

Common Tangents of Two Disjoint Polygons in Linear Time and Constant Workspace

MIKKEL ABRAHAMSEN, University of Copenhagen, Denmark

BARTOSZ WALCZAK, Jagiellonian University, Poland

We provide a remarkably simple algorithm to compute all (at most four) common tangents of two disjoint simple polygons. Given each polygon as a read-only array of its corners in cyclic order, the algorithm runs in linear time and constant workspace and is the first to achieve the two complexity bounds simultaneously. The set of common tangents provides basic information about the convex hulls of the polygons—whether they are nested, overlapping, or disjoint—and our algorithm thus also decides this relationship.

1 INTRODUCTION

A tangent of a polygon is a line touching the polygon such that all of the polygon lies on the same side of the line. We consider the problem of computing the common tangents of two disjoint polygons that are simple, that is, they have no self-intersections. The set of common tangents provides basic information about the convex hulls of the polygons—whether they are disjoint, overlapping, or nested. We call a common tangent *outer* if the two polygons lie on the same side of it and *separating* otherwise. Two disjoint polygons have two outer common tangents unless their convex hulls are nested, and if they are properly nested, then there is no outer common tangent. Two polygons have two separating common tangents unless their convex hulls overlap, and if they properly overlap, then there is no separating common tangent. See Figures 1–3 for illustrations. Common tangents arise in many different contexts, for instance in problems related to convex hulls [Preparata and Hong 1977], shortest paths [Guibas and Hershberger 1989], ray shooting [Hershberger and Suri 1995], and clustering [Abrahamsen et al. 2017].

We provide a very simple algorithm to compute the common tangents of two disjoint simple polygons. In view of the above, the algorithm also determines whether the two polygons have (properly) nested, (properly) overlapping, or disjoint convex hulls. Given each of the two polygons as a read-only array of its corners in cyclic order, our algorithm runs in linear time and uses seven variables each storing a boolean value or an index of a corner in one of the arrays. The algorithm is therefore asymptotically optimal with respect to time and workspace, and it operates in the *constant workspace model* of computation.

The constant workspace model is a restricted version of the RAM model in which the input is read-only, the output is write-only, and only $O(\log b)$ additional bits of *workspace* (with both read and write access) are available, where b denotes the bit length of the input. It is natural to consider algorithms in this model as memory-optimal, because $\Omega(\log b)$ bits are required to store a pointer to an entry in the input. Since blocks of $\Theta(\log b)$ bits are considered to form *words* in the memory, algorithms in the constant workspace model use $O(1)$ words of workspace, which explains the name of the model. (Likewise, time complexity is usually measured with respect to the number of words in the input, with the assumption that arithmetic operations on words can be performed in

A journal version of this paper appeared in *ACM Trans. Algorithms* 15, 1 (2018), 12:1–12:21, doi:10.1145/3284355.

Preliminary expositions of the results contained in this paper appeared at SoCG 2015 [Abrahamsen 2015] and ESA 2016 [Abrahamsen and Walczak 2016].

Mikkel Abrahamsen was partially supported by Danish Council for Independent Research grant DFF-0602-02499B.

Bartosz Walczak was partially supported by National Science Center of Poland grant 2015/17/D/ST1/00585.

Authors' addresses: Mikkel Abrahamsen, Department of Computer Science, University of Copenhagen, Denmark, e-mail: miab@di.ku.dk; Bartosz Walczak, Department of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland, e-mail: walczak@tcs.uj.edu.pl.

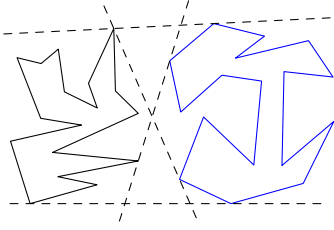


Fig. 1. The convex hulls are disjoint—outer and separating common tangents exist.

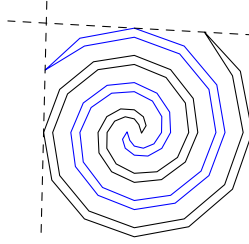


Fig. 2. The convex hulls overlap—only outer common tangents exist.

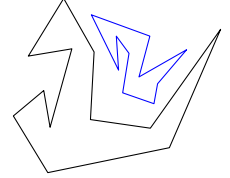


Fig. 3. The convex hulls are nested—no common tangents exist.

constant time.) The practical relevance of studying problems in the constant workspace model is increasing, as there are many current and emerging memory technologies where writing can be much more expensive than reading in terms of time and energy [Carson et al. 2016].

The constant workspace model was first considered explicitly for geometric problems by Asano et al. [2011]. Recently, there has been growing interest in algorithms for geometric problems using constant or restricted workspace. We refer the reader to the recent survey by Banyassady et al. [2018] for an overview of the results. In complexity theory, the class of decision problems solvable using constant workspace is usually denoted by L . The constant workspace model was shown to be surprisingly powerful—for instance, the problem of deciding whether two vertices of an undirected graph lie in the same connected component belongs to L [Reingold 2008].

The problem of computing common tangents of two polygons has received much attention in the special case that the polygons are convex. For instance, computing the outer common tangents of disjoint convex polygons is used as a subroutine in the classical divide-and-conquer algorithm for the convex hull of a set of n points in the plane due to Preparata and Hong [1977]. They gave a naive linear-time algorithm for outer common tangents, which suffices for an $O(n \log n)$ -time convex hull algorithm. The problem is also considered in various dynamic convex hull algorithms [Brodal and Jacob 2002; Hershberger and Suri 1992; Overmars and van Leeuwen 1981]. Overmars and van Leeuwen [1981] gave an $O(\log n)$ -time algorithm for computing an outer common tangent of two disjoint convex polygons when a separating line is known, where each polygon has at most n corners. Kirkpatrick and Snoeyink [1995] gave an $O(\log n)$ -time algorithm for the same problem but without using a separating line. Guibas et al. [1991] gave a lower bound of $\Omega(\log^2 n)$ on the time required to compute an outer common tangent of two intersecting convex polygons even when they are known to intersect in at most two points. They also described an algorithm achieving that bound. Toussaint [1983] considered the problem of computing separating common tangents of convex polygons. He gave a linear-time algorithm using the technique of “rotating calipers”. Guibas et al. [1991] gave an $O(\log n)$ -time algorithm for the same problem. All the above-mentioned algorithms with sublinear running times make essential use of convexity of the polygons. If the polygons are not convex, a linear-time algorithm can be used to compute the convex hulls before computing the tangents. Many such algorithms have been described, and the one due to Melkman [1987] is usually considered the simplest. However, if the polygons are given in read-only memory, then $\Omega(n)$ extra bits are required to store the convex hulls, so this approach does not work in the constant workspace model.

In the following, we provide a brief description of our algorithm, which is presented in full detail using the pseudocode in Algorithm 2 on page 6. Algorithm 1 on page 6 is a simplified version of

Algorithm 2, which finds the separating common tangents whenever they exist, but is guaranteed to find the outer common tangents only when the convex hulls of the two polygons are disjoint.

In order to find a particular common tangent of two polygons P_0 and P_1 , we maintain a pair of corners of support $q_0 \in P_0$ and $q_1 \in P_1$, where the line $\mathcal{L}(q_0, q_1)$ passing through q_0 and q_1 is considered as a candidate for the requested common tangent. In each step of the algorithm, we traverse each polygon in order to find a corner that does not lie on the “correct side” of the candidate line $\mathcal{L}(q_0, q_1)$, that is, on the side where all of the polygon should lie if $\mathcal{L}(q_0, q_1)$ was the requested common tangent. Each polygon is traversed starting from its current corner of support in a direction determined by the type of the tangent that we aim to find—the separating common tangents are computed by traversing both polygons in the same direction, whereas the outer common tangents are computed by choosing opposite directions (see Figure 4 on page 6). If both polygons lie entirely on the “correct side” of $\mathcal{L}(q_0, q_1)$, then we return $\mathcal{L}(q_0, q_1)$ as the solution. Otherwise, when we first encounter an edge e of one of the polygons, say P_0 , that ends at a corner q'_0 on the “wrong side”, we distinguish two cases, determined by where e intersects the line $\mathcal{L}(q_0, q_1)$ with respect to q_0 and q_1 . If q_1 does *not* lie between q_0 and the intersection point of e and $\mathcal{L}(q_0, q_1)$, then q'_0 becomes the new corner of support of P_0 for the next step of the algorithm (the corner of support of P_1 remains at q_1). Otherwise, q_1 lies in the convex hull of P_0 and therefore cannot be a support of a common tangent. In the latter case, we temporarily block q_0 from further updates until the first update to q_1 . If no update to q_1 occurs before a full traversal of P_1 , we conclude that the convex hulls are nested and no common tangents exist. The key observation is that if the requested common tangent exists, then it must be found before either polygon has been fully traversed for the second time by its corner of support. Therefore, if an update occurs during the third full traversal of a polygon, we conclude that the common tangent does not exist.

In order to guarantee a linear bound on the total running time, in each step, the search for a corner on the “wrong side” is performed by a tandem walk on the two polygons. That is, we traverse both polygons starting from the current corners of support and advancing alternately by one edge until finding the first corner on the “wrong side” in either of the two polygons—that corner becomes the start of the next search on that polygon (as the new corner of support), while the search on the other polygon is reverted to where it started. To our knowledge, the idea of a tandem walk was first applied by Even and Shiloach [1981] to a problem not related to geometry.

Barba et al. [2015] describe a linear-time constant-workspace algorithm, attributed to A. Pilz, for the following problem: given a simple polygonal chain P with endpoints on its convex hull, and given a line L that separates the two endpoints of P , find the two edges of the convex hull of P that are crossed by L . It applies an analogous principle of updating the candidate line by parallel traversal of two independent parts of P . These updates, however, make the points of support of the candidate line move only towards the endpoints of P , never coming back to points visited before. On the other hand, our algorithm sometimes needs to make more than one full traversal of a polygon (but never more than two) in order to find the requested common tangent, which makes its analysis significantly more involved. An easy adaptation of Pilz’s algorithm can be used to find the outer common tangents of two polygons that are separated by a given line. However, finding such a line seems to be no easier than computing the separating common tangents.

The rest of the paper is organized as follows. In Section 2, we introduce the terminology and conventions used throughout the paper and state some well-known properties of the common tangents of two polygons. In Section 3, we describe two algorithms for computing the common tangents if they exist or detecting that they do not exist, where one is a simplified version of the other for special cases indicated before. Section 4 contains proofs that the algorithms work correctly under the assumption of a crucial lemma, which is then proved in Section 5. We conclude

in Section 6 by discussing how to avoid the general position assumption of Sections 2–5 and by suggesting some related open problems for future research.

2 BASIC TERMINOLOGY AND NOTATION

For any two points a and b in the plane, the closed line segment with endpoints a and b is denoted by ab . When $a \neq b$, the line passing through a and b is denoted by $\mathcal{L}(a, b)$. The segment ab and the line $\mathcal{L}(a, b)$ are considered *oriented* in the direction from a towards b . A *simple polygon* or just a *polygon* with *corners* a_0, \dots, a_{n-1} , denoted by $\mathcal{P}(a_0, \dots, a_{n-1})$, is a closed curve in the plane composed of n edges $a_0a_1, \dots, a_{n-2}a_{n-1}, a_{n-1}a_0$ that have no common points other than the common endpoints of pairs of edges consecutive in that cyclic order. The polygon $\mathcal{P}(a_0, \dots, a_{n-1})$ is considered *oriented* so that its *forward* traversal visits corners a_0, \dots, a_{n-1} in this cyclic order. A *polygonal region* is a closed and bounded region of the plane whose boundary is a polygon. For any two points $a = (a_x, a_y)$ and $b = (b_x, b_y)$ in \mathbb{R}^2 , we let

$$\det(a, b) = \begin{vmatrix} a_x & b_x \\ a_y & b_y \end{vmatrix} = a_x b_y - b_x a_y.$$

For $a_0, \dots, a_{n-1} \in \mathbb{R}^2$, we let

$$\det^*(a_0, \dots, a_{n-1}) = \det(a_0, a_1) + \dots + \det(a_{n-2}, a_{n-1}) + \det(a_{n-1}, a_0).$$

In particular, for any three points $a = (a_x, a_y)$, $b = (b_x, b_y)$, and $c = (c_x, c_y)$ in \mathbb{R}^2 , we have

$$\det^*(a, b, c) = \begin{vmatrix} a_x & b_x \\ a_y & b_y \end{vmatrix} + \begin{vmatrix} b_x & c_x \\ b_y & c_y \end{vmatrix} + \begin{vmatrix} c_x & a_x \\ c_y & a_y \end{vmatrix} = \begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{vmatrix}.$$

For two distinct points a and b in the plane, the *left side* and the *right side* of an oriented line $\mathcal{L}(a, b)$ are the two closed half-planes LHP(a, b) and RHP(a, b), respectively, defined as follows:

$$\text{LHP}(a, b) = \{c \in \mathbb{R}^2 : \det^*(a, b, c) \geq 0\},$$

$$\text{RHP}(a, b) = \{c \in \mathbb{R}^2 : \det^*(a, b, c) \leq 0\},$$

where LHP/RHP stands for “left/right half-plane”. An oriented polygon $\mathcal{P}(a_0, \dots, a_{n-1})$ is *counterclockwise* when $\det^*(a_0, \dots, a_{n-1}) > 0$ and *clockwise* when $\det^*(a_0, \dots, a_{n-1}) < 0$.

We assume for the rest of this paper that P_0 and P_1 are two disjoint simple polygons with n_0 and n_1 corners, respectively, each defined by a read-only array of its corners:

$$P_0 = \mathcal{P}(p_0[0], \dots, p_0[n_0 - 1]), \quad P_1 = \mathcal{P}(p_1[0], \dots, p_1[n_1 - 1]).$$

We make no assumption (yet) on whether P_0 and P_1 are oriented counterclockwise or clockwise. We further assume that the corners of P_0 and P_1 are in general position, that is, P_0 and P_1 have no corners in common and the combined set of corners $\{p_0[0], \dots, p_0[n_0 - 1], p_1[0], \dots, p_1[n_1 - 1]\}$ contains no triple of collinear points. This assumption simplifies the description and the analysis of the algorithm but can be avoided, as we explain in the last section. We do not assume the polygonal regions bounded by P_0 and P_1 to be disjoint—they may be nested. Indices of the corners of each P_k are considered modulo n_k , so that $p_k[i]$ and $p_k[j]$ denote the same corner when $i \equiv j \pmod{n_k}$.

A *tangent* of P_k is a line L such that P_k has a common point with L and is contained in one of the two closed half-planes determined by L . A line L is a *common tangent* of P_0 and P_1 if it is a tangent of both P_0 and P_1 ; it is an *outer common tangent* if P_0 and P_1 lie on the same side of L and a *separating common tangent* otherwise. The following lemma asserts well-known properties of common tangents of polygons. See Figures 1–3.

LEMMA 2.1. *A line is a tangent of a polygon P if and only if it is a tangent of the convex hull of P . Moreover, under the general position assumption, the following holds:*

- P_0 and P_1 have no common tangents if the convex hulls of P_0 and P_1 are nested;
- P_0 and P_1 have two outer common tangents and no separating common tangents if the convex hulls of P_0 and P_1 properly overlap;
- P_0 and P_1 have two outer common tangents and two separating common tangents if the convex hulls of P_0 and P_1 are disjoint.

3 ALGORITHMS

We distinguish four cases of the common tangent problem: find the pair of indices (s_0, s_1) such that

- (1) $P_0 \subset \text{RHP}(p_0[s_0], p_1[s_1])$ and $P_1 \subset \text{RHP}(p_0[s_0], p_1[s_1])$,
- (2) $P_0 \subset \text{LHP}(p_0[s_0], p_1[s_1])$ and $P_1 \subset \text{LHP}(p_0[s_0], p_1[s_1])$,
- (3) $P_0 \subset \text{RHP}(p_0[s_0], p_1[s_1])$ and $P_1 \subset \text{LHP}(p_0[s_0], p_1[s_1])$,
- (4) $P_0 \subset \text{LHP}(p_0[s_0], p_1[s_1])$ and $P_1 \subset \text{RHP}(p_0[s_0], p_1[s_1])$.

The line $\mathcal{L}(p_0[s_0], p_1[s_1])$ is an outer common tangent in cases 1–2 and a separating common tangent in cases 3–4. We say that (s_0, s_1) is the *solution* to the particular case of the problem. An algorithm solving each case is expected to find the solution (s_0, s_1) if it exists (i.e. the convex hulls of P_0 and P_1 are not nested in cases 1–2 and are disjoint in cases 3–4) and to report “no solution” otherwise.

We will describe two general algorithms. Algorithm 1, very simple, fully solves the separating common tangent problem (cases 3–4), finding the separating common tangent if the convex hulls of P_0 and P_1 are disjoint and otherwise reporting that the requested tangent does not exist. Furthermore, Algorithm 1 solves the outer common tangent problem (cases 1–2) provided that the convex hulls of P_0 and P_1 are disjoint. Algorithm 1 also correctly reports that the outer common tangents do not exist if the convex hulls of P_0 and P_1 are nested. However, Algorithm 1 can fail to find the outer common tangents if the convex hulls of P_0 and P_1 properly overlap. Algorithm 2 is an improved version of Algorithm 1 that solves the problem correctly in all cases.

The general idea behind either algorithm is as follows. The algorithm maintains a pair of indices (s_0, s_1) called the *candidate solution*, which determines the line $\mathcal{L}(p_0[s_0], p_1[s_1])$ called the *candidate line*. If each of the two polygons lies on the “correct side” of the candidate line, which is either $\text{RHP}(p_0[s_0], p_1[s_1])$ or $\text{LHP}(p_0[s_0], p_1[s_1])$ depending on the particular case of 1–4 to be solved, then the algorithm returns (s_0, s_1) as the requested solution. Otherwise, for some $u \in \{0, 1\}$, the algorithm finds an index v_u such that $p_u[v_u]$ lies on the “wrong side” of the candidate line, updates s_u by setting $s_u \leftarrow v_u$, and repeats. This general scheme guarantees that if (s_0, s_1) is claimed to be the solution, then it indeed is. However, the algorithm can fall in an infinite loop—when there is no solution or when the existing solution keeps being missed. A detailed implementation of the scheme must guarantee that the solution is found in linearly many steps if it exists. Then, if the solution is not found in the guaranteed number of steps, the algorithm terminates and reports “no solution”.

The particular case of 1–4 to be solved is specified to the algorithms by providing two binary parameters $\alpha_0, \alpha_1 \in \{+1, -1\}$ specifying that the final solution (s_0, s_1) should satisfy

$$\begin{aligned} P_0 \subset \text{RHP}(p_0[s_0], p_1[s_1]) & \quad \text{if } \alpha_0 = +1, & P_0 \subset \text{LHP}(p_0[s_0], p_1[s_1]) & \quad \text{if } \alpha_0 = -1, \\ P_1 \subset \text{RHP}(p_0[s_0], p_1[s_1]) & \quad \text{if } \alpha_1 = +1, & P_1 \subset \text{LHP}(p_0[s_0], p_1[s_1]) & \quad \text{if } \alpha_1 = -1. \end{aligned}$$

For clarity, instead of using the parameters α_0 and α_1 explicitly, the pseudocode uses half-planes $\mathcal{H}_0(a, b)$ and $\mathcal{H}_1(a, b)$ defined as follows, for any $k \in \{0, 1\}$ and any distinct $a, b \in \mathbb{R}^2$:

$$\mathcal{H}_k(a, b) = \{c \in \mathbb{R}^2 : \alpha_k \det^*(a, b, c) \leq 0\} = \begin{cases} \text{RHP}(a, b) & \text{if } \alpha_k = +1, \\ \text{LHP}(a, b) & \text{if } \alpha_k = -1. \end{cases}$$

Algorithm 1:

```

1  $s_0 \leftarrow 0; \quad v_0 \leftarrow 0; \quad s_1 \leftarrow 0; \quad v_1 \leftarrow 0; \quad u \leftarrow 0$ 
2 while  $s_0 < 2n_0$  and  $s_1 < 2n_1$  and  $(v_0 < s_0 + n_0$  or  $v_1 < s_1 + n_1)$ 
3    $v_u \leftarrow v_u + 1$ 
4   if  $p_u[v_u] \notin \mathcal{H}_u(p_0[s_0], p_1[s_1])$ 
5      $s_u \leftarrow v_u; \quad v_{1-u} \leftarrow s_{1-u}$ 
6    $u \leftarrow 1 - u$ 
7 if  $s_0 \geq 2n_0$  or  $s_1 \geq 2n_1$ 
8   return "no solution"
9 return  $(s_0, s_1)$ 

```

See Figure 4 for the meaning of $\mathcal{H}_u(a, b)$ and the assumed orientations of P_0 and P_1 .

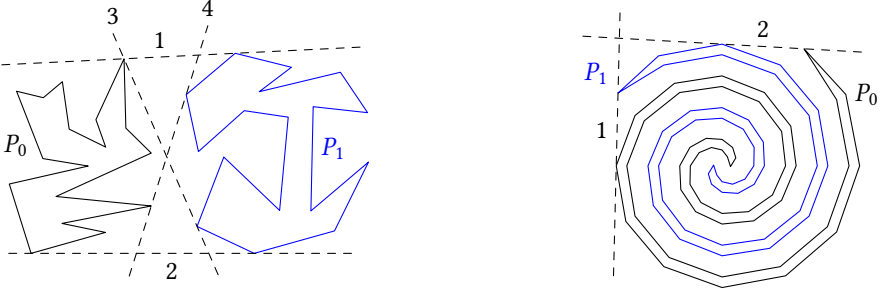
Algorithm 2:

```

1  $s_0 \leftarrow 0; \quad v_0 \leftarrow 0; \quad b_0 \leftarrow \text{false}; \quad s_1 \leftarrow 0; \quad v_1 \leftarrow 0; \quad b_1 \leftarrow \text{false}; \quad u \leftarrow 0$ 
2 while  $s_0 < 2n_0$  and  $s_1 < 2n_1$  and  $(v_0 < s_0 + n_0$  or  $v_1 < s_1 + n_1)$ 
3    $v_u \leftarrow v_u + 1$ 
4   if  $p_u[v_u] \notin \mathcal{H}_u(p_0[s_0], p_1[s_1])$  and not  $b_u$ 
5     if  $p_{1-u}[s_{1-u}] \in \Delta(p_u[s_u], p_u[v_u - 1], p_u[v_u])$ 
6        $b_u \leftarrow \text{true}$ 
7     else
8        $s_u \leftarrow v_u; \quad v_{1-u} \leftarrow s_{1-u}; \quad b_{1-u} \leftarrow \text{false}$ 
9    $u \leftarrow 1 - u$ 
10 if  $s_0 \geq 2n_0$  or  $s_1 \geq 2n_1$  or  $b_0$  or  $b_1$ 
11   return "no solution"
12 return  $(s_0, s_1)$ 

```

See Figure 4 for the meaning of $\mathcal{H}_u(a, b)$ and the assumed orientations of P_0 and P_1 ; $\Delta(a, b, c)$ is the triangle spanned by a, b, c .



	$\mathcal{H}_0(a, b)$	$\mathcal{H}_1(a, b)$	orientation of P_0	orientation of P_1
1	RHP(a, b)	RHP(a, b)	counterclockwise	clockwise
2	LHP(a, b)	LHP(a, b)	clockwise	counterclockwise
3	RHP(a, b)	LHP(a, b)	clockwise	clockwise
4	LHP(a, b)	RHP(a, b)	counterclockwise	counterclockwise

Fig. 4. The meaning of $\mathcal{H}_0(a, b)$ and $\mathcal{H}_1(a, b)$ and the assumed orientations of P_0 and P_1 in the pseudocodes of Algorithm 1 and Algorithm 2, depending on which common tangent of 1–4 is requested.

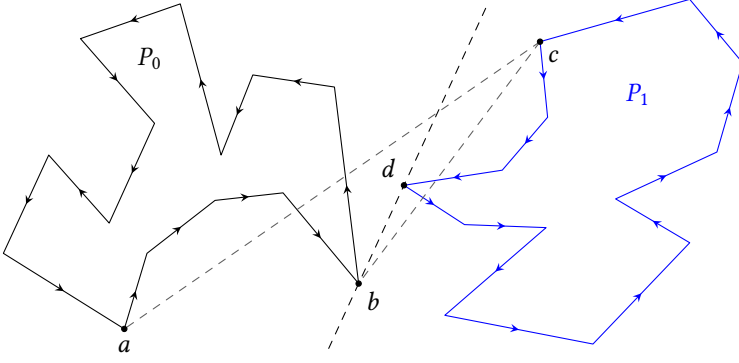


Fig. 5. An example of how Algorithm 1 finds the separating common tangent $\mathcal{L}(b, d)$ of P_0 and P_1 starting from $(p_0[0], p_1[0]) = (a, c)$. The segments $p_0[s_0]p_1[s_1]$ on intermediate candidate lines are also shown.

The final solution (s_0, s_1) should satisfy $P_k \subset \mathcal{H}_k(p_0[s_0], p_1[s_1])$. A test of the form $c \notin \mathcal{H}_k(a, b)$ in the pseudocode should be understood as testing whether $\alpha_k \det^*(a, b, c) > 0$. Another assumption that we make when presenting the pseudocode concerns the direction in which each polygon P_k is traversed in order to find an index v_k such that $p_k[v_k] \notin \mathcal{H}_k(p_0[s_0], p_1[s_1])$. For a reason that will become clear later when we analyze correctness of the algorithms, we require that

- P_0 is traversed counterclockwise when $\alpha_1 = +1$ and clockwise when $\alpha_1 = -1$,
- P_1 is traversed clockwise when $\alpha_0 = +1$ and counterclockwise when $\alpha_0 = -1$.

In the pseudocode, the forward orientation of P_k is *assumed* to be the one in which the corners of P_k should be traversed according to the conditions above. When this has not been guaranteed in the problem setup, a reference to a corner of P_k of the form $p_k[i]$ in the pseudocode should be understood as $p_k[\beta_k i]$ for the constant $\beta_k \in \{+1, -1\}$ computed as follows at the very beginning:

$$\beta_0 = \alpha_1 \operatorname{sgn} \det^*(p_0[0], \dots, p_0[n_0 - 1]), \quad \beta_1 = -\alpha_0 \operatorname{sgn} \det^*(p_1[0], \dots, p_1[n_1 - 1]).$$

The assumptions made in the pseudocode of the two algorithms for each particular case of 1–4 are summarized in Figure 4.

Algorithm 1 maintains a candidate solution (s_0, s_1) starting from $(s_0, s_1) = (0, 0)$. At the beginning and after each update to (s_0, s_1) , the algorithm traverses P_0 and P_1 in parallel with indices (v_0, v_1) , starting from $(v_0, v_1) = (s_0, s_1)$ and advancing v_0 and v_1 alternately. The variable $u \in \{0, 1\}$ determines the polygon P_u in which the traversal is advanced in the current iteration. If the test in line 4 of Algorithm 1 succeeds, that is, the corner $p_u[v_u]$ lies on the “wrong side” of the candidate line, then the algorithm updates the candidate solution by setting $s_u \leftarrow v_u$ and reverts v_{1-u} back to s_{1-u} in line 5. The algorithm returns (s_0, s_1) when both polygons have been entirely traversed with indices v_0 and v_1 without detecting any corner on the “wrong side” of the candidate line. This can happen only when $P_0 \subset \mathcal{H}_0(p_0[s_0], p_1[s_1])$ and $P_1 \subset \mathcal{H}_1(p_0[s_0], p_1[s_1])$, as required.

See Figure 5 for an example run of Algorithm 1 for the separating common tangent problem (case 4). The following theorem asserts that Algorithm 1 is correct for the separating common tangent problem and “partially correct” for the outer common tangent problem.

THEOREM 3.1. *If Algorithm 1 is to solve the outer common tangent problem (case 1 or 2), then it returns the solution (s_0, s_1) if the convex hulls of P_0 and P_1 are disjoint and reports “no solution” if the convex hulls of P_0 and P_1 are nested. If Algorithm 1 is to solve the separating common tangent problem (case 3 or 4), then it returns the solution (s_0, s_1) if the convex hulls of P_0 and P_1 are disjoint and reports “no solution” otherwise. Moreover, Algorithm 1 runs in linear time and uses constant workspace.*

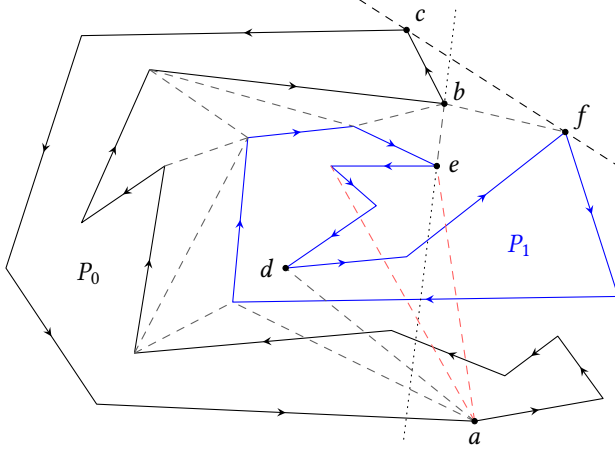


Fig. 6. An example of how Algorithm 2 finds and Algorithm 1 fails to find the outer common tangent $\mathcal{L}(c, f)$ of P_0 and P_1 starting from $(p_0[0], p_1[0]) = (a, d)$. The segments $p_0[s_0]p_1[s_1]$ on intermediate candidate lines are shown by dashed lines—gray for those considered by Algorithm 2 and red for those considered by Algorithm 1 but not Algorithm 2. Both algorithms proceed along the same lines until the 26th iteration. In particular, in the 18th iteration of both algorithms, an update makes $(p_0[s_0], p_1[s_1]) = (b, e)$ and the dotted line $\mathcal{L}(b, e)$ becomes the candidate line. In the 27th iteration, after the assignment $v_u \leftarrow v_u + 1$, the algorithms encounter $u = 0$ and $p_0[v_0] = a$. Algorithm 1 then makes $p_0[s_0] = a$, and in the 33rd iteration, it reaches back the state $(p_0[s_0], p_1[s_1]) = (p_0[v_0], p_1[v_1]) = (a, d)$ and $u = 0$ where it started except that the indices s_0, v_0, s_1 , and v_1 have increased by the sizes of the respective polygons. If the conditions $s_0 < 2n_0$ and $s_1 < 2n_1$ of the “while” loop were ignored, Algorithm 1 would keep updating (s_0, s_1) indefinitely. By contrast, in the 27th iteration of Algorithm 2, the test in line 5 succeeds and b_0 is set. Algorithm 2 continues by making $(p_0[s_0], p_1[s_1]) = (b, f)$ in the 28th iteration (while clearing b_0) and finally $(p_0[s_0], p_1[s_1]) = (c, f)$ in the 29th iteration.

If the convex hulls of P_0 and P_1 properly overlap, then Algorithm 1 can fail to find the solution even though it exists. An example of such behavior is presented in Figure 6. Algorithm 2 is an improved version of Algorithm 1 that solves the problem correctly in all cases including the case of properly overlapping convex hulls. In line 5 of Algorithm 2, $\Delta(a, b, c)$ denotes the triangular region spanned by a, b , and c , and a test of the form $z \in \Delta(a, b, c)$ is equivalent to testing whether $\det^*(z, a, b)$, $\det^*(z, b, c)$, and $\det^*(z, c, a)$ are all positive or all negative (they are all non-zero, by the general position assumption). Suppose that $u = 0$ (for simplicity of the explanation that follows) and the test in line 5 succeeds. Hence $p_1[s_1]$ belongs to the convex hull of P_0 . Algorithm 1 would now set s_0 to v_0 . Intuitively, this would “reverse” the orientation of the candidate line (which is most evident when the angle $p_0[s_0]p_1[s_1]p_0[v_0]$ is close to π), possibly leading to failure. Algorithm 2 proceeds differently: s_0 remains unchanged, and a boolean variable b_0 is set in order to prevent updates to s_0 in further iterations of the algorithm until one of the iterations makes an update to s_1 and clears b_0 in line 8. As we will show, such an update to s_1 must occur unless the convex hull of P_1 is contained in the convex hull of P_0 , and preventing updates to s_0 when b_0 is set suffices to guarantee correctness of the algorithm in all cases.

See Figure 6 for an example run of Algorithm 2 for the outer common tangent problem (case 1), where the convex hulls of P_0 and P_1 properly overlap. If the convex hulls of P_0 and P_1 are disjoint, then the test in line 5 of Algorithm 2 never succeeds, the variables b_0 and b_1 remain unset, and thus Algorithm 2 essentially becomes Algorithm 1.

THEOREM 3.2. *If Algorithm 2 is to solve the outer common tangent problem (case 1 or 2), then it returns the solution (s_0, s_1) unless the convex hulls of P_0 and P_1 are nested, in which case it reports “no solution”. If Algorithm 2 is to solve the separating common tangent problem (case 3 or 4), then it returns the solution (s_0, s_1) if the convex hulls of P_0 and P_1 are disjoint and reports “no solution” otherwise. Moreover, Algorithm 2 runs in linear time and uses constant workspace.*

4 CORRECTNESS OF ALGORITHM 1 AND ALGORITHM 2

In this section, we prove Theorem 3.1 and Theorem 3.2 on correctness and efficiency of Algorithms 1 and 2 while leaving the proof of a key lemma to the next section. First, we prove the claims on running time and workspace usage in Theorems 3.1 and 3.2.

LEMMA 4.1. *Algorithms 1 and 2 run in linear time and use constant workspace.*

PROOF. It is clear that the algorithms use constant workspace. For the bound on the running time, we prove the following two claims:

- (1) Before and after every iteration of the “while” loop in Algorithm 1 or 2, we have $(v_u - s_u) - (v_{1-u} - s_{1-u}) \in \{-1, 0\}$.
- (2) In each iteration, the sum $s_0 + s_1 + v_0 + v_1$ is increased by at least 1.

Initially, we have $s_u = v_u = s_{1-u} = v_{1-u} = 0$, so statement 1 holds before the first iteration. Now, suppose that statement 1 holds before iteration i . After the assignment $v_u \leftarrow v_u + 1$, we have $(v_u - s_u) - (v_{1-u} - s_{1-u}) \in \{0, 1\}$, and the sum $s_0 + s_1 + v_0 + v_1$ has been increased by 1. The former implies that if the assignments $s_u \leftarrow v_u$ and $v_{1-u} \leftarrow s_{1-u}$ are performed in iteration i (in line 5 of Algorithm 1 or line 8 of Algorithm 2), then the sum $s_0 + s_1 + v_0 + v_1$ remains unchanged or is increased by 1 again, and we have $(v_u - s_u) - (v_{1-u} - s_{1-u}) = 0$ afterwards. In total, the sum $s_0 + s_1 + v_0 + v_1$ is increased by 1 or 2 in iteration i . Finally, after the assignment $u \leftarrow 1 - u$, we have $(v_u - s_u) - (v_{1-u} - s_{1-u}) \in \{-1, 0\}$, so statement 1 holds at the end of iteration i .

Statement 1 implies that each iteration starts with $s_0 < 2n_0$, $s_1 < 2n_1$, $v_0 - s_0 \leq \max(n_0, n_1)$, and $v_1 - s_1 \leq \max(n_0, n_1)$ (where at least one of the last two inequalities is strict), otherwise the “while” loop would terminate before that iteration. Therefore, we have $s_0 + s_1 + v_0 + v_1 \leq 2(s_0 + s_1) + 2\max(n_0, n_1) < 6(n_0 + n_1)$ before each iteration fully performed by the algorithm, in particular the last one. This, the fact that $s_0 + s_1 + v_0 + v_1 = 0$ before the first iteration, and statement 2 imply that the algorithm makes at most $6(n_0 + n_1)$ iterations, each of which takes constant time. \square

Let $k \in \{0, 1\}$. We extend the notation $p_k[x]$ to all real numbers x to make the function $\mathbb{R} \ni x \mapsto p_k[x] \in P_k$ a continuous and piecewise linear traversal of P_k wrapping around with period n_k :

$$p_k[x] = (\lceil x \rceil - x)p_k[\lfloor x \rfloor] + (x - \lfloor x \rfloor)p_k[\lceil x \rceil] \in p_k[\lfloor x \rfloor]p_k[\lceil x \rceil], \quad \text{for } x \in \mathbb{R} \setminus \mathbb{Z}.$$

When $x, y \in \mathbb{R}$ and $x \leq y$, we let $P_k[x, y]$ denote the part of P_k from $p_k[x]$ to $p_k[y]$ in the forward direction of P_k , that is, $P_k[x, y] = \{p_k[z] : z \in [x, y]\}$. We say that $P_k[x, y]$ is a *cap* of $\mathcal{H}_k(a, b)$ (for distinct points $a, b \in \mathbb{R}^2$) if $P_k[x, y] \subset \mathcal{H}_k(a, b)$ and $P_k[x, y] \cap \mathcal{L}(a, b) = \{x, y\}$; this allows $x = y$.

LEMMA 4.2. *For each $k \in \{0, 1\}$, Algorithm 1 maintains the following invariant before and after every iteration of the “while” loop: $P_k[s_k, v_k] \subset \mathcal{H}_k(p_0[s_0], p_1[s_1])$.*

PROOF. Algorithm 1 starts with $s_0 = v_0 = s_1 = v_1 = 0$, so the invariant holds initially. To show that it is preserved by every iteration, suppose it holds before iteration i . Let u and v_u denote the values in iteration i after the assignment $v_u \leftarrow v_u + 1$ and before the assignment $u \leftarrow 1 - u$. Suppose the test in line 4 of Algorithm 1 succeeds, that is, $p_u[v_u] \notin \mathcal{H}_u(p_0[s_0], p_1[s_1])$; otherwise, clearly, the invariant is preserved by iteration i . The updates in line 5 yield $s_0 = v_0$ and $s_1 = v_1$, which makes the invariant satisfied after iteration i . \square

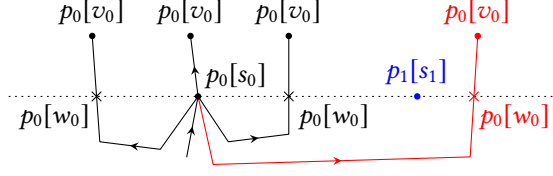


Fig. 7. Illustration for the statement of Lemma 4.3 in four possible cases of how $p_0[v_0 - 1]p_0[v_0]$ can intersect the candidate line $\mathcal{L}(p_0[s_0], p_1[s_1])$ when the test in line 4 of Algorithm 2 succeeds. The half-plane $\mathcal{H}_0(p_0[s_0], p_1[s_1])$ is below the dotted line. Algorithm 2 makes the update $s_0 \leftarrow v_0$ in the first three cases. In the last case (drawn red), b_0 becomes set and the second part of the invariant in Lemma 4.3 becomes satisfied.

LEMMA 4.3. (See Figure 7 for an illustration.) *For each $k \in \{0, 1\}$, Algorithm 2 maintains the following invariant before and after every iteration of the “while” loop:*

- if $b_k = \text{false}$, then $P_k[s_k, v_k] \subset \mathcal{H}_k(p_0[s_0], p_1[s_1])$;
- if $b_k = \text{true}$, then there is $w_k \in (s_k, v_k)$ such that $P_k[s_k, w_k]$ is a cap of $\mathcal{H}_k(p_0[s_0], p_1[s_1])$ and $p_{1-k}[s_{1-k}] \in p_k[s_k]p_k[w_k]$.

Moreover, on every update $s_u \leftarrow v_u$ in line 8 of Algorithm 2, if s_u denotes the value before the update, then $p_u[v_u] \notin \mathcal{H}_u(p_0[s_0], p_1[s_1])$ and there is $w_u \in [v_u - 1, v_u)$ such that $P_u[s_u, w_u]$ is a cap of $\mathcal{H}_u(p_0[s_0], p_1[s_1])$ and $p_{1-u}[s_{1-u}] \notin p_u[s_u]p_u[w_u]$.

PROOF. Algorithm 2 starts with $s_0 = v_0 = s_1 = v_1 = 0$ and $b_0 = b_1 = \text{false}$, so the invariant holds initially. To show that it is preserved by every iteration, suppose it holds before iteration i . Let u and v_u denote the values in iteration i after the assignment $v_u \leftarrow v_u + 1$ and before the assignment $u \leftarrow 1 - u$. Suppose the test in line 4 of Algorithm 2 succeeds, that is, $p_u[v_u] \notin \mathcal{H}_u(p_0[s_0], p_1[s_1])$ and $b_u = \text{false}$; otherwise, clearly, the invariant is preserved by iteration i . This and the assumption that the invariant holds before iteration i imply $P_u[s_u, v_u - 1] \subset \mathcal{H}_u(p_0[s_0], p_1[s_1])$ and $P_u[s_u, v_u] \not\subset \mathcal{H}_u(p_0[s_0], p_1[s_1])$. Let $w_u \in [v_u - 1, v_u)$ be maximal such that $P_u[s_u, w_u] \subset \mathcal{H}_u(p_0[s_0], p_1[s_1])$. That is, $p_u[w_u]$ is the intersection point of $p_u[v_u - 1]p_u[v_u]$ and $\mathcal{L}(p_0[s_0], p_1[s_1])$. The general position assumption implies that $P_u[s_u, w_u]$ is a cap of $\mathcal{H}_u(p_0[s_0], p_1[s_1])$. We have $p_{1-u}[s_{1-u}] \in p_u[s_u]p_u[w_u]$ if and only if $p_{1-u}[s_{1-u}] \in \Delta(p_u[s_u], p_u[v_u - 1], p_u[v_u])$. Therefore, if the test in line 5 succeeds, then the assignment $b_u \leftarrow \text{true}$ in line 6 makes the invariant satisfied after iteration i . Now, suppose the test in line 5 fails. It follows that $p_{1-u}[s_{1-u}] \notin p_u[s_u]p_u[w_u]$, so the update $s_u \leftarrow v_u$ in line 8 satisfies the second statement of the lemma. Furthermore, the updates in line 8 yield $s_0 = v_0$, $s_1 = v_1$, and $b_{1-u} = \text{false}$, which makes the invariant satisfied after iteration i . \square

Most effort in proving correctness of the two algorithms lies in the following two lemmas:

LEMMA 4.4. *If the convex hulls of P_0 and P_1 are disjoint, then the “while” loop in Algorithm 1 ends with $s_0 < 2n_0$ and $s_1 < 2n_1$.*

LEMMA 4.5. *If Algorithm 2 is to solve the outer common tangent problem (case 1 or 2) and the convex hulls of P_0 and P_1 are not nested or Algorithm 2 is to solve the separating common tangent problem (case 3 or 4) and the convex hulls of P_0 and P_1 are disjoint, then the “while” loop in Algorithm 2 ends with $s_0 < 2n_0$ and $s_1 < 2n_1$.*

If the convex hulls of P_0 and P_1 are disjoint, then the test in line 5 of Algorithm 2 never succeeds, b_1 and b_2 remain unset all the time, and thus Algorithm 2 becomes equivalent to Algorithm 1. Therefore, Lemma 4.4 is a direct consequence of Lemma 4.5. We prove Lemma 4.5 in the next section. Here, we proceed with the proofs of Theorems 3.1 and 3.2 assuming Lemma 4.5.

PROOF OF THEOREM 3.1. Algorithm 1 returns (s_0, s_1) only when the “while” loop has terminated with $v_0 \geq s_0 + n_0$ and $v_1 \geq s_1 + n_1$, which implies $P_0 \subset \mathcal{H}_0(p_0[s_0], p_1[s_1])$ and $P_1 \subset \mathcal{H}_1(p_0[s_0], p_1[s_1])$, in view of Lemma 4.2. Therefore, whenever Algorithm 1 returns (s_0, s_1) , it is the correct solution. This implies that Algorithm 1 correctly reports “no solution” in all cases where there is indeed no solution, that is, if the algorithm is to solve the outer common tangent problem (case 1 or 2) and the convex hulls of P_0 and P_1 are nested or it is to solve the separating common tangent problem (case 3 or 4) and the convex hulls of P_0 and P_1 are not disjoint. By Lemma 4.4, if the convex hulls of P_0 and P_1 are disjoint, then the “while” loop ends with $s_0 < 2n_0$ and $s_1 < 2n_1$, so the algorithm returns (s_0, s_1) , which is then the correct solution, as we have already argued. By Lemma 4.1, the running time and the workspace usage are as stated. \square

For the proof of correctness of Algorithm 2, we need one more lemma.

LEMMA 4.6. *If Algorithm 2 is to solve the outer common tangent problem (case 1 or 2) and the convex hulls of P_0 and P_1 are not nested or Algorithm 2 is to solve the separating common tangent problem (case 3 or 4) and the convex hulls of P_0 and P_1 are disjoint, then the “while” loop in Algorithm 2 ends with $b_0 = b_1 = \text{false}$.*

PROOF. Consider the final values of s_0, v_0, b_0, s_1, v_1 , and b_1 when the “while” loop in Algorithm 2 is terminated. Lemma 4.5 yields $s_0 < 2n_0$ and $s_1 < 2n_1$. This and the termination condition implies $v_0 \geq s_0 + n_0$ and $v_1 \geq s_1 + n_1$. If the algorithm is to solve the separating common tangent problem (case 3 or 4) and we have $b_0 = \text{true}$ or $b_1 = \text{true}$, then Lemma 4.3 implies that the convex hulls of P_0 and P_1 are not disjoint, contrary to the assumption of the present lemma. Now, suppose the algorithm is to solve the outer common tangent problem (case 1 or 2). Thus $\mathcal{H}_0(p_0[s_0], p_1[s_1]) = \mathcal{H}_1(p_0[s_0], p_1[s_1])$. If $b_k = \text{true}$ and $b_{1-k} = \text{false}$ for some $k \in \{0, 1\}$, then Lemma 4.3 and the fact that $P_{1-k}[s_{1-k}, v_{1-k}] = P_{1-k}$ yield a cap $P_k[s_k, w_k]$ of $\mathcal{H}_k(p_0[s_0], p_1[s_1])$ such that P_{1-k} is contained in the polygonal region bounded by $P_k[s_k, w_k] \cup p_k[s_k]p_k[w_k]$, and therefore the convex hull of P_{1-k} is contained in the convex hull of P_k , contrary to the assumption of the lemma. If $b_0 = b_1 = \text{true}$, then Lemma 4.3 yields a cap $P_0[s_0, w_0]$ of $\mathcal{H}_0(p_0[s_0], p_1[s_1])$ and a cap $P_1[s_1, w_1]$ of $\mathcal{H}_1(p_0[s_0], p_1[s_1])$ such that the points $p_1[w_1], p_0[s_0], p_1[s_1]$, and $p_0[w_0]$ occur in this order on $\mathcal{L}(p_0[s_0], p_1[s_1])$, which is impossible when $\mathcal{H}_0(p_0[s_0], p_1[s_1]) = \mathcal{H}_1(p_0[s_0], p_1[s_1])$. \square

PROOF OF THEOREM 3.2. Algorithm 2 returns (s_0, s_1) only when the “while” loop has terminated with $v_0 \geq s_0 + n_0$, $v_1 \geq s_1 + n_1$, and $b_0 = b_1 = \text{false}$, which implies $P_0 \subset \mathcal{H}_0(p_0[s_0], p_1[s_1])$ and $P_1 \subset \mathcal{H}_1(p_0[s_0], p_1[s_1])$, in view of Lemma 4.3. Therefore, whenever Algorithm 2 returns (s_0, s_1) , it is the correct solution. This implies that Algorithm 2 correctly reports “no solution” in all cases where there is indeed no solution, that is, if the algorithm is to solve the outer common tangent problem (case 1 or 2) and the convex hulls of P_0 and P_1 are nested or it is to solve the separating common tangent problem (case 3 or 4) and the convex hulls of P_0 and P_1 are not disjoint. In the other cases, the “while” loop ends with $s_0 < 2n_0$ and $s_1 < 2n_1$, by Lemma 4.5, and with $b_0 = b_1 = \text{false}$, by Lemma 4.6, and therefore the algorithm returns (s_0, s_1) , which is then the correct solution, as we have already argued. By Lemma 4.1, the running time and the workspace usage are as stated. \square

5 PROOF OF LEMMA 4.5

To complete the proof of correctness of the two algorithms, it remains to prove Lemma 4.5. For the rest of this section, we adopt the assumptions of Lemma 4.5, in particular the assumption that the solution exists, and we show that it is found and returned by Algorithm 2.

5.1 Reduction to one case of the outer common tangent problem

We will reduce Lemma 4.5 for all cases 1–4 of the common tangent problem just to case 1. First, we explain what we mean by such a reduction. An input to Algorithm 2 is a quadruple $(P_0, P_1, \alpha_0, \alpha_1)$, where $P_0 = \mathcal{P}(p_0[0], \dots, p_0[n_0-1])$, $P_1 = \mathcal{P}(p_1[0], \dots, p_1[n_1-1])$, and $\alpha_0, \alpha_1 \in \{+1, -1\}$ are implicit parameters that determine the particular case 1–4 of the common tangent problem to be solved by the algorithm (see Section 3). Consider two inputs $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$, where $P_k = \mathcal{P}(p_k[0], \dots, p_k[n_k-1])$ and $P'_k = \mathcal{P}(p'_k[0], \dots, p'_k[n_k-1])$ for each $k \in \{0, 1\}$. Whenever a (or b, c , etc.) denotes the point $p_k[i]$ (where $k \in \{0, 1\}$ and $i \in \mathbb{Z}$), we let a' (or b', c' , etc.) denote the point $p'_k[i]$. The inputs $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$ are *equivalent* if the following holds for all $a, b, c \in \{p_0[0], \dots, p_0[n_0-1], p_1[0], \dots, p_1[n_1-1]\}$:

$$\begin{aligned} \alpha_0 \operatorname{sgn} \det^*(a, b, c) &= \alpha'_0 \operatorname{sgn} \det^*(a', b', c') & \text{if } |\{a, b, c\} \cap \{p_0[0], \dots, p_0[n_0-1]\}| \in \{0, 2\}, \\ \alpha_1 \operatorname{sgn} \det^*(a, b, c) &= \alpha'_1 \operatorname{sgn} \det^*(a', b', c') & \text{if } |\{a, b, c\} \cap \{p_0[0], \dots, p_0[n_0-1]\}| \in \{1, 3\}. \end{aligned}$$

With this definition, equivalent inputs have the same solutions and lead to the same outcomes of Algorithm 2, as we show in the next two lemmas. Therefore, equivalence of inputs provides a formal way of reducing one case of Lemma 4.5 to another one.

LEMMA 5.1. *If inputs $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$ are equivalent and $s_0, s_1 \in \mathbb{Z}$, then (s_0, s_1) either is the correct solution or is not the correct solution to both inputs.*

PROOF. The conditions in the definition of equivalence imply that $\alpha_k \det^*(p_0[s_0], p_1[s_1], p_k[i]) > 0$ if and only if $\alpha'_k \det^*(p'_0[s_0], p'_1[s_1], p'_k[i]) > 0$, for any $k \in \{0, 1\}$ and $i \in \mathbb{Z}$. This implies that $P_k \subset \mathcal{H}_k(p_0[s_0], p_1[s_1])$ if and only if $P'_k \subset \mathcal{H}_k(p'_0[s_0], p'_1[s_1])$, for any $k \in \{0, 1\}$, that is, $\mathcal{L}(p_0[s_0], p_1[s_1])$ is the requested common tangent of P_0 and P_1 if and only if $\mathcal{L}(p'_0[s_0], p'_1[s_1])$ is the requested common tangent of P'_0 and P'_1 . \square

LEMMA 5.2. *If inputs $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$ are equivalent, then Algorithm 2 applied to $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$ ends with the same final value of the pair of variables (s_0, s_1) .*

PROOF. Let $k \in \{0, 1\}$. Recall from Section 3 the implicit constant $\beta_k \in \{+1, -1\}$, which determines whether Algorithm 2 traverses P_k (P'_k) forwards or backwards. We show that β_k has the same value for both inputs. It is well known that P_k can be *triangulated*; in particular, there are $n_k - 2$ triangles of the form $\mathcal{P}(a_t, b_t, c_t)$ with $a_t, b_t, c_t \in \{p_k[0], \dots, p_k[n_k-1]\}$ ($1 \leq t \leq n_k - 2$), all with the same orientation (counterclockwise or clockwise), such that

$$\det^*(p_k[0], \dots, p_k[n_k-1]) = \sum_{t=1}^{n_k-2} (\det(a_t, b_t) + \det(b_t, c_t) + \det(c_t, a_t)) = \sum_{t=1}^{n_k-2} \det^*(a_t, b_t, c_t).$$

Equivalence of $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$ implies

$$\alpha_{1-k} \operatorname{sgn} \det^*(a_t, b_t, c_t) = \alpha'_{1-k} \operatorname{sgn} \det^*(a'_t, b'_t, c'_t) \quad \text{for all } t \in \{1, \dots, n_k - 2\}.$$

We conclude that all triangles $\mathcal{P}(a'_t, b'_t, c'_t)$ ($1 \leq t \leq n_k - 2$) have the same orientation and

$$\alpha_{1-k} \operatorname{sgn} \det^*(p_k[0], \dots, p_k[n_k-1]) = \alpha'_{1-k} \operatorname{sgn} \det^*(p'_k[0], \dots, p'_k[n_k-1]).$$

This and the definition of β_k implies that β_k has the same value for both inputs.

We show that Algorithm 2 proceeds in exactly the same way for both inputs. Specifically, we show the same number of iterations of the “while” loop is performed for both inputs, and for each iteration i , the variables s_0, v_0, b_0, s_1, v_1 , and b_1 have the same values for both inputs before and after iteration i (it is clear that u has the same value, because it depends only on i). This implies, in particular, that the algorithm ends with the same final value of the pair of variables (s_0, s_1) .

The initial setup is common for both inputs. Now, suppose s_0, v_0, b_0, s_1, v_1 , and b_1 have the same values for both inputs before iteration i . The test in line 4 produces the same outcome for both inputs, because equivalence of $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, \alpha'_1)$ implies that $\alpha_u \det^*(p_0[s_0], p_1[s_1], p_u[v_u]) > 0$ if and only if $\alpha'_u \det^*(p'_0[s_0], p'_1[s_1], p'_u[v_u]) > 0$. If that outcome is positive, then the test in line 5 produces the same outcome for both inputs, by an analogous argument. It follows that the same assignments are performed in iteration i for both inputs, and therefore s_0, v_0, b_0, s_1, v_1 , and b_1 have the same values for both inputs after iteration i . \square

First, we reduce Lemma 4.5 for cases 2 and 4 of the common tangent problem to cases 1 and 3 thereof. Consider the transformation $\phi: \mathbb{R}^2 \ni (x, y) \mapsto (-x, y) \in \mathbb{R}^2$ (horizontal flip). Let $P'_0 = \mathcal{P}(\phi(p_0[0]), \dots, \phi(p_0[n_0 - 1]))$ and $P'_1 = \mathcal{P}(\phi(p_1[0]), \dots, \phi(p_1[n_1 - 1]))$. Clearly, for any three points $a, b, c \in \mathbb{R}^2$, we have $\det^*(\phi(a), \phi(b), \phi(c)) = -\det^*(a, b, c)$. It follows that the input $(P_0, P_1, \alpha_0, \alpha_1)$ is equivalent to $(P'_0, P'_1, -\alpha_0, -\alpha_1)$. If the former is case 2 or 4 of the common tangent problem, then the latter is case 1 or 3 thereof, respectively. By Lemmas 5.1 and 5.2, it remains to prove Lemma 4.5 for cases 1 and 3 of the common tangent problem.

Now, we reduce Lemma 4.5 for case 3 of the common tangent problem to case 1 thereof. Suppose an input $(P_0, P_1, \alpha_0, \alpha_1)$ is case 3 of the common tangent problem, that is, $\alpha_0 = +1$ and $\alpha_1 = -1$. Assume that the convex hulls of P_0 and P_1 are disjoint (as in Lemma 4.5). It follows that there is a straight line separating the two convex hulls in the plane. Assume without loss of generality that it is the vertical line $x = 0$ and every corner of P_0 has negative x -coordinate while every corner of P_1 has positive x -coordinate, applying an appropriate rotation or translation of the plane to turn $(P_0, P_1, \alpha_0, \alpha_1)$ into an equivalent input that has these properties. Consider the transformation

$$\phi: (\mathbb{R} \setminus \{0\}) \times \mathbb{R} \ni (x, y) \mapsto \left(\frac{y}{x}, \frac{1}{x}\right) \in (\mathbb{R} \setminus \{0\}) \times \mathbb{R}.$$

For any three points $a = (a_x, a_y)$, $b = (b_x, b_y)$, and $c = (c_x, c_y)$ in $(\mathbb{R} \setminus \{0\}) \times \mathbb{R}$, we have

$$\det^*(\phi(a), \phi(b), \phi(c)) = \begin{vmatrix} \frac{a_y}{a_x} & \frac{b_y}{b_x} & \frac{c_y}{c_x} \\ \frac{1}{a_x} & \frac{1}{b_x} & \frac{1}{c_x} \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ \frac{a_y}{a_x} & \frac{b_y}{b_x} & \frac{c_y}{c_x} \\ \frac{1}{a_x} & \frac{1}{b_x} & \frac{1}{c_x} \end{vmatrix} = \frac{\det^*(a, b, c)}{a_x b_x c_x}. \quad (\dagger)$$

Since a, b , and c are collinear if and only if $\det^*(a, b, c) = 0$, it follows from (\dagger) that ϕ preserves collinearity (actually, it is a projective transformation). This and the fact that ϕ is a bijection on $(\mathbb{R} \setminus \{0\}) \times \mathbb{R}$ imply that ϕ transforms the polygons P_0 and P_1 into (simple) polygons $P'_0 = \mathcal{P}(\phi(p_0[0]), \dots, \phi(p_0[n_0 - 1]))$ and $P'_1 = \mathcal{P}(\phi(p_1[0]), \dots, \phi(p_1[n_1 - 1]))$, respectively. Since the corners of P_0 have negative x -coordinates and the corners of P_1 have positive x -coordinates, the equality (\dagger) implies the following, for all $a, b, c \in \{p_0[0], \dots, p_0[n_0 - 1], p_1[0], \dots, p_1[n_1 - 1]\}$:

$$\begin{aligned} \text{sgn } \det^*(\phi(a), \phi(b), \phi(c)) &= \text{sgn } \det^*(a, b, c) & \text{if } |\{a, b, c\} \cap \{p_0[0], \dots, p_0[n_0 - 1]\}| \in \{0, 2\}, \\ \text{sgn } \det^*(\phi(a), \phi(b), \phi(c)) &= -\text{sgn } \det^*(a, b, c) & \text{if } |\{a, b, c\} \cap \{p_0[0], \dots, p_0[n_0 - 1]\}| \in \{1, 3\}. \end{aligned}$$

Therefore, the inputs $(P_0, P_1, \alpha_0, \alpha_1)$ and $(P'_0, P'_1, \alpha'_0, -\alpha'_1)$ are equivalent. Since the former is case 3 of the common tangent problem, the latter is case 1 thereof. By Lemmas 5.1 and 5.2, it remains to prove Lemma 4.5 for case 1 of the problem. This is what the remainder of Section 5 is devoted to.

5.2 Auxiliary concepts

For the sequel, we assume that the convex hulls of P_0 and P_1 are not nested, P_0 is oriented counter-clockwise, P_1 is oriented clockwise, and Algorithm 2 is to solve case 1 of the outer common tangent problem—compute a pair of indices (s_0, s_1) such that $P_0, P_1 \subset \text{RHP}(p_0[s_0], p_1[s_1])$.

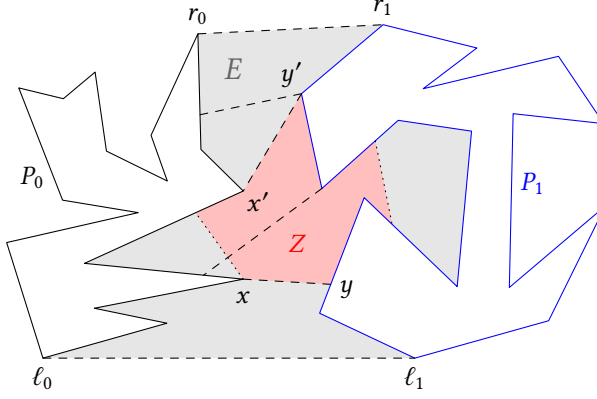


Fig. 8. Illustration for the concepts of a door and a zone and for Observations 5.3–5.5. Some pairwise non-crossing doors including $\ell_0\ell_1$ and r_0r_1 are indicated by dashed lines. The polygonal regions $P_0[\ell_0, r_0]$ and $P_1[\ell_1, r_1]$ and the doors $\ell_0\ell_1$ and r_0r_1 determine the shaded zone E , which contains all doors and zones. The boundary of the red zone Z contains two doors $xy < x'y'$; Z lies to the left of xy and to the right of $x'y'$.

Recall that a segment ab in the plane is considered oriented from a to b , so that forward traversal of ab starts at a and ends at b . A *polygonal path* is a curve in the plane composed of n segments $a_0a_1, a_1a_2, \dots, a_{n-1}a_n$ with no common points other than the common endpoints of pairs of consecutive segments in that order. Such a polygonal path is considered oriented from a_0 to a_n , so that forward traversal of it starts at a_0 and ends at a_n . A segment or a polygonal path is *degenerate* if it consists of a single point. When a non-degenerate polygonal path $a_0a_1, a_1a_2, \dots, a_{n-1}a_n$ (a non-degenerate segment if $n = 1$) is contained in the boundary of a polygonal region Q , we say that Q lies *to the left* or *to the right* of $a_0a_1, a_1a_2, \dots, a_{n-1}a_n$ if forward traversal of $a_0a_1, a_1a_2, \dots, a_{n-1}a_n$ agrees with counterclockwise traversal or clockwise traversal, respectively, of the boundary of Q . For $k \in \{0, 1\}$ and $a, b \in P_k$, let $P_k[a, b]$ be the polygonal path from a to b along P_k in the forward direction (counterclockwise for P_0 and clockwise for P_1); in particular, $P_k[a, a] = \{a\}$.

Let $\ell_0, r_0 \in P_0$ and $\ell_1, r_1 \in P_1$ be such that $P_0, P_1 \subset \text{LHP}(\ell_0, \ell_1) \cap \text{RHP}(r_0, r_1)$. Thus r_0 and r_1 determine the requested outer common tangent, while ℓ_0 and ℓ_1 determine the other one. It is possible that $\ell_0 = r_0$ or $\ell_1 = r_1$ (but not both). For clarity of presentation, we will ignore this special case and proceed as if $\ell_0 \neq r_0$ and $\ell_1 \neq r_1$. Our arguments remain correct when $\ell_k = r_k$ ($k \in \{0, 1\}$) after adding the following exceptions to the definitions of a polygon, a polygonal path, and $P_k[a, b]$:

- the point $\ell_k = r_k$ is allowed to occur twice on a polygon (as two corners) or a polygonal path (as both endpoints of the path), where one occurrence is denoted by ℓ_k and the other by r_k ;
- $P_k[\ell_k, r_k] = P_k$: forward traversal of $P_k[\ell_k, r_k]$ makes one full traversal of P_k from ℓ_k to r_k in the forward direction of P_k (counterclockwise for P_0 and clockwise for P_1).

A *door* is a segment xy such that $xy \cap P_0 = \{x\}$ and $xy \cap P_1 = \{y\}$. We always orient the door from the endpoint on P_0 to the endpoint on P_1 . A *zone* is a polygonal region Z such that the interior of Z is disjoint from $P_0 \cup P_1$ and the boundary of Z is the union of some non-empty part of P_0 (not necessarily connected), some non-empty part of P_1 (likewise), and some segments with both endpoints on $P_0 \cup P_1$ (not necessarily doors). These concepts are illustrated in Figure 8. Let E be the polygonal region bounded by $\ell_0\ell_1 \cup P_0[\ell_0, r_0] \cup P_1[\ell_1, r_1] \cup r_0r_1$. Since $E \subset \text{LHP}(\ell_0, \ell_1) \cap \text{RHP}(r_0, r_1)$, it follows that E lies to the left of $\ell_0\ell_1$, to the right of $P_0[\ell_0, r_0]$, to the left of $P_1[\ell_1, r_1]$, and to the right of r_0r_1 . Figure 8 also illustrates the next three observations.

OBSERVATION 5.3. *The polygonal region E is a zone and satisfies $E \cap P_0 = P_0[\ell_0, r_0]$ and $E \cap P_1 = P_1[\ell_1, r_1]$. Moreover, every door or zone is contained in E . In particular, every door has one endpoint on $P_0[\ell_0, r_0]$ and the other on $P_1[\ell_1, r_1]$.*

PROOF. Let $k \in \{0, 1\}$ and $S_k = (\text{LHP}(\ell_0, \ell_1) \cap \text{RHP}(r_0, r_1)) \setminus P_k[\ell_k, r_k]$. Since E and the polygonal region bounded by P_k lie on opposite sides of $P_k[\ell_k, r_k]$, the sets $P_k \setminus P_k[\ell_k, r_k]$ and $E \setminus P_k[\ell_k, r_k]$ are contained in different connected components of S_k . A consequence of this property is that $E \cap P_k = P_k[\ell_k, r_k]$. This, for both $k \in \{0, 1\}$, proves the first statement. Another consequence is that $P_k \setminus P_k[\ell_k, r_k]$ and P_{1-k} belong to different connected components of S_k , because $E \setminus P_k[\ell_k, r_k]$ and P_{1-k} intersect. Therefore, $P_k[\ell_k, r_k]$ intersects the interior of every segment or polygonal region that is contained in $\text{LHP}(\ell_0, \ell_1) \cap \text{RHP}(r_0, r_1)$ and intersects both $P_k \setminus P_k[\ell_k, r_k]$ and P_{1-k} . However, every door or zone is contained in $\text{LHP}(\ell_0, \ell_1) \cap \text{RHP}(r_0, r_1)$ (because P_0 and P_1 are) and is internally disjoint from $P_k[\ell_k, r_k]$. This, for both $k \in \{0, 1\}$, proves the last two statements. \square

For two doors xy and $x'y'$, let $xy \leq x'y'$ denote that $P_0[x, x'] \subseteq P_0[\ell_0, r_0]$ and $P_1[y, y'] \subseteq P_1[\ell_1, r_1]$ (that is, forward traversal of $P_0[\ell_0, r_0]$ encounters x no later than x' and forward traversal of $P_1[\ell_1, r_1]$ encounters y no later than y'), and let $xy < x'y'$ denote that $xy \leq x'y'$ and $xy \neq x'y'$. Two doors are *non-crossing* if they are disjoint or they intersect only at a common endpoint. The following observation implies that $<$ is a total order on any set of pairwise non-crossing doors:

OBSERVATION 5.4. *Any two non-crossing doors xy and $x'y'$ satisfy $xy < x'y'$ or $x'y' < xy$.*

PROOF. Observation 5.3 yields $xy, x'y' \subset E$. If neither $xy < x'y'$ nor $x'y' < xy$, then the points x, x', y , and y' are distinct and occur on the boundary of E in this cyclic order (clockwise or counterclockwise), which contradicts the assumption that xy and $x'y'$ do not cross. \square

OBSERVATION 5.5. *The boundary of every zone Z contains exactly two doors. Moreover, if these doors are denoted by xy and $x'y'$ so that $xy < x'y'$, then*

- (1) Z lies to the left of xy and to the right of $x'y'$,
- (2) a door $x''y''$ is disjoint from the interior of Z if and only if $x''y'' \leq xy$ or $x'y' \leq x''y''$.

PROOF. The boundary of Z intersects both P_0 and P_1 , so it contains at least two doors. By Observation 5.4, since the doors on the boundary of Z are pairwise non-crossing, they are totally ordered by $<$. Let xy be the minimum door and $x'y'$ be the maximum door on the boundary of Z with respect to the order $<$. We will show statements 1 and 2 for xy and $x'y'$. Statement 2, minimality of xy , and maximality of $x'y'$ imply that the boundary of Z contains no other doors.

For every door $x''y''$, since $Z \subseteq E$ and $x''y'' \subset E$ (by Observation 5.3), the following holds: if the set $E \setminus x''y''$ has two connected components intersecting Z , then $x''y''$ intersects the interior of Z . If neither $x''y'' \leq xy$ nor $x'y' \leq x''y''$, then the set $E \setminus x''y''$ has two connected components intersecting Z (by the definition of \leq), so $x''y''$ intersects the interior of Z , which is one implication in statement 2. It also follows that either of the sets $E \setminus xy$ and $E \setminus x'y'$ has only one connected component intersecting Z , so Z is contained in the polygonal region Z^* bounded by $xy \cup P_0[x, x'] \cup P_1[y, y'] \cup x'y'$, which is contained in E . If $x \neq x'$, then Z^* lies to the right of $P_0[x, x']$ (because E does), and if $y \neq y'$, then Z^* lies to the left of $P_1[y, y']$ (because E does). Therefore, Z^* and thus Z lie to the left of xy and to the right of $x'y'$, which is statement 1. Moreover, if $x''y'' \leq xy$ or $x'y' \leq x''y''$, then $x''y''$ is disjoint from the interior of Z^* and therefore is disjoint from the interior of Z , which is the converse implication in statement 2. \square

For the rest of this subsection, fix points $q_0 \in P_0$ and $q_1 \in P_1$, and consider doors contained in q_0q_1 (doors on q_0q_1 in short). Recall that a door xy is always oriented from its endpoint x on P_0 to its endpoint y on P_1 . The *sign* of such a door xy on q_0q_1 is

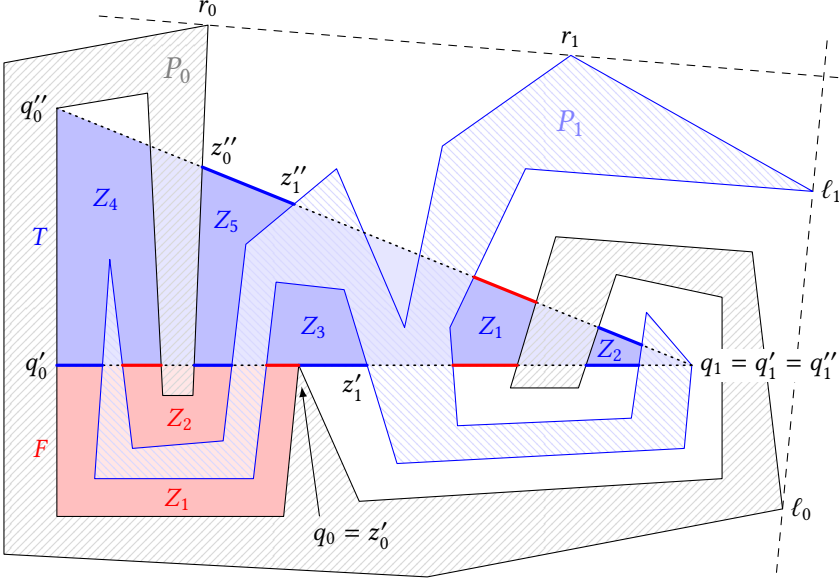


Fig. 9. Thick blue segments are positive doors and thick red segments are negative doors on $q'_0q'_1$ and $q''_0q''_1$. The primary doors are $z'_0z'_1$ on $q'_0q'_1$ and $z'_0z'_1$ on $q''_0q''_1$. The red region F (considered in Lemma 5.8) determines red zones $Z_1 < Z_2$. The blue region T (considered in Lemma 5.9) determines blue zones $Z_1 < Z_2 < Z_3 < Z_4 < Z_5$, of which Z_1, Z_2 , and Z_5 are two-sided while Z_3 and Z_4 are one-sided.

- +1 if forward traversal of q_0q_1 encounters x first and y second (then xy is *positive* on q_0q_1),
- -1 if forward traversal of q_0q_1 encounters y first and x second (then xy is *negative* on q_0q_1).

See Figure 9 for an illustration.

OBSERVATION 5.6. *The signs of all doors on q_0q_1 sum up to 1.*

PROOF. Let x_1y_1, \dots, x_dy_d be all the doors on q_0q_1 enumerated in the order they are encountered by forward traversal of q_0q_1 . The *first endpoint* of such a door is the one closer to q_0 , and the *last endpoint* is the one closer to q_1 . Every subsegment of q_0q_1 connecting a point on P_0 with a point on P_1 contains at least one of the doors. Since $q_0 \in P_0$, the subsegment of q_0q_1 from q_0 to the first endpoint of x_1y_1 contains no points of P_1 , so x_1y_1 is positive on q_0q_1 . For $i \in \{1, \dots, d-1\}$, the subsegment of q_0q_1 from the last endpoint of x_iy_i to the first endpoint of $x_{i+1}y_{i+1}$ contains no points of P_0 or no points of P_1 , so the sign of $x_{i+1}y_{i+1}$ is opposite to the sign of x_iy_i on q_0q_1 . Finally, since $q_1 \in P_1$, the subsegment of q_0q_1 from the last endpoint of x_dy_d to q_1 contains no points of P_0 , so x_dy_d is positive on q_0q_1 . This implies that x_iy_i is positive on q_0q_1 for i odd, x_iy_i is negative on q_0q_1 for i even, and d is odd. Therefore, the signs of x_1y_1, \dots, x_dy_d on q_0q_1 sum up to 1. \square

The doors on q_0q_1 are pairwise non-crossing, so they are totally ordered by the relation $<$, by Observation 5.4. Let x_1y_1, \dots, x_dy_d be all the doors on q_0q_1 ordered so that $x_1y_1 < \dots < x_dy_d$. The *primary door* on q_0q_1 is the door x_jy_j with minimum $j \in \{1, \dots, d\}$ such that the signs of x_1y_1, \dots, x_jy_j on q_0q_1 sum up to 1. Such an index j exists, because d is a candidate, by Observation 5.6. Minimality of j in the definition of the primary door directly implies the following:

OBSERVATION 5.7. *The primary door x_iy_i is positive on q_0q_1 . Moreover, if $i \geq 2$, then the door $x_{i-1}y_{i-1}$ is also positive on q_0q_1 .*

Tracking the primary door on the segment $q_0q_1 = p_0[s_0]p_1[s_1]$ as it changes in the course of the algorithm is the key idea in the proof of the remaining case of Lemma 4.5 that follows.

5.3 Proof of Lemma 4.5 for the remaining case

We go back to the proof of Lemma 4.5. Having reduced Lemma 4.5 to case 1 of the outer common tangent problem, we have assumed the setup of that case: the convex hulls of P_0 and P_1 are not nested, P_0 is oriented counterclockwise, P_1 is oriented clockwise, and Algorithm 2 is to compute a pair of indices (s_0, s_1) such that $P_0, P_1 \subset \text{RHP}(p_0[s_0], p_1[s_1])$, that is, $p_0[s_0] = r_0$ and $p_1[s_1] = r_1$.

Algorithm 2 starts with $(s_0, s_1) = (0, 0)$ and then makes some updates to the candidate solution (s_0, s_1) in line 8 until the end of the “while” loop. The second part of Lemma 4.3 explains what these updates look like: on every update $s_u \leftarrow v_u$ in line 8 of Algorithm 2, if s_u denotes the value before the update, then $p_u[v_u] \notin \text{RHP}(p_0[s_0], p_1[s_1])$ and there is $w_u \in [v_u - 1, v_u)$ such that $P_u[s_u, w_u]$ is a cap of $\text{RHP}(p_0[s_0], p_1[s_1])$, $P_u[w_u, v_u]$ is the segment $p_u[w_u]p_u[v_u]$, and $p_{1-u}[s_{1-u}] \notin p_u[s_u]p_u[w_u]$.

First, we present informally the general proof idea. Imagine that an update like above happens in continuous time, as follows. Let $q_0 = p_0[s_0]$ and $q_1 = p_1[s_1]$. If $s_u = w_u$, then the point q_u moves continuously along the segment $p_u[w_u]p_u[v_u]$ from $p_u[w_u]$ to $p_u[v_u]$. If $s_u < w_u$, then the point q_u jumps over all points $p_u[x]$ with $x \in (s_u, w_u)$ and then moves continuously along the segment $p_u[w_u]p_u[v_u] \setminus \{p_u[w_u]\}$ as in the case $s_u = w_u$. Thus $q_u = p_u[s_u]$ again after the assignment $s_u \leftarrow v_u$. As q_u is moving during the update, we track the primary door z_0z_1 on q_0q_1 and show that

- (1) the door z_0z_1 is only moving (piecewise continuously) forward in the order $<$,
- (2) the point q_u never passes or jumps over z_u .

To see how this implies Lemma 4.5, consider the overall move of q_0, q_1 , and z_0z_1 during all updates to the candidate solution (s_0, s_1) , starting from $q_0 = p_0[0]$ and $q_1 = p_1[0]$. For each $k \in \{0, 1\}$, statement 1 implies that z_k is only moving forward on $P_k[\ell_k, r_k]$, never passing or jumping over r_k , and statement 2 asserts that q_k never passes or jumps over z_k , whence it follows that q_k passes or jumps over r_k at most once.

Now, we proceed with the proof of Lemma 4.5. The next two lemmas formalize statement 1 above—Lemma 5.8 for the initial jump over $P_u[s_u, w_u]$, and Lemma 5.9 for the continuous move along $p[w_u]p[v_u]$. See Figure 9 for an illustration of Lemmas 5.8 and 5.9. Statement 2 above is formalized by the invariant in Lemma 5.10.

LEMMA 5.8. *Let $u \in \{0, 1\}$, $q_u, q'_u \in P_u$, and $q_{1-u} = q'_{1-u} \in P_{1-u}$. If $P_u[q_u, q'_u]$ is a cap of $\text{RHP}(q_0, q_1)$ and $q_{1-u} \notin q_uq'_u$, then the same door is primary on q_0q_1 and on $q'_0q'_1$.*

PROOF. The lemma is trivial when $q_u = q'_u$, so assume $q_u \neq q'_u$. Thus $q_0q_1 \subset q'_0q'_1$ or $q'_0q'_1 \subset q_0q_1$ (because $q_{1-u} \notin q_uq'_u$). Let $q''_0q''_1$ be the longer of q_0q_1 and $q'_0q'_1$ ($q'_0q'_1$ in the former and q_0q_1 in the latter case). Let F be the polygonal region bounded by $P_u[q_u, q'_u] \cup q_uq'_u$. Thus $F \subset \text{RHP}(q''_0, q''_1)$. Let \mathcal{Z} be the set of zones contained in F with boundaries contained in $(P_0 \cap F) \cup (P_1 \cap F) \cup q_uq'_u$. For every door $xy \subset q_uq'_u$, there is a zone in \mathcal{Z} to the right of xy (if F is to the right of xy) or to the left of xy (if F is to the left of xy). By Observation 5.5, the zones in \mathcal{Z} can be ordered as Z_1, \dots, Z_d and the doors contained in $q_uq'_u$ can be ordered as $x_1y_1, x^1y^1, \dots, x_dy_d, x^dy^d$ so that

- every zone $Z_i \in \mathcal{Z}$ has exactly two doors on the boundary, namely, x_iy_i and x^iy^i ,
- $x_1y_1 < x^1y^1 < \dots < x_dy_d < x^dy^d$.

For each $i \in \{1, \dots, d\}$, since $Z_i \subset \text{RHP}(q''_0, q''_1)$, Observation 5.5 (1 and 2) implies that

- x_iy_i is negative and x^iy^i is positive on $q''_0q''_1$,
- x_iy_i and x^iy^i are consecutive in the order $<$ of the doors on $q''_0q''_1$.

By Observation 5.7, none of $x_1y_1, x^1y^1, \dots, x_dy_d, x^dy^d$ is primary on $q''_0q''_1$. Moreover, for each door xy on the shorter of q_0q_1 and $q'_0q'_1$, the following two sums are equal:

- the sum of the signs of all doors on $q_0''q_1''$ up to xy in the order $<$,
- the sum of the signs of all doors on the shorter of q_0q_1 and $q_0'q_1'$ up to xy in the order $<$.

We conclude that the same door is primary on $q_0''q_1''$ and on the shorter of q_0q_1 and $q_0'q_1'$. \square

LEMMA 5.9. *Let $u \in \{0, 1\}$, $q_u'q_u'' \subset P_u$, and $q_{1-u}' = q_{1-u}'' \in P_{1-u}$. Let $z_0'z_1'$ be the primary door on $q_0'q_1'$ and $z_0''z_1''$ be the primary door on $q_0''q_1''$. If $q_u'' \notin \text{RHP}(q_0', q_1')$, then $z_0'z_1' < z_0''z_1''$.*

PROOF. Let T be the triangular region bounded by $q_0'q_1' \cup q_0'q_0'' \cup q_1'q_1'' \cup q_0''q_1''$, where either $q_0'q_0''$ or $q_1'q_1''$ is a degenerate segment. Thus $T \subset \text{LHP}(q_0', q_1') \cap \text{RHP}(q_0'', q_1'')$. Let \mathcal{Z} be the set of zones contained in T with boundaries contained in $q_0'q_1' \cup (P_0 \cap T) \cup (P_1 \cap T) \cup q_0''q_1''$. For every door $xy \subset q_0'q_1' \cup q_0''q_1''$, there is a zone in \mathcal{Z} to the right of xy (if T lies to the right of xy) or to the left of xy (if T lies to the left of xy). By Observation 5.5, the zones in \mathcal{Z} can be ordered as Z_1, \dots, Z_d and the doors contained in $q_0'q_1' \cup q_0''q_1''$ can be ordered as $x_1y_1, x^1y^1, \dots, x_dy_d, x^dy^d$ so that

- every zone $Z_i \in \mathcal{Z}$ has exactly two doors on the boundary, namely, x_iy_i and x^iy^i ,
- $x_1y_1 < x^1y^1 < \dots < x_dy_d < x^dy^d$.

For every $i \in \{1, \dots, d\}$, since $Z_i \subset \text{LHP}(q_0', q_1') \cap \text{RHP}(q_0'', q_1'')$, Observation 5.5 (1) implies that

- x_iy_i is a positive door on $q_0'q_1'$ or a negative door on $q_0''q_1''$,
- x^iy^i is a negative door on $q_0'q_1'$ or a positive door on $q_0''q_1''$.

We will use these two properties extensively without explicit reference.

We say that a zone $Z_i \in \mathcal{Z}$ is *one-sided* if x_iy_i and x^iy^i lie both on $q_0'q_1'$ or both on $q_0''q_1''$, otherwise we say that Z_i is *two-sided*. For each one-sided zone $Z_i \in \mathcal{Z}$, the doors x_iy_i and x^iy^i have opposite signs on $q_0'q_1'$ or $q_0''q_1''$ (whichever they lie on). For each two-sided zone $Z_i \in \mathcal{Z}$, if $x_i'y_i$ and $x_i''y_i''$ denote the two doors on the boundary of Z_i so that $x_i'y_i \subseteq q_0'q_1'$ and $x_i''y_i'' \subseteq q_0''q_1''$, then the sign of $x_i'y_i$ on $q_0'q_1'$ is equal to the sign of $x_i''y_i''$ on $q_0''q_1''$. Let I be the set of indices $i \in \{1, \dots, d\}$ such that Z_i is a two-sided zone in \mathcal{Z} . The above and Observation 5.6 implies that

- the signs of the doors $x_i'y_i$ on $q_0'q_1'$ over all $i \in I$ sum up to 1,
- the signs of the doors $x_i''y_i''$ on $q_0''q_1''$ over all $i \in I$ sum up to 1,

and the following four sums are equal, for each $j \in I$:

- the sum of the signs of all doors on $q_0'q_1'$ up to $x_j'y_j$ in the order $<$,
- the sum of the signs of the doors $x_i'y_i$ on $q_0'q_1'$ over all $i \in I \cap \{1, \dots, j\}$,
- the sum of the signs of the doors $x_i''y_i''$ on $q_0''q_1''$ over all $i \in I \cap \{1, \dots, j\}$,
- the sum of the signs of all doors on $q_0''q_1''$ up to $x_j''y_j''$ in the order $<$.

Let $j \in I$ be minimum such that the four sums above are equal to 1. Since x_iy_i is positive and x^iy^i is negative on $q_0'q_1'$ for every (one-sided) zone $Z_i \in \mathcal{Z}$ such that $x_iy_i, x^iy^i \subseteq q_0'q_1'$, it follows that the primary door on $q_0'q_1'$ is either $x_j'y_j$ or x_iy_i for some (one-sided) zone $Z_i \in \mathcal{Z}$ with $x_iy_i, x^iy^i \subseteq q_0'q_1'$ and $i < j$, and thus $x_iy_i < x_j'y_j$. For every (one-sided) zone $Z_i \in \mathcal{Z}$ such that $x_iy_i, x^iy^i \subseteq q_0''q_1''$, since x_iy_i is negative and x^iy^i is positive on $q_0''q_1''$, neither x_iy_i nor x^iy^i is primary on $q_0''q_1''$, by Observation 5.7. It follows that $x_j''y_j''$ is the primary door on $q_0''q_1''$. Since the primary door is always positive, we have $x_j'y_j = x_jy_j$ and $x_j''y_j'' = x^jy^j$, and thus $x_j'y_j < x_j''y_j''$. We conclude that $z_0'z_1' \leq x_j'y_j < x_j''y_j'' = z_0''z_1''$. \square

LEMMA 5.10. *Let $a_0 \in [0, n_0]$ and $a_1 \in [0, n_1]$ be such that $p_0[a_0] = \ell_0$ and $p_1[a_1] = \ell_1$. Algorithm 2 maintains the following invariant: if $c_0 \in [a_0, a_0 + n_0]$ and $c_1 \in [a_1, a_1 + n_1]$ are such that $p_0[c_0]p_1[c_1]$ is the primary door on $p_0[s_0]p_1[s_1]$, then $s_0 \leq c_0$ and $s_1 \leq c_1$.*

The invariant in Lemma 5.10 implies that $s_0 \leq c_0 < a_0 + n_0 < 2n_0$ and $s_1 \leq c_1 < a_1 + n_1 < 2n_1$ at the end of the “while” loop in Algorithm 2, which is the assertion of Lemma 4.5.

PROOF. Algorithm 2 starts with $s_0 = 0 \leq a_0 \leq c_0$ and $s_1 = 0 \leq a_1 \leq c_1$, so the invariant holds initially. Consider an update $s_u \leftarrow v_u$ in line 8 of Algorithm 2, where $u \in \{0, 1\}$, assuming that the invariant holds before the update. Let s_u denote the value before the update and w_u be as claimed by the second part of Lemma 4.3. Let $q_u = p_u[s_u]$, $q'_u = p_u[w_u]$, $q''_u = p_u[v_u]$, and $q_{1-u} = q'_{1-u} = q''_{1-u} = p_{1-u}[s_{1-u}]$. The conclusions of Lemma 4.3 imply that $q_u q'_u$ is a segment of $\mathcal{L}(q_0, q_1)$ not containing q_{1-u} , $P_u[q_u, q'_u]$ is a cap of $\text{RHP}(q_0, q_1)$, $P_u[q'_u, q''_u]$ is the single segment $q'_u q''_u$, and $q''_u \notin \text{RHP}(q_0, q_1) = \text{RHP}(q'_0, q'_1)$, where the last equality follows from $q'_{1-u} = q_{1-u} \in \mathcal{L}(q_0, q_1) \setminus q_u q'_u$. These conditions are what we need to apply Lemma 5.8 (to q_0, q_1, q'_0 , and q'_1) and Lemma 5.9 (to q'_0, q'_1, q''_0 , and q''_1). Let $z_0 z_1, z'_0 z'_1$, and $z''_0 z''_1$ be the primary doors on $q_0 q_1, q'_0 q'_1$, and $q''_0 q''_1$, respectively. Lemma 5.8 and Lemma 5.9 yield $z_0 z_1 = z'_0 z'_1 < z''_0 z''_1$. Let $c_0, c'_0 \in [a_0, a_0 + n_0)$ and $c_1, c'_1 \in [a_1, a_1 + n_1)$ be such that $p_0[c_0] = z_0, p_0[c'_0] = z'_0, p_1[c_1] = z_1$, and $p_1[c'_1] = z'_1$. This and $z_0 z_1 < z'_0 z'_1$ imply $c_0 < c'_0$ and $c_1 < c'_1$. This and the assumption that $s_0 \leq c_0$ and $s_1 \leq c_1$ (which is the invariant before the update) imply $s_0 < c'_0$ and $s_1 < c'_1$. This already gives one inequality of the invariant after the update, namely, $s_{1-u} \leq c'_{1-u}$. It remains to prove $v_u \leq c''_u$, which is the other inequality of the invariant after the update. Since $q''_u \notin \text{RHP}(q_0, q_1)$ and $q'_{1-u} = q_{1-u}$, we have $\text{RHP}(q_0, q_1) \cap q''_0 q''_1 = \{q'_{1-u}\}$. This and $P_u[q_u, q'_u] \subset \text{RHP}(q_0, q_1)$ imply $P_u[q_u, q'_u] \cap q''_0 q''_1 = \emptyset$. Since $P_u[q'_u, q''_u] = q'_u q''_u$ and $q'_u \notin q''_0 q''_1$, we have $P_u[q'_u, q''_u] \cap q''_0 q''_1 = \{q''_u\}$. Thus $P_u[q_u, q''_u] \cap q''_0 q''_1 = \{q''_u\}$. This and the fact that $p_u[c''_u] = z''_u \in P_u \cap q''_0 q''_1$ imply $c''_u \notin [s_u, v_u]$. This and $s_u < c'_u$ give the requested inequality $v_u \leq c''_u$. We conclude that the invariant is preserved at the considered update to s_u . \square

6 CONCLUDING REMARKS

So far, we were assuming that the combined set \hat{P} of corners of P_0 and P_1 contains no triple of collinear points, which guarantees that expressions of the form $\det^*(a, b, c)$ with $a, b, c \in \hat{P}$ evaluated in line 4 of Algorithm 1 and in lines 4 and 5 of Algorithm 2 are non-zero. Now, we adapt the algorithms to handle the case that \hat{P} may contain triples of collinear points. Consider the following family of transformations:

$$\phi_\epsilon : \mathbb{R}^2 \ni (x, y) \mapsto (x + \epsilon y, y + \epsilon(x + \epsilon y)^2) \in \mathbb{R}^2.$$

For any distinct points $a, b, c \in \mathbb{R}^2$, the expression $\det^*(\phi_\epsilon(a), \phi_\epsilon(b), \phi_\epsilon(c))$ is a continuous function of ϵ that attains value zero for finitely many arguments ϵ . Therefore, there is $\epsilon_0 > 0$ such that the sign of $\det^*(\phi_\epsilon(a), \phi_\epsilon(b), \phi_\epsilon(c))$ is equal to the same constant $\sigma(a, b, c) \in \{+1, -1\}$ for all $\epsilon \in (0, \epsilon_0)$. The value of $\sigma(a, b, c)$ can be easily computed from the coordinates of a, b , and c by treating ϵ as a symbolic positive infinitesimal. We can modify Algorithms 1 and 2 to use $\sigma(a, b, c)$ instead of $\det^*(a, b, c)$ for the tests in the aforementioned lines, so that $c \notin \mathcal{H}_k(a, b)$ means $\sigma(a, b, c) = \alpha_k$, and $z \in \Delta(a, b, c)$ means $\sigma(z, a, b) = \sigma(z, b, c) = \sigma(z, c, a)$. Conceptually, this is the same as invoking the algorithms on the polygons P_0 and P_1 transformed by ϕ_ϵ , where ϵ is small enough so that $\sigma(a, b, c) = \text{sgn } \det^*(\phi_\epsilon(a), \phi_\epsilon(b), \phi_\epsilon(c))$ for all triples of distinct points $a, b, c \in \hat{P}$. Therefore, if either algorithm claims to find a solution, the fact that it is correct on the transformed input implies that it is correct on the original input. The same reasoning (and the same modification of the two algorithms) can be applied with the following family of transformations ψ_ϵ instead of ϕ_ϵ :

$$\psi_\epsilon : \mathbb{R}^2 \ni (x, y) \mapsto (x + \epsilon y, y - \epsilon(x + \epsilon y)^2) \in \mathbb{R}^2.$$

Any straight line is transformed by ϕ_ϵ and ψ_ϵ (with ϵ small enough) into two curve lines bend in the opposite directions. This property is important when we want to find “degenerate” common tangents. Namely, if the convex hulls of P_0 and P_1 touch, then one of ϕ_ϵ and ψ_ϵ makes them overlap properly while the other makes them disjoint; therefore, one modification of Algorithm 1 or 2 finds the “degenerate” separating common tangent while the other does not. Similarly, if the convex

hulls of P_0 and P_1 are nested and their boundaries touch, then one of ϕ_ϵ and ψ_ϵ makes them overlap properly while the other moves the smaller convex hull to the interior of the larger; therefore, one modification of Algorithm 2 finds the “degenerate” outer common tangent while the other does not. We recognize that there is no solution when both modifications report no solution.

It remains open whether an outer common tangent of two polygons that are not disjoint can be found in linear time and constant workspace. Another natural question is whether the diameter of the convex hull of a simple polygon can be computed in linear time and constant workspace.

REFERENCES

- M. Abrahamsen. 2015. An optimal algorithm for the separating common tangents of two polygons. In *31st International Symposium on Computational Geometry (SoCG 2015) (LIPIcs)*, Vol. 34. 198–208. arXiv:1511.04036 (corrected version).
- M. Abrahamsen, M. de Berg, K. Buchin, M. Mehr, and A. D. Mehrabi. 2017. Minimum perimeter-sum partitions in the plane. In *33rd International Symposium on Computational Geometry (SoCG 2017) (LIPIcs)*, Vol. 77. 4:1–4:15.
- M. Abrahamsen and B. Walczak. 2016. Outer common tangents and nesting of convex hulls in linear time and constant workspace. In *24th Annual European Symposium on Algorithms (ESA 2016) (LIPIcs)*, Vol. 57. 4:1–4:15.
- T. Asano, W. Mulzer, G. Rote, and Y. Wang. 2011. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.* 2, 1 (2011), 46–68.
- B. Banyassady, M. Korman, and W. Mulzer. 2018. Computational Geometry Column 67. *SIGACT News* 49, 2 (2018), 77–94.
- L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. 2015. Space–time trade-offs for stack-based algorithms. *Algorithmica* 72, 4 (2015), 1097–1129.
- G. S. Brodal and R. Jacob. 2002. Dynamic planar convex hull. In *43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*. 617–626.
- E. Carson, J. Demmel, L. Grigori, N. Knight, P. Koanantakool, O. Schwartz, and H. V. Simhadri. 2016. Write-avoiding algorithms. In *30th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016)*. 648–658.
- S. Even and Y. Shiloach. 1981. An on-line edge-deletion problem. *J. ACM* 28, 1 (1981), 1–4.
- L. Guibas and J. Hershberger. 1989. Optimal shortest path queries in a simple polygon. *J. Comput. System Sci.* 39, 2 (1989), 126–152.
- L. Guibas, J. Hershberger, and J. Snoeyink. 1991. Compact interval trees: a data structure for convex hulls. *Int. J. Comput. Geom. Appl.* 1, 1 (1991), 1–22.
- J. Hershberger and S. Suri. 1992. Applications of a semi-dynamic convex hull algorithm. *BIT Numer. Math.* 32, 2 (1992), 249–267.
- J. Hershberger and S. Suri. 1995. A pedestrian approach to ray shooting: shoot a ray, take a walk. *J. Algorithms* 18, 3 (1995), 403–431.
- D. Kirkpatrick and J. Snoeyink. 1995. Computing common tangents without a separating line. In *4th International Workshop on Algorithms and Data Structures (WADS 1995) (LNCS)*, Vol. 955. Springer, 183–193.
- A. A. Melkman. 1987. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.* 25, 1 (1987), 11–12.
- M. H. Overmars and J. van Leeuwen. 1981. Maintenance of configurations in the plane. *J. Comput. System Sci.* 23, 2 (1981), 166–204.
- F. P. Preparata and S. J. Hong. 1977. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM* 20, 2 (1977), 87–93.
- O. Reingold. 2008. Undirected connectivity in log-space. *J. ACM* 55, 4 (2008), 17:1–17:24.
- G. T. Toussaint. 1983. Solving geometric problems with the rotating calipers. In *IEEE Mediterranean Electrotechnical Conference (MELECON 1983)*. A10.02/1–4.