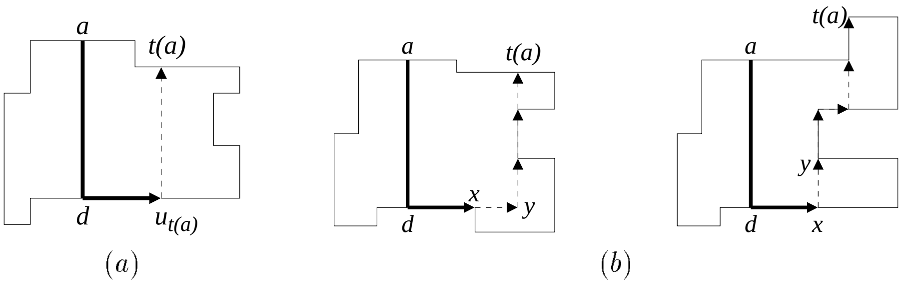


Figure 11. The directed path ρ is shown as an oriented polyline.

If G does not cross ρ, then the proposition is clearly true. Suppose that G crosses ρ. Let ρ' be the tree path directed from d to $t(a)$ in G. Since $\mathsf{bd}\,(a,t(a))$ does not contain any terminal, G does not contain any tree edge outside the region bounded by ρ' and $\mathsf{bd}\,[t(a),d]$. Therefore, some edge on ρ' must lie fully or partially to the left of ρ as G crosses ρ. Let e be the first such edge on ρ'.

If ρ' contains the segment $\bar{p}(d)\,d$, then G is clearly the union of the segment $\bar{p}(d)\,d$ and a subtree interconnecting $[t(a),\bar{p}(d)]$. Assume that ρ' does not contain the segment $\bar{p}(d)\,d$. Then e must have a non-purple endpoint v between d and $\bar{p}(d)$ on s_d and $e \perp s_d$. This means that e lies fully to the left of ρ and so v lies to the left of $u_{t(a)}$ in Figure 11(a) or to the left of x in Figure 11(b). W.l.o.g., we only consider the orientation as shown in Figure 11. Let s_a be the polygon side containing a. The right endpoint of s_a must lie to the right of e; otherwise, it would induce a purple vertex on s_d between d and v, contradicting that ρ' does not contain the segment $\bar{p}(d)\,d$. Since v is non-purple, e is either a body or a complete line without any Steiner vertex in its interior. Let w be the other endpoint of e.

If e is a body, then consider the head h through w. The left endpoint of h must be a corner-vertex; otherwise, h would cross ad which is impossible. See Figure 12(a). Since a corner-vertex

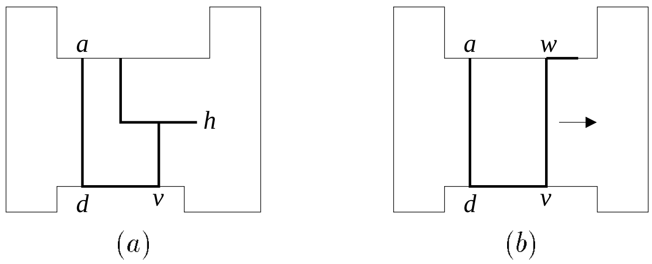


Figure 12.

is in $\mathbb{P}$, the left endpoint of h induces a purple vertex between d and v, contradicting that ρ' does not contain the segment $\bar{p}(d)\,d$. If e is a complete line without any Steiner vertex in its interior, then w lies on s_a. Since e lies to the left of ρ, $w \in \mathsf{bd}\,(a, t(a))$ and w is not a terminal. Thus, there must be a tree edge in G pinned to the right of e at w, and there is no tree edge in G pinned to the left of e at w as there is no terminal on $\mathsf{bd}\,(a, w)$ to be connected. No tree edge is pinned to the right of e at v, otherwise we can sweep e to the right to reduce the tree length, a contradiction. See Figure 12(b). Therefore, we can sweep e to the right while preserving the tree length and the purple properties. We sweep e until e becomes incident to a purple vertex or $t(a)$. In the first case, the modified G is the union of the segment $\bar{p}(d)\,d$ and a subtree interconnecting $[t(a), \bar{p}(d)]$. In the second case, the modified G is the union of the segment $u_{t(a)}\,d$ and a subtree interconnecting $[t(a), u_{t(a)}]$. ❑

By Proposition 1, $i([a, d])$ equals

$$\min \begin{cases} \tau^*((a, \bar{p}(d)]) + |\bar{p}(d)\,d| + |ad| \\ \tau([t(a), u_{t(a)}]) + |u_{t(a)}\,d| + |ad|, \text{ if } u_{t(a)} \text{ is defined.} \end{cases} \tag{4}$$

Formula (4) handles the case where $I([a, d])$ does not contain the edge $a\,g(a)$. For the symmetric case where $I([a, d])$ contains the edge $a\,g(a)$ but not the edge $\bar{g}(d)\,d$, a formula symmetric to formula (4) can be obtained by similar analysis. In all, when ad is a non-purple grid line, if $g(a)\,\bar{g}(d)$ is a purple grid line, we evaluate formula (2), and formula (4) and its symmetric counterpart. The minimum is $i([a, d])$. This takes $O(n)$ time. If $g(a)\,\bar{g}(d)$ is a non-purple grid line, we evaluate formula (3), and formula (4) and its symmetric counterpart. This takes $O(1)$ time.

6.4 Compute $\ell(v, [a, d])$

By definition, it is clear that $L(v, [a, d])$ equals the union of $J_a(v, v, [a, x])$ and $J_d(v, v, [y, d])$ for some grid vertices x, y on $\mathsf{bd}\,[a, d]$, where $[a, x]$ contains the terminals in $[a, d] - [y, d]$. Therefore, $\ell(v, [a, d]) = \min\{\, j_a(v, v, [a, y)) + j_d(v, v, [y, d]) \,:\, \text{grid vertex } y \text{ on } \mathsf{bd}\,(a, d]\,\}$. The minimization takes $O(n)$ time.

6.5 Compute $\tau([a,d])$

We show the following result in this subsection.

Lemma 8 *Computing $\tau([a,d])$ takes $O(1)$ time when ad is a non-purple grid line, and $O(k^2)$ time when a or d is purple.*

Case 1 : ad is a non-purple grid line. If $T([a,d])$ contains ad, then $T([a,d])$ can be set to be $I([a,d])$. Suppose that $T([a,d])$ does not contain ad. By the purple properties, no leg can be incident to a or d as they are non-purple. As $T([a,d])$ does not intersect $\mathsf{bd}\,(d,a)$ by its topological constraints, no tree edge of $T([a,d])$ crosses the grid line ad. This implies that no body can be incident to a or d (otherwise, the corresponding head must cross ad). Therefore, $a\,g(a)$ and $\bar{g}(d)\,d$ must be tree edges for the tree to be connected. By definition of purple grid lines, since a and d are non-purple, they must lie in the interior of two parallel opposite polygon sides such that $g(a)\,\bar{g}(d)$ is a grid line parallel to ad. Thus, $T([a,d])$ can be set to be $T([g(a),\bar{g}(d)]) \cup \{a\,g(a),\ \bar{g}(d)\,d\}$. Combining the above two cases, $\tau([a,d])$ equals

$$\min\{\, i([a,d]),\ \tau([g(a),\bar{g}(d)]) + |a\,g(a)| + |\bar{g}(d)\,d|\,\}. \tag{5}$$

This formula can be evaluated in $O(1)$ time.

Case 2 : a or d is purple. There are three possible structures A, B, and C for $T([a,d])$ in this case. We derive formulae for $\tau([a,d])$ for the three possibilities and so we can take the minimum eventually. We will see that evaluating the formulae for possibilities A, B, and C takes $O(1)$, $O(k^2)$, and $O(k)$ time respectively, and hence our claimed result follows.

<u>*Possibility A*</u> : *$T([a,d])$ has no tree edge in $\mathsf{int}\,P$.*

Then $\tau([a,d])$ equals the length of $\mathsf{bd}\,[a,d]$.

<u>*Possibility B*</u> : *$T([a,d])$ contains a corner with corner-vertex v.*

Let the legs incident to the corner vertex v be bv and cv such that $[b,c] \subset \mathsf{bd}\,[a,d]$. Note that b and c must be purple vertices. Because of the corner, $T([a,d])$ contains $L(v,[b,c])$ as a subtree. Consider the other tree edges in $T([a,d]) - L(v,[b,c])$. It is impossible that there are two bodies ℓ_1 and ℓ_2 in $T([a,d]) - L(v,[b,c])$ such that ℓ_1 has an endpoint on $\mathsf{bd}\,[a,b]$ and is pinned to cv, and ℓ_2 has an endpoint on $\mathsf{bd}\,[c,d]$ and is pinned to bv. Otherwise, ℓ_1 and ℓ_2 will cross each other. Thus, in $T([a,d]) - L(v,[b,c])$, each body with an endpoint on $\mathsf{bd}\,[a,b]$ is pinned to bv or each body with an endpoint on $\mathsf{bd}\,[c,d]$ is pinned to cv (these two symmetric cases are not necessarily mutually exclusive). Figure 13 shows the three possible scenarios.

In Figure 13(a), $\tau([a,d]) = \ell(v,[b,c]) + j_c(v,b,[c,d]) + j_b(v,v,[a,b]) - 2|bv| - |cv|$. In Figure 13(b), $\tau([a,d]) = \ell(v,[b,c]) + j_b(v,c,[a,b]) + j_c(v,v,[c,d]) - |bv| - 2|cv|$. Trying all $O(k^2)$ possible locations for b, c, and v gives the following formula.

$$\min_{b,c,v} \left\{ \begin{array}{l} \ell(v,[b,c]) + j_c(v,b,[c,d]) + j_b(v,v,[a,b]) - 2|bv| - |cv| \\ \ell(v,[b,c]) + j_b(v,c,[a,b]) + j_c(v,v,[c,d]) - |bv| - 2|cv| \end{array} \right. \tag{6}$$

<u>*Possibility C*</u> : *$T([a,d])$ contains some tree edges in $\mathsf{int}\,P$ but it does not contain any corner.*

There are two symmetric cases where d is purple or a is purple. We will derive two formulae, formula (7) and formula (8), for the case where d is purple. They can be evaluated in $O(k)$

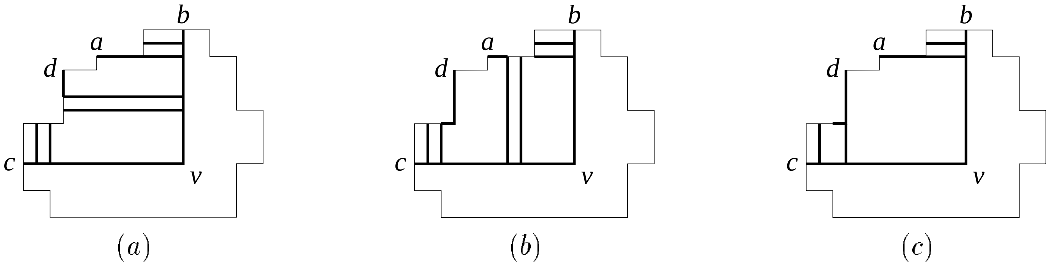


Figure 13. In (a), each body with an endpoint on $\mathsf{bd}\,[a,b]$ is pinned to bv, but some body with an endpoint on $\mathsf{bd}\,[c,d]$ may be pinned to bv. The figure in (b) shows the symmetric case. The figure in (c) shows that the two symmetric cases may occur simultaneously.

time. Similar analysis can be applied to the case where a is purple to obtain two formulae symmetric to formulae (7) and (8) that can be evaluated in $O(k)$ time. The minimum among these four formulae gives $\tau([a,d])$ for possibility C. We derive the formulae for the case where d is purple below.

We do a clockwise traversal of $\mathsf{bd}\,[a,d]$ starting at a and sooner or later, we will encounter an endpoint of a line ℓ in $T([a,d])$. In the absence of corner, ℓ in $T([a,d])$ must be a complete line or a body pinned to a head which must be a purple complete line. If ℓ is a body, then let ℓ be pinned to the purple complete line xy. If ℓ is a complete line, then let xy be ℓ itself and xy may or may not be purple. We define xy such that $[x,y] \subseteq \mathsf{bd}\,[a,d]$. Figures 14(a) and 14(b) show the two possible scenarios. Given our choice of xy, we observe that a tree path between

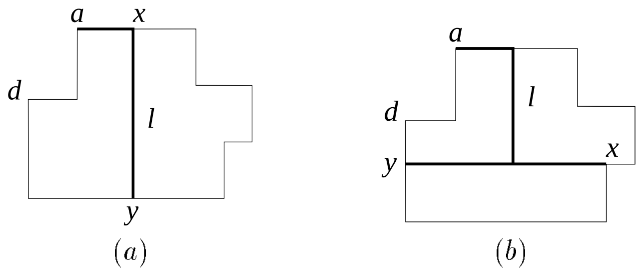


Figure 14. In the figures, the boundary traversed from a to the upper endpoint of ℓ in clockwise order is a sequence of tree edges in $T([a,d])$.

a and any tree vertex on $\mathsf{bd}\,[y,d]$ must pass through the interior of xy. If xy is purple, then by this observation, we can decompose $T([a,d])$ by splitting xy to yield $I([x,y])$, $J_x(y,y,[a,x])$, and $J_y(x,x,[y,d])$. This gives rise to the following formula for $\tau([a,d])$ that can be evaluated in $O(k)$ time.

$$\min\left\{ \begin{array}{l} i([x,y]) + j_x(y,y,[a,x]) \\ +j_y(x,x,[y,d]) - 2|xy| \end{array} \ : \ \text{purple grid line } xy, [x,y] \subseteq \mathsf{bd}\,[a,d] \right\} \tag{7}$$

If xy is non-purple, then xy is ℓ itself and no tree edge is pinned to the interior of xy. Thus, either $a = x$ or $a\,g(a)$ must be a tree edge by the way we identify ℓ. This implies that in this case, $\tau([a,d])$ equals

$$\min \begin{cases} \tau([y,d]) + i([a,y]), \text{ where } ay,\ y \in \mathsf{bd}\,[a,d], \text{ is a non-purple grid line} \\ \tau([g(a),d]) + |a\,g(a)| \end{cases} \tag{8}$$

Formula (8) can be evaluated in $O(1)$ time as there is at most one non-purple grid line incident to a.

6.6 Compute $j_a(v,w,[a,d])$ and $j_d(v,w,[a,d])$

We will discuss separately the computation of $j_a(v,v,[a,d])$ and $j_a(v,w,[a,d])$ for $w \neq v$. The computation of $j_d(v,v,[a,d])$ and $j_d(v,w,[a,d])$ is similar and we omit the discussion of it. It is essential to compute $j_a(v,v,[a,d])$ before $j_a(v,w,[a,d])$ for any $w \neq v$ as the latter computation depends on the availability of $j_a(v,v,[a,d])$. The same holds for $j_d(v,v,[a,d])$ and $j_d(v,w,[a,d])$.

Case 1: compute $j_a(v,v,[a,d])$. The main result in this case is stated below.

Lemma 9 *Computing $j_a(v,v,[a,d])$ takes $O(n)$ time when d is purple, and $O(k)$ time when d is non-purple. Similarly, computing $j_d(v,v,[a,d])$ takes $O(n)$ time when a is purple, and $O(k)$ time when a is non-purple.*

Recall that by definition, a is purple, $v \in \mathbb{P} - \mathsf{bd}\,[a,d]$, av is a line in $J_a(v,v,[a,d])$, and the degree of v is 1. There are two possible structures A and B for $J_a(v,v,[a,d])$ depending on whether there is a body e_d pinned to av and incident to d. For the case when e_d does not exist, we obtain formula (9). For the case when e_d exists, we obtain formula (10) for d purple and formula (11) for d non-purple. Formulae (9), (10), and (11) can be evaluated in $O(k)$, $O(n)$, and $O(1)$ time respectively. Hence, when d is purple, we evaluate formulae (9) and (10) and return the minimum to be $j_a(v,v,[a,d])$, and when d is non-purple, we evaluate formulae (9) and (11) and return the minimum. This establishes Lemma 9.

In the following, we analyze the two possibilities that e_d exists or not and derive the formulae.

<u>*Possibility A*</u> : *e_d does not exist.*

Proposition 2 *In $J_a(v,v,[a,d])$, if e_d does not exist, then it can be assumed that $\bar{g}(d)\,d$ is the only tree edge incident to d, or there is a purple tree vertex x on $\mathsf{bd}\,[a,d)$ such that a tree path from a to any tree vertex on $\mathsf{bd}\,[x,d]$ passes through x.*

Proof. If there is no body incident to av, then let x be a and the proposition is trivially true. Otherwise, among the bodies pinned to av, let e be the farthest one from a and let e be incident to grid vertex u on $\mathsf{bd}\,[a,d]$. By our assumption, $u \neq d$. If u is purple, then let x be u and we are done. If u is non-purple, then u must lie in the interior of a polygon side s_u parallel to av. Since $u \neq d$, $u\,g(u)$ must be a tree edge in order that the tree is connected. If $g(u) = d$, then $\bar{g}(d)\,d$ is a tree edge and we are done. If $g(u)$ is purple, then let x be $g(u)$ and we are done. Otherwise, we sweep e downward to become incident to $g(u)$, redefine $u = g(u)$, and repeat the above analysis again. The sweeping of e must terminate with $g(u) = d$ or $g(u)$ purple before or when $g(u)$ becomes an endpoint of s_u, or before the interior of e intersects $\mathsf{bd}\,(d,a)$, or before e becomes incident to v. In the first case, an endpoint of s_u is a boundary vertex which is purple by definition. See Figure 15(a). In the second case, if further sweeping will make e intersect

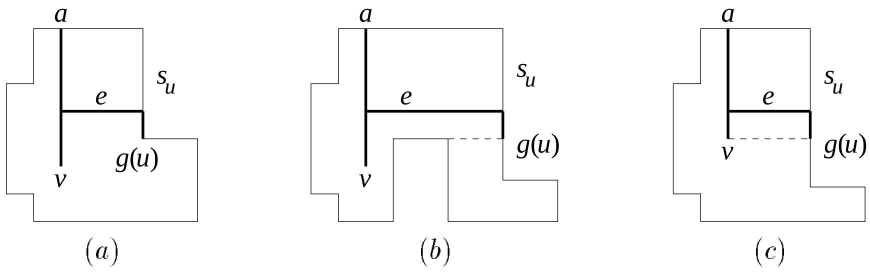


Figure 15.

$\mathsf{bd}\,(d, a)$, then e will overlap with a polygon side s parallel to e, and $g(u)$ is aligned with s. Thus, s makes $g(u)$ a purple vertex by definition. See Figure 15(b). In the third case, further sweeping will make e incident to v. If the first and second cases do not apply, then $g(u)$ is an endpoint of a grid line through v. See Figure 15(c). Since $v \in \mathbb{P}$, $g(u)$ is purple. ◻

By Proposition 2, if e_d does not exist, then $J_a(v, v, [a, d])$ can be constructed by adding the edge $\bar{g}(d)\, d$ to $J_a(v, v, [a, \bar{g}(d)])$, or $J_a(v, v, [a, d])$ can be split at a purple tree vertex x on $\mathsf{bd}\,[a, d)$ to yield $J_a(v, v, [a, x])$ and $T([x, d])$. Thus, $j_a(v, v, [a, d])$ equals

$$\min \left\{ \begin{array}{l} j_a(v, v, [a, \bar{g}(d)]) + |\bar{g}(d)\, d| \\ \min\{j_a(v, v, [a, x]) + \tau([x, d]) : \text{ purple vertex } x \in \mathsf{bd}\,[a, d)\}, \end{array} \right. \tag{9}$$

which can be computed in $O(k)$ time.

<u>*Possibility B*</u> : *e_d exists.*

If d is purple, then $J_a(v, v, [a, d])$ must be equal to the union of e_d, $J_a(v, v, [a, x])$, and $T([y, d])$ for some grid vertices x, y on $\mathsf{bd}\,[a, d]$, where $[a, x]$ contains the terminals in $[a, d] - [y, d]$. Trying all choices of y gives

$$\min\{j_a^*(v, v, [a, y)) + |e_d| + \tau([y, d]) : \text{ grid vertex } y \in \mathsf{bd}\,(a, d]\}, \tag{10}$$

which takes $O(n)$ time.

Suppose that d is non-purple. We have the following observation.

Proposition 3 *It can be assumed that e_d is the only tree edge incident to d.*

Proof. Since a is purple and d is non-purple, e_d is not incident to a and d lies in the interior of a polygon side parallel to av. So $\bar{g}(d)\, d$ is the only other possible tree edge incident to d. But if both e_d and $\bar{g}(d)\, d$ are present, we can sweep e_d to become incident to $\bar{g}(d)$ and $\bar{g}(d)\, d$ is then the only tree edge incident to d, which has been taken care of by formula (9). Thus, we can assume that e_d is the only tree edge incident to d. ◻

Proposition 3 implies the following formula when d is non-purple,

$$j_a^*(v, v, [a, d)) + |e_d|, \tag{11}$$

which can be evaluated in $O(1)$ time.

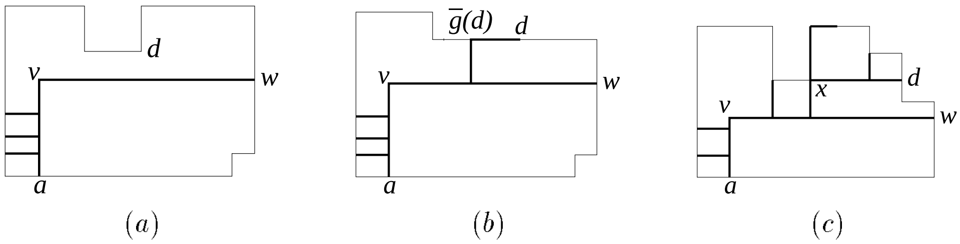


Figure 16. Three possible configurations of $J_a(v, w, [a, d])$ which is in bold line segments.

Case 2: compute $j_a(v, w, [a, d])$ when $w \neq v$. The main result to be established is stated below.

Lemma 10 *Suppose that $w \neq v$. Computing $j_a(v, w, [a, d])$ takes $O(n)$ time when d is purple, $O(n)$ time when d is non-purple and $\bar{g}(d)$ is purple, and $O(k)$ time otherwise. Similarly, computing $j_d(v, w, [a, d])$ takes $O(n)$ time when a is purple, $O(n)$ time when a is non-purple and $g(a)$ is purple, and $O(k)$ time otherwise.*

There are three scenarios depending on whether each body with an endpoint on $\mathsf{bd}\,[a, d]$ is pinned to av and whether there is a body e_d pinned to vw and incident to d. If each body with an endpoint on $\mathsf{bd}\,[a, d]$ is pinned to av (see Figure 16(a)), then $j_a(v, w, [a, d])$ must be equal to

$$j_a(v, v, [a, d]) + |vw|. \tag{12}$$

Formula (12) can be evaluated in $O(1)$ time. On the other hand, if there exists a body with an endpoint on $\mathsf{bd}\,[a, d]$ but pinned to vw instead, then whether e_d exists or not makes all the difference. As in Case 1, there will be two corresponding possible structures for $J_a(v, w, [a, d])$. For the case when e_d does not exist, we obtain formula (13). For the case when e_d exists, we obtain formula (14) for d purple, formula (15) for d and $\bar{g}(d)$ non-purple, and formula (16) for d non-purple and $\bar{g}(d)$ purple. Formulae (13), (14), (15), and (16) can be evaluated in $O(k)$, $O(n)$, $O(1)$, and $O(n)$ time respectively. Hence, when d is purple, we evaluate formulae (12), (13), and (14); when d is non-purple and $\bar{g}(d)$ is purple, we evaluate formulae (12), (13), and (16); and when d and $\bar{g}(d)$ are non-purple, we evaluate formulae (12), (13), and (15). This establishes Lemma 10.

In the following, we analyze the two possibilities that e_d exists or not and derive formulae (13), (14), (15), and (16). The analysis is very similar to that in Case 1 and so we omit proofs of some claims.

Possibility A : e_d does not exist.

Proposition 4 *In $J_a(v, w, [a, d])$, if e_d does not exist, then it can be assumed that $\bar{g}(d)\,d$ is the only tree edge incident to d, or there is a purple tree vertex x on $\mathsf{bd}\,[a, d)$ such that a tree path from a to any tree vertex on $\mathsf{bd}\,[x, d]$ passes through x.*

See Figures 16(b) and 16(c) for illustrations of the two possibilities in Proposition 4. Proposition 4 is similar to Proposition 2 and can be proved by similar analysis. By Proposition 4,

$j_a(v, w, [a, d])$ equals

$$\min \begin{cases} j_a(v, w, [a, \bar{g}(d)]) + |\bar{g}(d)\, d| \\ \min\{j_a(v, w, [a, x]) + \tau([x, d]) : \text{ purple vertex } x \in \mathsf{bd}\,[a, d)\}, \end{cases} \tag{13}$$

which can be evaluated in $O(k)$ time.

Possibility B : e_d exists.

If d is purple, then $J_a(v, w, [a, d])$ must be equal to the union of e_d, $J_a(v, w, [a, x])$, and $T([y, d])$ for some grid vertices x, y on $\mathsf{bd}\,[a, d]$, where $[a, x]$ contains the terminals in $[a, d] - [y, d]$. Trying all choices of y gives

$$\min\{j_a^*(v, w, [a, y)) + |e_d| + \tau([y, d]) : \text{ grid vertex } y \in \mathsf{bd}\,(a, d]\}, \tag{14}$$

which can be evaluated in $O(n)$ time.

Suppose that d is non-purple. If $\bar{g}(d)$ is non-purple, then we can assume that e_d is the only tree edge incident to d. Otherwise, we can sweep e_d to become incident to $\bar{g}(d)$ and reduce this case to possibility A. Thus, $j_a(v, w, [a, d])$ equals

$$j_a^*(v, w, [a, d)) + |e_d|, \tag{15}$$

which can be evaluated in $O(1)$ time. If $\bar{g}(d)$ is purple and $\bar{g}(d)\, d$ is not a tree edge, e_d is still the only tree edge incident to d. If $\bar{g}(d)$ is purple and $\bar{g}(d)\, d$ is a tree edge, then we may not be able to sweep e_d because this may make e_d incident to v which violates the topology constraint that the degree of v is at most 2. Instead, we split at the tree vertex $\bar{g}(d)$ to obtain $J_a^*(v, w, [a, x])$, $T([y, \bar{g}(d)])$, $\bar{g}(d)\, d$, and e_d, where x and y are some grid vertices on $\mathsf{bd}\,[a, \bar{g}(d)]$ such that $[a, x]$ contains the terminals in $[a, d] - [y, d]$. Thus, we obtain the following formula

$$\min \begin{cases} j_a^*(v, w, [a, d)) + |e_d| \\ \min\{\, j_a^*(v, w, [a, y)) + \tau([y, \bar{g}(d)]) + |\bar{g}(d)\, d| + |e_d| : \text{grid vertex } y \in \mathsf{bd}\,(a, \bar{g}(d)]\,\} \end{cases} \tag{16}$$

which can be evaluated in $O(n)$ time.

7 Summary and Discussion

In Table 1, the first column shows the various types of tree lengths computed by our dynamic programming algorithm. The second column shows the total number of tree lengths for each type and the computing time needed per tree length. The time needed to compute each tree length is from the results in Section 6. From Table 1 and Lemma 6, it is clear that the total running time of our algorithm is $O(k^3 n)$.

Our time complexity analysis assumes that $k = O(n)$. A more precise analysis reveals that the running time is $O(k^4 + k^3 n)$. When $k \gg n$, it becomes $O(k^4)$ which is too large when compared with the value of n. We suspect that this can be handled by first finding the *relative hull* of the terminals. A relative hull can be obtained by connecting every pair of neighboring terminals on $\mathsf{bd}\, P$ by a shortest rectilinear shortest path in P. The minimal Steiner tree desired should lie inside this relative hull and so we can work on this relative hull instead of P. In general, the relative hull bounds a set of disconnected open regions and we only need to run the dynamic programming algorithm on each open region. By finding the relative hull that minimizes the sum of the area of the open regions, we think it is likely that the complexity of the boundaries of the open regions will be $O(n)$. A rigorous investigation of the correctness and complexity of this strategy is a future research problem.

Table 1.

Types	(Total no. for each type, computing time per value)
$j_a^*(v,w,[a,d))$	$(O(k^2n), O(1))$
$j_d^*(v,w,(a,d])$	$(O(k^2n), O(1))$
$\tau^*([a,d))$	$(O(kn), O(1))$
$\tau^*((a,d])$	$(O(kn), O(1))$
$i([a,d])$	$(O(k), O(n))$, if ad is purple $(O(k), O(n))$, if ad is non-purple and $g(a)\,\bar{g}(d)$ is purple $(O(n), O(1))$, otherwise
$\ell(v,[a,d])$	$(O(k^2), O(n))$
$\tau([a,d])$	$(O(n), O(1))$, if ad is a non-purple grid line $(O(kn), O(k^2))$, otherwise
$j_a(v,v,[a,d])$	$(O(k^3), O(n))$, if d is purple $(O(k^2n), O(k))$, if d is non-purple
$j_d(v,v,[a,d])$	$(O(k^3), O(n))$, if a is purple $(O(k^2n), O(k))$, if a is non-purple
$j_a(v,w,[a,d])$	$(O(k^3), O(n))$, if d is purple $(O(k^3), O(n))$, if d is non-purple and $\bar{g}(d)$ is purple $(O(k^2n), O(k))$, otherwise
$j_d(v,w,[a,d])$	$(O(k^3), O(n))$, if a is purple $(O(k^3), O(n))$, if a is non-purple and $g(a)$ is purple $(O(k^2n), O(k))$, otherwise

References

[1] P.K. Agarwal and M.T. Shing, Algorithms for special cases of rectilinear Steiner trees. I. Points on the boundary of a rectilinear rectangle. *Networks*, 20 (1990), pp. 453-485.

[2] M. Bern, Faster exact algorithms for Steiner tree in planar networks. *Networks*, 7 (1990), pp. 109–120.

[3] S.W. Cheng, A. Lim, and C.T. Wu, Optimal rectilinear Steiner tree for extremal point sets. *Proceedings of the Fourth Annual Symposium on Algorithms and Computation (ISAAC)*, 1993, LNCS 762, pp. 523–532.

[4] S.W. Cheng and C.K. Tang, A fast algorithm for computing optimal rectilinear Steiner tree for extremal point sets. *Proceedings of the Sixth Annual International Symposium on Algorithms and Computation (ISAAC)*, 1995, LNCS 1004, pp. 322–331.

[5] J. Cohoon, D. Richards, and J. Salowe, An optimal steiner tree algorithm for a net whose terminals lie on the perimeter of a rectangle. *IEEE Transactions On Computer-Aided Design*, 9 (1990), pp. 398–407.

[6] R. Erickson, C. Monma, and A. Veinott, Send and split methods for minimum-concave-cost network flows. *Math. of O.R.*, 12 (1987), pp. 70–81.

[7] M. Kaufmann, S. Gao, and K. Thulasiraman, On Steiner minimal trees in grid graphs and its application to VLSI routing. *Proceedings of International Symposium on Algorithms and Computation (ISAAC)*, 1994, LNCS 834, pp. 351–359.

[8] C.E. Leiserson and F.M. Maley, Algorithms for Routing and Testing Routability of Planar VLSI Layouts. *Proceedings of Seventh Annual Symposium on Theory of Computing*, 1985, pp. 69–78.

[9] F.M. Maley, *Single-Layer Wire Routing and Compaction*, MIT Press, 1989.

[10] J. Provan, Convexity and the steiner tree problem. *Networks*, 18 (1988), pp. 55–72.

[11] D. Richard and J. Salowe, A linear-time algorithm to construct a rectilinear Steiner minimal tree for k-extremal point sets. *Algorithmica*, 7 (1992), pp. 247–276.