Maximal Area Triangles in a Convex Polygon*

Kai Jin

1  Department of Computer Science, University of Hong Kong
cscjjk@gmail.com

Abstract
The widely known linear time algorithm for computing the maximum area triangle in a convex polygon was found incorrect by Keikha et. al. [15]. We present an alternative algorithm in this paper. Comparing to the only previously known correct solution, ours is much simpler and more efficient. Moreover, it can be easily extended to compute the minimum area triangle enclosing a convex polygon. Also, our new approach seems powerful in solving other related problems.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Maximum-area Triangles, Discrete Convex Geometry, Geometric Optimization, Computational Geometry, Convex Polygon

1 Introduction
In [15], the widely known linear time algorithm for computing the maximum area triangle in a given convex polygon $P$ (given in [10]) was found incorrect. An alternative algorithm given in [5] is also wrong for the same reason. Also, another algorithm (see [24]) not mentioned in [15] - perhaps because published by an informal journal - is also wrong (non-surprisingly and very badly wrong). Whether this problem can be solved in linear time was proposed as an open problem in [15], and is of significance because many follow-up results in computational geometry depend on finding the maximum area triangle (as a preprocessing step).

To study this problem, [15] introduced the 3-stable triangles, i.e. those triangles whose corners lie at $P$’s vertices and cannot be improved to a larger one by adjusting the position of one corner. They proved that there are only $O(n)$ such triangles since all of them interleave (see the definition below or in [15, 5]), where $n$ is the number of vertices in $P$. But it was then leaved as an open problem how to compute all the 3-stable triangles in $O(n)$ time.

In this paper we present a linear time algorithm for computing all the 3-stable triangles. However, via personal communication, the authors of [15] later point out that Chandran and Mount gave a correct linear time algorithm in [7]. So, our algorithm is not the first correct one. Nevertheless, as we will discuss below, ours is much simpler - almost straightforward to implement - and more efficient - the constant behind the asymptotic complexity is smaller.

More importantly, we use a new approach that we have never seen so far and it looks powerful in solving other polygonal inclusion or circumscribing problems. The following dual problem only admits $O(n \log n)$ time solutions in literature ([3, 25]) but can be solved in linear time by our new approach. (Reported by a subsequent paper.) Given $n$ half-planes which constitute a convex polygon, find three of them whose intersecting area is minimum. It is safe to conclude that some incentives are brought in for investigating whether even more efficient algorithms exist for some related polygonal inclusion or circumscribing problem.

* This work was partially supported by someone.
In addition, in the second part of this paper (Section 4), we extend our algorithm to compute all the generally 3-stable triangles, i.e. those triangles whose corners lie in $P$’s boundary and whose area cannot be improved through adjusting one corner. This further leads to a new linear time solution for computing the minimum enclosing triangle.

1.1 Previous known algorithm given in [7] and other related work.

We now briefly introduce Chandran and Mount’s method. They first define the $P$-stable triangles (outside $P$). All sides of a $P$-stable triangle must be touched by $P$. In particular, two of them must have their midpoints touched by $P$; these two are called the legs whereas the remaining one is called the base. Moreover, one of the following holds. (1) The base is flush with $P$ (namely, it contains an edge of $P$). (2) one of the legs is flush with an edge of $P$ and has as its midpoint a vertex of this edge. Note that if (1) holds, it is said $P$-anchored.

The $P$-anchored triangles are in fact introduced in [17], where it is proved that they contain all the local minimums of enclosing triangles of $P$. It is then proved in [22] that all the $P$-anchored triangles are “interspersing” - if we move the base to the next edge, the midpoints of two legs will both move in clockwise. Then, by using the rotating-caliper technique ([26]) together with some clever algorithmic tricks, [22] enumerates all the $P$-anchored triangles in linear time and thus find the minimum enclosing triangles. The algorithm is somewhat involved and description is highlighted and so details on implementation is given in [23].

Later, by using some “more involved” (as said in the paper) observations and techniques, [7] managed to enumerate all the $P$-stable triangles in linear time. Their algorithm is actually a terrific example of the rotating-caliper technique. Moreover, it was proved that by enumerating all the $P$-stable triangles, one can easily obtain all local maximums of triangles enclosed in $P$. To sum up, the known algorithm in [7] indeed computes all the maximum area triangles enclosed in $P$ and all the minimum area triangles enclosing $P$ simultaneously.

As comparison, our algorithm for computing the maximum area triangles is more direct - it does not need to consider all those triangles enclosing $P$ at all. Another major difference is that we do not apply the rotating-caliper technique. In the rotating-caliper approach, usually we first determine one object (like a corner) and then maintain the other objects correspondingly. In our approach, briefly we first determine a pair of objects (like two corners) and then maintain the other correspondingly. The difficulty of designing algorithms based on rotating-caliper technique lies in maintaining the corresponding corners efficiently, while our difficulty lies in determining the first pair of objects efficiently. See more in Subsection 1.3.

The problems of searching for extremal figures with special properties inside or outside a polygon were initiated in [10, 5, 8], and have since been studied extensively. The minimum area triangle enclosing a convex polygon can be found in $O(n \log^2 n)$ time [17] or even in $O(n)$ time [22]. The minimum perimeter triangle enclosing a convex polygon is solved in $O(n)$ time [4]. The maximum perimeter triangle enclosed by a convex polygon can be solved in $O(n \log n)$ time [5]. It is interesting to know whether this can be optimized to linear time.

As extended research of finding extremal triangles, [5, 2, 3, 8, 1, 21] studied extremal $k$-gon (with respect to area or perimeter) inside or outside a convex polygon. In particular, the maximum $k$-gon can be computed in $O(n \log n)$ time when $k$ is a constant. It is interesting to know whether this can be optimized to linear time (by our approach). Another direction is to find maximum area triangles inside a simple polygon [19]. Also, we can ask this type of problems in three dimensional space; pioneer research in this direction are [28, 27].

Instead of $k$-gon, people care about extremal figures that admit more geometric properties, such as equilateral triangle and squares, rectangles and parallelograms, disks and ellipses.
In particular, the largest-area parallelogram is studied by the same author of this paper in [14, 15]. We gave a very nontrivial algorithm for computing all locally maximal parallelograms in a convex polygon in $O(n \log^2 n)$ time. See [13] for more introductions in this area.

In addition, some interesting properties of figures inscribed or circumscribed in convex polygons or curves can be found in [3, 11, 9, 12, 18, 20] and the references within.

1.2 Preliminary

Let $v_1, \ldots, v_n$ be a clockwise enumeration of the vertices of the given convex polygon $P$. Assume that no three vertices lie in the same line. Let $AB$ denote the line defined by two distinct points $A, B$. For any vertex $A$, let $A - 1, A + 1$ respectively denote the clockwise previous and next vertex of $A$ (thus, for $A = v_i$, we have $A - 1 = v_{i-1}$ and $A + 1 = v_{i+1}$). Let $e_1, \ldots, e_n$ denote the $n$ edges of $P$, so that $e_i$ is the directed line segment $v_i \rightarrow v_{i+1}$.

Note: For any triangle that forms by points on the boundary of $P$, we always list its three corners in clockwise order; when $\triangle ABC$ is written, we assume $A, B, C$ lie in clockwise.

Consider any triangle $\triangle ABC$ in which all corners lie on the boundary of $P$. Corner $A$ is stable if it has largest distance to $BC$ among all points of $P$ on the right of $BC$; Similarly, $B$ is stable if it has largest distance to $CA$ among all points of $P$ on the right of $CA$; Similarly, $C$ is stable if it has largest distance to $AB$ among all points of $P$ on the right of $AB$. The triangle $\triangle ABC$ is said 3-stable, if all its three corners are stable and lie on the vertices of $P$.

A triangle is said a Locally Maximal Area Triangle (LMAT), if all of its corners lie on vertices of $P$ and it has a larger or equal area comparing to its neighboring triangles. This means there exists $\varepsilon > 0$ such that the area of $T$ is at least the area of $T'$ for each triangle $T'$ that is at a Hausdorff distance less than $\varepsilon$ from $T$. Obviously, an LMAT must be 3-stable. (But 3-stable does not imply an LMAT. This is addressed more clearly in Appendix [3] yet not important for understanding the main result.) In our algorithm, we actually compute all 3-stable triangles (and thus all LMATs). There are $O(n)$ 3-stable triangles due to:

Lemma 1. Any two 3-stable triangles in a convex polygon are interleaving.

Later in the paper Lemma 1 will be enhanced to Lemma 12 (proved in Appendix A).

1.3 Technique overview

We start by computing one 3-stable triangle of $P$ using a trivial and standard method; denote the result 3-stable triangle by $\triangle v_s,v_t,v_r$. By Lemma 1, any 3-stable triangle interleaves $\triangle v_s,v_t,v_r$, and hence it has an edge $BC$ such that $B \in \{v_s, \ldots, v_t\}$ whereas $C \in \{v_t, \ldots, v_r\}$. We then compute all the 3-stable triangles in a process called Rotate-and-Kill. Initially, it sets two pointers $(B, C) = (v_s, v_t)$. In each iteration, we first compute $A$ so that $A$ has the largest distance to $BC$. This only costs amortized $O(1)$ time because the slope of $BC$ will keep decreasing and so $A$ goes only in clockwise direction. Then, check whether $\triangle ABC$ is 3-stable and report it if so. After that we go to the next iteration by either killing $B$ (i.e. moving pointer $B$ to its next vertex) or killing $C$ (i.e. moving pointer $C$ to its next vertex). We have to make sure that $B$ is killed only when its related pairs $(B, C + 1), (B, C + 2), \ldots, (B, v_r)$ cannot form an edge of any 3-stable triangle; and $C$ is killed only when its related pairs $(B + 1, C), (B + 2, C), \ldots (v_t, C)$ cannot form an edge of any 3-stable triangle. Thus our algorithm will not miss any 3-stable triangles. Eventually, $(B, C)$ reaches $(v_t, v_r)$ and we terminate the Rotate-and-Kill process. The entire process runs in linear time because the decision condition we applied for killing $B$ or $C$ is amazingly simple and can be computed in $O(1)$ time, and we note that the key of our approach lies in designing this condition.
Compute one 3-stable triangle

In this section, we show how to compute one 3-stable triangle.

Step 1. Choose $A$ to be an arbitrary vertex of $P$; say $A = v_1$. Find $B, C$ so that $ABC$ is the largest triangle rooted at $A$. This can be done in $O(n)$ time. We can enumerate a vertex $B$ and maintain $C_B$ in amortized $O(1)$ time so that $C_B$ is the vertex with the largest distance to $\overrightarrow{AB}$ among all vertices on the right of $\overrightarrow{AB}$. Then, select $B$ so that $ABC_B$ is maximum.

Since $ABC$ is the largest triangle rooted at $A$, corners $B, C$ are stable. Moreover, if corner $A$ is also stable, $\triangle ABC$ is 3-stable and we can go to the next section.

Now, assume that $A$ is not stable. Moreover, assume that $A + 1$ has a larger distance than $A$ in the distance to $\overrightarrow{BC}$. Otherwise $A - 1$ has a larger distance than $A$ and it is symmetric.

Step 2. This step is presented in Algorithm 1.

```
1 A ← A + 1;
2 repeat
3   while (B + 1 > B in the distance to $\overrightarrow{AC}$) do
4     B ← B + 1;
5   end
6   while (C + 1 > C in the distance to $\overrightarrow{AB}$) do
7     C ← C + 1;
8   end
9 until B + 1 <= B and C + 1 <= C;
```

Algorithm 1: Algorithm for Step 2

To distinguish, we denote the value of the three pointers $(A, B, C)$ at the end phase of this algorithm by $(A_1, B_1, C_1)$; and the value at the beginning phase by $(A_0, B_0, C_0)$.

![Figure 1](image_url)

Figure 1 Illustration of Step 2 and Step 3

Observation 2. Corners $B_1, C_1$ are stable in $\triangle A_1 B_1 C_1$.

Proof. It reduces to prove the following arguments.
1. $B_1$ is at least as far as $B_1 - 1$ in the distance to $\overrightarrow{A_1 C_1}$.
2. $B_1$ is at least as far as $B_1 + 1$ in the distance to $\overrightarrow{A_1 C_1}$.
3. $C_1$ is at least as far as $C_1 - 1$ in the distance to $\overrightarrow{A_1 B_1}$.
4. $C_1$ is at least as far as $C_1 + 1$ in the distance to $\overrightarrow{A_1 B_1}$. 
Due to the termination condition of the algorithm, (2) and (4) hold. In the following we point out two facts that hold throughout the algorithm, which respectively imply (1) and (3).

(i) $B$ is at least as far as $B - 1$ in the distance to $\overrightarrow{AC}$.
(ii) $C$ is at least as far as $C - 1$ in the distance to $\overrightarrow{AB}$.

These facts hold at the beginning, because $B_0, C_0$ are stable in $A_0B_0C_0$. When $A, B$ or $C$ are increased by 1 (at Line 1,4,7), these facts still hold as illustrated in Figure 1(b),(c),(d).

**Step 3.** So far, we obtain $\triangle A_1B_1C_1$ where $B_1, C_1$ are stable. If $A_1$ is also stable, $\triangle A_1B_1C_1$ is 3-stable and we proceed to next section. Now, consider the case where $A_1$ is not stable. This implies that $A + 2$ has a larger distance than $A + 1$ in the distance to $\overrightarrow{B_1C_1}$ (see Figure 1).

We call Algorithm 1 again with initial value $(A_1, B_1, C_1)$ and terminal value $(A_2, B_2, C_2)$. Furthermore, we repeat such a process until $\triangle A_iB_iC_i$ is 3-stable for some integer $i$.

**Analysis of correctness and running time.** At every change of $A, B, C$ in Algorithm 1 the area of $\triangle ABC$ increases. Therefore, the above process terminates eventually. Moreover, it runs in $O(n)$ time because pointers $A, B, C$ can only move in the clockwise direction and $A$ cannot return to $A_0$ since $A_0B_0C_0$ is the largest triangle rooted at $A_0$.

### 3. Compute all the 3-stable triangles

In this section, we assume $\triangle v_iv_jv_k$ is 3-stable and we compute all 3-stable triangles by a Rotate-and-Kill process as mentioned in Subsection 1.3. We start by an observation.

A vertex pair $(v_j, v_k)$ is **dead** if it cannot form an edge of any 3-stable triangle; precisely, if there is no vertex $v_i$ such that $\triangle v_iv_jv_k$ is 3-stable and $v_i, v_j, v_k$ lie in clockwise order.

**Observation 3.** Assume $B \in \{v_2, \ldots, v_t\}$, $C \in \{v_3, \ldots, v_r\}$, and $B \neq C$. Then, either

1. $(B, C + 1), (B, C + 2), \ldots, (B, v_r)$ are all dead; or
2. $(B + 1, C), (B + 2, C), \ldots, (v_t, C)$ are all dead.

**Proof.** Suppose that (1) and (2) are both false. Without loss of generality, assume that $(B, C')$ and $(B', C)$ are not dead. This implies that there exist $A_1, A_2$ so that $\triangle A_1B'C'$ and $\triangle A_2B'C$ are 3-stable. By the assumption of $B'$ and $C'$, we know that $\triangle A_1B'C'$ does not interleave $\triangle A_2B'C$. So at least one of them is not 3-stable due to Lemma 1. Contradictory.

This observation is important. It implies that such a Rotate-and-Kill process as described in Subsection 1.3 may exist! More clearly, in each intermediate iteration, we always have an available action to take - we can kill $B$ when (1) holds and kill $C$ when (2) holds.

The next question is how to determine (1) and (2) efficiently. In order to design a linear time algorithm, we need a condition which can guide us determine (1) and (2) and which can be computed in $O(1)$ time. Such a delightful condition is presented below. (We want to mention that there exist more direct methods to determine (1) and (2) in $O(\log n)$ time.)

#### 3.1 More preliminaries

**Definition 4.** Assume $v_j, v_k$ are distinct vertices of $P$. See Figure 2.

Let $r_1, r_2$ respectively denote the ray at $v_j$ which have the same direction as $e_{k-1}, e_k$. Let $r_1', r_2'$ respectively denote the ray at $v_k$ which have the opposite direction to $e_{j-1}, e_j$. These four rays together define the region $Q_{j,k}$. The intersecting point between $r_1, r_1'$ is denoted by $H_{j,k}$. The intersecting point between $r_2, r_2'$ is denoted by $I_{j,k}$. The intersecting point between $r_1, r_2'$ is denoted by $J_{j,k}$. The intersecting point between $r_2, r_1'$ is denoted by $K_{j,k}$.
Note: When $I_{j,k}$ (or $H_{j,k}, J_{j,k}, K_{j,k}$) is undefined in the above (when the corresponding two rays do not intersect), we define $I_{j,k} = \infty$ (or $H_{j,k} = \infty, J_{j,k} = \infty, K_{j,k} = \infty$, respectively) and assume that it has an infinite large distance to $\frac{v_jv_k}{v_jv_k}$. We consider that the region $Q_{j,k}$ contains its boundary. We consider that polygon $P$ also contains its boundary.

![Figure 2](image-url) Definition of the area $Q_{j,k}$.

Roughly speaking, $Q_{j,k}$ indicate the potential region of a corner in a 3-stable triangle when the other two corners are fixed on $v_j$ and $v_k$. This is made precisely as follows.

**Observation 5.** When $v_j, v_k$ are stable in $\triangle v_l v_j v_k$, point $v_i$ lies in $Q_{j,k}$.

(Recall that whenever we write $\triangle v_l v_j v_k$, we assume that $v_i, v_j, v_k$ lie in clockwise order.)

**Proof.** Because $v_j$ is stable, $v_i$ can only lie in the sector area bounded by $r'_1$ and $r'_2$. Because $v_k$ is stable, $v_i$ can only lie in the sector area bounded by $r_1$ and $r_2$. Together, $v_i$ lies in the intersection of these two areas, which is defined as $Q_{j,k}$.

**Definition 6.** We regard vertex pair $(v_j, v_k)$ **illegal** if $v_{k+1}$ is closer than $v_k$ in the distance to $\frac{v_jv_k}{v_jv_k}$ (as shown in Figure 3a) and regard $(v_j, v_k)$ **legal** otherwise (see Figure 3b).

Note: For each vertex $v_j$ of $P$, the sequence of vertex pairs $(v_j, v_{j+1}), (v_j, v_{j+2}), \ldots, (v_j, v_{j-1})$ starts with several legal pairs and then is followed by several illegal pairs. In particular, $(v_j, v_{j+1})$ is legal and $(v_j, v_{j-1})$ is illegal. (All subscripts taken modulo $n$.)

![Figure 3](image-url) Illustration of the notion of “legal pair”.

**Observation 7 (Some conditions when $(v_j, v_k)$ is dead).**

1. If $Q_{j,k}$ does not intersect $P$, then $(v_j, v_k)$ is dead.
2. When $(v_j, v_k)$ is illegal, $Q_{j,k}$ does not intersect $P$ and hence is dead.
3. If some point $A$ lies in $P$ and it lies on the right of $\frac{v_jv_k}{v_jv_k}$ and $A > K_{j,k}$ in the distance to $\frac{v_jv_k}{v_jv_k}$, then we can determine that $(v_j, v_k)$ is dead.

**Proof.** 1. If $(v_j, v_k)$ is not dead, there exists $v_i$ such that $\triangle v_l v_j v_k$ is 3-stable. By Observation 5, $Q_{j,k}$ contains $v_i$ and hence intersects $P$.

2. Note that $(v_j, v_k)$ is illegal means $k \neq j + 1$ because $(v_j, v_{j+1})$ is always legal. Thus $v_k$ does not lie in $Q_{j,k}$. Moreover, since $(v_j, v_k)$ is illegal, the area bounded by $r'_1, r'_2$ (except apex $v_k$) lies outside $P$ (see Figure 3c). Together, $Q_{j,k}$ lie outside $P$ and is thus dead.

3. Suppose to the opposite that it is not dead but $\triangle v_j v_k$ is 3-stable. By Observation 7, $X \in Q_{j,k}$. It follows that $X < K_{j,k}$ in the distance to $\frac{v_jv_k}{v_jv_k}$ (see Figure 2(a)). Therefore, $X > A$ in the distance to $\frac{v_jv_k}{v_jv_k}$, and so $X$ is not stable in $\triangle v_j v_k$. Contradictory.
3.2 Rotate-and-Kill process

| **Input:** | $r, s, t$ such that $v_r, v_s, v_t$ is 3-stable. |
|------------|---------------------------------------------|
| **1**      | $(B, C) \leftarrow (v_s, v_t); A \leftarrow v_r; $ |
| **2**      | **repeat**                                 |
| **3**      | while $A + 1 > A$ in the distance to $\overrightarrow{BC}$ do |
| **4**      | $A \leftarrow A + 1;$                      |
| **5**      | **end**                                    |
| **6**      | Output $ABC$ if it is 3-stable; Output $(A + 1)BC$ if it is 3-stable; |
| **7**      | if $A > I_{BC}$ in the distance to $\overrightarrow{BC}$ then |
| **8**      | $C \leftarrow C + 1;$                      |
| **9**      | **else**                                   |
| **10**     | $B \leftarrow B + 1;$                      |
| **11**     | **end**                                    |
| **12**     | **until** $(B, C) = (v_r, v_t);$            |

**Algorithm 2:** The Rotate-and-Kill process

We present the Rotate-and-Kill process in Algorithm 2. An example is shown at the end of this section. The idea of this process and the analysis of its running time is briefly discussed in Section 1.3. The correctness is assured by the following lemma.

**Lemma 8.**
1. At Line 10, $(B, C + 1), (B, C + 2), \ldots, (B, v_t)$ are dead.
2. At Line 5, $(B + 1, C), (B + 2, C), \ldots, (v_t, C)$ are dead.
3. Throughout the “repeat” statement, $B \in \{v_s, \ldots, v_t\}$, $C \in \{v_t, \ldots, v_r\}$, and $B \neq C$.

Moreover, the “repeat” statement will terminate successfully at $(B, C) = (v_r, v_t)$.

**Proof.**
3. For ease of understanding, we first prove Claim 3, assuming that Claim 1, 2 are correct. Notice that $(v_t, v_r)$ forms an edge of a 3-stable triangle (that is $\triangle v_r v_s v_t$). Therefore, Claim 1 and 2 promise that $B$ would not be killed when $(B = v_t, C < v_t)$, and $C$ would not be killed when $(B < v_t, C = v_r)$. So $(B, C)$ eventually become $(v_t, v_r)$, at which moment the algorithm terminates, and we have $B \in \{v_s, \ldots, v_t\}$ and $C \in \{v_t, \ldots, v_r\}$ throughout.

In addition, we argue that $B \neq C$ throughout. Suppose $B = C$, it must be $B = C = v_t$. This means that we have killed $B$ at $(B, C) = (v_t-1, v_t)$. But this is impossible by the following analysis. We have $I_{BC} = 0$ when $B, C$ are two adjacent vertices, and so “$A > I_{BC}$” holds. So we would enter Line 8 and kill $C$ rather than kill $B$ when $(B, C) = (v_t-1, v_t)$.

1. Assume we are at Line 10 and so $A \leq I_{BC}$ in the distance to $\overrightarrow{BC}$. Further assume that $C$ is closer than $C + 1$ in the distance to $\overrightarrow{B(B + 1)}$, as shown in Figure 1. Otherwise, $(B, C + 1), \ldots, (B, v_r)$ are all illegal and hence are dead (by Observation 7).

Let $l$ denote the unique line at $I_{BC}$ that is parallel to $\overrightarrow{BC}$. Let $C^*$ denote the last vertex in the sequence $C + 1, \ldots, v_r$ so that $(B, C^*)$ is legal. We first state two facts.

a. Polygon $P$ lies in the (closed) half-plane delimited by $l$ and containing $BC$.

b. $Q_{B,C+1}, \ldots, Q_{B,C^*}$ lie in the (open) half-plane delimited by $l$ and not containing $BC$.

Combining Fact a and b, no region in $\{Q_{B,C+1}, \ldots, Q_{B,C^*}\}$ intersects $P$. So $(B, C + 1), \ldots, (B, C^*)$ are dead according to Observation 7. Moreover, $\{(B, C^* + 1), \ldots, (B, v_r)\}$ are dead because of illegal (by Observation 7). Together, Claim 1 holds.

**Proof of a.** In the distance to $\overrightarrow{BC}$, we know $A \leq I_{BC}$ whereas $A$ has the largest distance among all vertices of $P$ that lie on the right of $\overrightarrow{BC}$, so polygon $P$ is bounded by $l$. 2017
Proof of b. Here, we define a region $R_{j,k}$ for each legal pair $(v_j, v_k)$ so that $k \neq j + 1$. Recall $H_{j,k}, I_{j,k}, J_{j,k}$ in Definition 4. See Figure 2 (a) and (b). The unique quadrant bounded by $J_{j,k}I_{j,k}, J_{j,k}H_{j,k}$ and containing $Q_{j,k}$ is defined as $R_{j,k}$. Observe Figure 4 (b), we have $R_{B,C} \supseteq R_{B,C+2} \supseteq \ldots \supseteq R_{B,C}$. It implies that $Q_{B,C+1}, Q_{B,C+2}, \ldots, Q_{B,C}$ lie in $R_{B,C+1}$. Now, it reduces to show that $R_{B,C+1}$ lies entirely in the (open) half-plane delimited by $l$ and not containing $BC$ (see Figure 4 (a)). This holds because apex $J_{B,C+1}$ of $R_{B,C+1}$ lies in that half-plane, which is because $I_{B,C}J_{B,C+1}$ is a translate of $C(C+1)$ and $C$ is closer than $C+1$ to $B(B+1)$.

2. Now, we are at Line 8, and so $A > I_{B,C}$ in the distance to $BC$. Since $A > I_{B,C}$, we have $I_{B,C} \neq \infty$, so $C$ is closer than $C+1$ in the distance to $B(B+1)$, as shown in Figure 5.

Let $D$ denote the intersection between $CC+1$ and $l$. Let $d_{B,C}(X)$ denote the distance from $X$ to $BC$ for any vertex pair $(B, C)$. See Figure 5 (a). We claim two inequalities:

I $d_{B+1,C}(D) < d_{B+1,C}(A)$.  
II $d_{B+1,C}(K_{B+1,C}) = d_{B+1,C}(D)$.

Together, $d_{B+1,C}(A) > d_{B+1,C}(K_{B+1,C})$, thus $(B+1, C)$ is dead by Observation 7.3.

Proof of I. Since $A > I_{B,C}$ in the distance to $BC$, it lies in the (open) half-plane delimited by $l$ and not containing $BC$. Moreover, since $P$ is convex, $A$ lies in the (closed) half-plane delimited by $CC+1$ and containing $P$. Together, we obtain (I).

Proof of II. Obviously, both segment $(B+1)K_{B+1,C}$ and $CD$ are translates of $BI_{B,C}$. So, $(B+1)K_{B+1,C}DC$ is a parallelogram. This implies (II).
Next, we argue that \((B + 2, C)\) is dead. This reduces to prove the following:

(III) \(d_{B+2,C}(A) > d_{B+2,C}(K_{B+2,C})\).

(III) is implied by the above inequality \(d_{B+1,C}(A) > d_{B+1,C}(K_{B+1,C})\). This is simply illustrated in Figure 5(b). The proof is similar to the proofs of (I) and (II) and is omitted.

Furthermore, by induction, \((B + 1, C), (B + 2, C), \ldots, (v_t, C)\) are all dead.

\[\text{Remark.}\] To argue that \((B + 1, C), \ldots, (v_t, C)\) are dead, we may first try to apply Observation 7.1, that is, for example, by showing that \(Q_{B+1,C}\) did not contain points of \(P\), we deduce that \((B + 1, C)\) is dead. However, this argument does not work because \(Q_{B+1,C}\) may contain points (and vertices) of \(P\) indeed. The correct argument applies Observation 7.3.

4 Compute all the generally 3-stable triangles

In this section, we introduce generally 3-stable triangles and compute them in linear time. As application, this leads to a linear time algorithm for computing minimum enclosing triangles.

\[\text{Figure 6 Example of Rotate-and-Kill process.}\]
4.1 Definition and observations of generally 3-stable triangles

Recall that \( v_1, \ldots, v_n \) are vertices of \( P \) and \( e_1, \ldots, e_n \) are edges of \( P \). We call each vertex or edge a unit of \( P \). We denote the boundary of \( P \) by \( \partial P \). We regard that the edges do not contain their endpoints. Therefore, each point in \( \partial P \) belongs to a unique unit.

Recall that \( \triangle ABC \) is 3-stable if the three corners \( A, B, C \) are stable in \( \triangle ABC \) and lie on some vertices of \( P \). In the following we define a superset called generally 3-stable triangles. Basically, it only removes the requirement of lying on the vertices. However, there could be infinite many of such triangles and we select representatives from each equivalent class.

Definition 9 (Generally 3-stable triangles). Assume that there is a triangle whose corners are stable and lie in \( e_i, e_j, e_k \) respectively (in clockwise). It is clear that \( e_j \) is parallel to \( e_i \) as shown in Figure 7(a), otherwise the corner in \( e_i \) is not stable. Let the intersecting segment of \( e_i \) and \( Q_{jk} \) be denoted by \( X_1X_2 \). Applying Observation 6 for each point \( A \) in \( e_i \), the corners of \( \triangle Av_i v_k \) are stable if and only if \( A \) lies in \( X_1X_2 \). Moreover, the areas of \( \triangle Av_i v_k \) are all the same for any \( A \in X_1X_2 \), so these triangles constitute an equivalent class.

Assume \( \triangle ABC \) has all its corners stable in the triangle and has all its corners lying on \( \partial P \). If the number of vertex-type corners in \( \{A, B, C\} \) is not 2, or the number is 2 but \( \triangle ABC \) is the representative in its equivalent class, we consider \( \triangle ABC \) generally 3-stable.

Note: The number of vertex-type corners in a generally 3-stable triangle may be 0 to 3. See Figure 7 for examples. In addition, the 3-stable triangles must be generally 3-stable; they are exactly those generally 3-stable triangles with 3 vertex-type corners.

Figures 7 Illustration for generally 3-stable triangles.

Observation 10. Assume some generally 3-stable triangle has corners lying on \( v_i, e_j, e_k \) (note \( v_i, e_j, e_k \) are in clockwise order). Then this triangle can be determined as follows. It must be \( \triangle ABC \), where \( A = v_i \), \( B \) is the unique point in \( e_j \) such that \( v_iB \parallel e_k \), and \( C \) is the unique point in \( e_k \) such that \( v_iC \parallel e_j \). See Figure 7(b).

Observation 11. Assume some generally 3-stable triangle has corners lying on \( e_i, e_j, e_k \) (note \( e_i, e_j, e_k \) are in clockwise order). Then this triangle can be determined as follows. Let \( I_1, I_2, I_3 \) respectively denote the intersection of \( e_i, e_j \), the intersection of \( e_j, e_k \), and the intersection of \( e_k, e_i \). The triangle must be \( \triangle ABC \), where \( A \) is the mid point of \( I_1, I_3 \), \( B \) is the mid point of \( I_1, I_2 \), and \( C \) is the mid point of \( I_2, I_3 \). See Figure 7(c).

Proof. Since \( A, B, C \) are stable, \( AB \parallel e_k, BC \parallel e_i \), and \( CA \parallel e_j \). Therefore, \( |I_3A| : |AI_1| = |I_2B| : |BI_1| = |I_2C| : |CI_1| = |I_4A| : |AI_3| \). Therefore, \( |I_3A| = |AI_1| \), thus \( A \) is the mid point of \( I_1, I_3 \). Symmetrically, \( B, C \) are the mid points of \( I_1, I_2 \) and \( I_2, I_3 \). △
4.2 Extended or parallel version of Lemma 1 and Observation 5, 7

Lemma 12. Any two generally 3-stable triangles in a convex polygon are interleaving.

See its easy proof in Appendix A.

Definition 13. We say unit pair \((u_1, u_2)\) is \(G\)-dead if there is no generally 3-stable triangle \(\triangle ABC\) such that \(B \in u_1\) and \(C \in u_2\). (Be aware that all edge of \(P\) are endpoint-exclusive.)

Note: \(G\)-dead is different from and stronger than dead for \((v_j, v_k)\). It clearly implies dead but the reverse is false. For example, in Figure 8(a), \((v_j, v_k)\) is dead but not \(G\)-dead.

The unit pairs \((e_j, e_k), (v_j, e_k), (e_j, v_k), (v_j, v_k)\) are \textit{illegal} when \(v_{k+1}\) is closer than \(v_k\) in the distance to \(v_j v_{j+1}\) (as shown in Figure 9), and are \textit{legal} otherwise. (See also Definition 8.)

Note: For each edge \(e_j\) of \(P\), the sequence of edge pairs \((e_j, e_{j+1}), (e_j, e_{j+2}), \ldots, (e_j, e_{j-1})\) starts with several legal pairs and then is followed by several illegal pairs. In particular, \((e_j, e_{j+1})\) is legal and \((e_j, e_{j-1})\) is illegal. (All subscripts taken modulo \(n\.)

\[\begin{align*}
&(a)(e_j, e_k) \\
&(b)(v_j, e_k) \\
&(c)(e_j, v_k)
\end{align*}\]

Figure 8 illegal unit pairs.

Definition 14. For each edge pair \((e_j, e_k)\), we define \(Q(e_j, e_k)\) as the intersecting area of the following two strip regions. The first one is defined by two lines parallel to \(e_k\) which are at \(v_j, v_{j+1}\) respectively. The second one is by two lines parallel to \(e_j\) which are at \(v_k, v_{k+1}\) respectively. See Figure 9. We regard that \(Q(e_j, e_k)\) does \textit{not} contain its boundaries.

Note: According to the definition, \(Q(e_j, e_k)\) is empty when \(e_j\) is parallel to \(e_k\).

\[\begin{align*}
&(a)Q(e_j, e_k) \\
&(b)Q(e_j, e_k) \\
&(c)Q(e_j, e_k) \\
&(d)A \text{ special case: } k=j+1
\end{align*}\]

Figure 9 Illustration of the definition of \(Q(e_j, e_k)\).

Observation 15.
1. When \(B, C\) are stable in \(\triangle ABC\) and \(B \in e_j, C \in e_k\), then \(A \in Q(e_j, e_k)\).
2. If an edge pair \((e_j, e_k)\) is illegal, it is \(G\)-dead.
   Indeed, every illegal unit pair is \(G\)-dead. (We do not apply it below; so, proof omitted).
3. Assume that \(J_{j,k} \neq \infty\), and there is a point \(A \in \partial P\) and on the right of \(\overrightarrow{v_j v_k}\) such that \(A \geq J_{j,k}\) in the distance to \(v_j v_k\), then we claim that \((e_j, e_{k-1})\) is \(G\)-dead.

Proof. 1. Since \(B\) is stable and lies in \(e_j\), we know \(AC \parallel e_j\). Since \(C\) is stable and lies in \(e_k\), we know \(AB \parallel e_k\). Together, it follows that \(A \in Q(e_j, e_k)\).

2. Suppose to the opposite that \((e_j, e_k)\) is not \(G\)-dead. There exists a generally 3-stable \(\triangle ABC\) (where \(A, B, C\) lie in clockwise order) such that \(B \in e_j\) and \(C \in e_k\). Applying
Claim 1, $A \in Q(e_j, e_k) \cap \partial P$. Further since $(e_j, e_k)$ is illegal (see Figure 9(b)), objects $A, e_j, e_k$ (and so the points $A, B, C$) lie in counterclockwise order in $\partial P$. Contradictory!

3. We prove it by contradiction. Suppose to the opposite that it is not G-dead and thus there exists a generally 3-stable $\triangle A^* BC$ such that $B \in e_j$ and $C \in e_{k-1}$. We claim that $A > A^*$ in the distance to $BC$, and so $A^*$ is not stable in $\triangle A^* BC$. See Figure 10. Make a parallel line of $BC$ at $A^*$ and assume it intersects the extended lines of $e_j, e_{k-1}$ at $X_{A^*}, Y_{A^*}$. Make a parallel line of $\overline{v^*_jv^*_k}$ at $J_{j,k}$ and assume it intersects the two extended lines at $X_{J}, Y_{J}$. Assume the two extended lines intersect at $O$. The above claim simply follows from the following arguments: (1) $A$ lies in the shadow area defined by $X_{A^*}Y_{A^*}$ and the extended lines of $e_j, e_{k-1}$. (This is because $A \geq J_{j,k}$.) (2) $|OX_{A^*}| < |OX_j|$ and $|OY_{A^*}| < |OY_j|$. (This is because $|OX_{A^*}| = 2|OB| < 2|Ov_j| = |OX_j|$ and $|OY_{A^*}| = 2|OC| < 2|Ov_k| = |OY_j|$.) ▶

4.3 Generalized Rotate-and-Kill process

Assume that $\triangle v_r v_s v_t$ is 3-stable and $r, s, t$ are computed. Denote

$$U(r, s, t) = \{(u_1, u_2) \mid u_1 \in \{v_s, e_s, \ldots, e_{t-1}, v_t\} \text{ and } u_2 \in \{v_t, e_t, \ldots, e_{r-1}, v_r\}\}.$$  (1)

Lemma 16. If $(u_1, u_2)$ is not G-dead, then it belongs to $U(r, s, t) \cup U(s, t, r) \cup U(t, r, s)$.

Lemma 16 simply follows from Lemma 12. Its proof is deferred to Appendix C.

Notice that the pairs in $U(r, s, t)$ can be arranged into a two dimensional table in a standard way. In the next, we describe a (generalized) Rotate-and-Kill process which visits a small subset in $U(r, s, t)$. It proceeds in a way which corresponds to a monotonic path from the top-left cell $(v_s, v_t)$ to the bottom-right cell $(v_t, v_r)$ in the table. Hence only $O(n)$ pairs are visited. More importantly, after the process the following properties hold.

(*) All the vertex pairs and edge pairs in $U(r, s, t)$ which are not G-dead are visited.

(**) All the unit pairs in $U(r, s, t)$ which are not G-dead are visited.

The correctness of our algorithm only depends on (*), so we omit the proof of (**) in this paper. We state (**) because an alternative algorithm (discussed later) depends on (**).

By calling the generalized Rotate-and-Kill process three times, with parameters $(r, s, t), (s, t, r), \text{ and } (t, r, s)$, the vertex pairs and edge pairs that are not G-dead are obtained according to (*) and Lemma 16. After computing all these pairs, we can further compute all the generally 3-stable triangles. This is mainly because each generally 3-stable triangle has two corners both vertex-type or both edge-type. We give the details in the next subsection.

We now present the generalized Rotate-and-Kill process in Algorithm 3.

The sentence $u + +$ means to move pointer $u$ to its clockwise next unit. The terminal endpoint (TermEnd) of a unit is defined as: $\text{TermEnd}(e_i) := v_{i+1}$ and $\text{TermEnd}(v_i) = v_i$. 
1. Let \( v_j = \text{TermEnd}(u_1) \) (so \( u_1 = v_j \) or \( u_1 = e_{j-1} \));
2. Let \( v_k = \text{TermEnd}(u_2) \) (so \( u_2 = v_k \) or \( u_2 = e_{k-1} \));

while \( A + 1 > A \) in the distance to \( \tilde{v}_j \tilde{v}_k \) do

- \( A \leftarrow A + 1; \)

end

switch \((u_1, u_2)\) do

- case \((v_j, v_k)\) do
  - \( u_1 + + \) if \( A \leq I_{j,k} \) in the distance to \( \tilde{v}_j \tilde{v}_k \) and otherwise \( u_2 + + \);

- case \((e_{j-1}, e_{k-1})\) do
  - \( u_1 + + \) if \( A \leq H_{j,k} \) in the distance to \( \tilde{v}_j \tilde{v}_k \) and otherwise \( u_2 + + \);

- case \((v_j, e_{k-1})\) do
  - \( u_1 + + \) if \( A < J_{j,k} \) in the distance to \( \tilde{v}_j \tilde{v}_k \) and otherwise \( u_2 + + \);

- case \((e_{j-1}, v_k)\) do
  - \( u_1 + + \) if \( A \leq K_{j,k} \) in the distance to \( \tilde{v}_j \tilde{v}_k \) and otherwise \( u_2 + + \);

end

end until \((u_1, u_2) = (v_1, v_r)\);

Algorithm 3: The generalized Rotate-and-Kill process

**Note.** In the vertex-edge case \("(u_1, u_2) = (v_j, e_{k-1})"\) the condition is \("<\", whereas in the other three cases it is \("\leq\"\). This is not a typo although it looks like one for the readers.

**Lemma 17.** Recall that all unit pairs in set \(U(r, s, t)\) (defined in Equation 7) can be arranged into a two dimensional table in such a way that the rows from top to bottom correspond to \(v_r, v_{r+1}, \ldots, v_t\) while the columns from left to right correspond to \(v_t, e_1, \ldots, e_r\).

1. In Algorithm 3, when \(u_1\) is to be killed, the vertex pairs and edge pairs on the right of the current row are all G-dead. When \(u_2\) is to be killed, the vertex pairs and edge pairs on the bottom of the current column are all G-dead. More precisely, we claim the following.

- **VV1.** When \((u_1, u_2) = (v_j, v_k)\) and \(A \leq I_{j,k}\), \((v_j, v_k)\) is G-dead for \(k' \in [k + 1, r]\).

- **VV2.** When \((u_1, u_2) = (v_j, v_k)\) and \(A > I_{j,k}\), \((v_j, v_k)\) is G-dead for \(j' \in [j + 1, t]\).

- **EE1.** When \((u_1, u_2) = (e_{j-1}, e_{k-1})\) and \(A \leq H_{j,k}\), \((e_{j-1}, e_{k-1})\) is G-dead for \(k' \in [k, r - 1]\).

- **EE2.** When \((u_1, u_2) = (e_{j-1}, e_{k-1})\) and \(A > H_{j,k}\), \((e_{j-1}, e_{k-1})\) is G-dead for \(j' \in [j, t - 1]\).

- **VE1.** When \((u_1, u_2) = (v_j, e_{k-1})\) and \(A < J_{j,k}\), \((v_j, e_{k-1})\) is G-dead for \(k' \in [k, r]\).

- **VE2.** When \((u_1, u_2) = (v_j, e_{k-1})\) and \(A > J_{j,k}\), \((v_j, e_{k-1})\) is G-dead for \(j' \in [j, t - 1]\).

- **EV1.** When \((u_1, u_2) = (e_{j-1}, v_k)\) and \(A \leq K_{j,k}\), \((e_{j-1}, v_k)\) is G-dead for \(k' \in [k - 1, r]\).

- **EV2.** When \((u_1, u_2) = (e_{j-1}, v_k)\) and \(A > K_{j,k}\), \((e_{j-1}, v_k)\) is G-dead for \(j' \in [j, t]\).

2. Throughout the “repeat” statement, \((u_1, u_2) \in U(r, s, t)\), and \((u_1, u_2) \neq (v_1, v_t)\) and \((u_1, u_2) \neq (e_{t-1}, v_1)\). Moreover, it terminates successfully at \((u_1, u_2) = (v_1, v_r)\).

This lemma implies (*). Instead, if we want to prove (**), we need to prove an enhanced statement of Lemma 17, which states that all unit pairs on the right are G-dead when \(u_1\) is to be killed, and all unit pairs on the bottom are G-dead when \(u_2\) is to be killed.
Lemma 17.2 can be proved the same as Lemma 8.3. The proofs of Lemma 8.1 and 8.2 can be simply borrowed to prove Claim VV1 and VV2. (In the previous proof, we only claim some dead vertex pairs, yet it is easy to see that they are also G-dead.) In the following we prove the other six claims of Lemma 17.1, by using some similar proving techniques.

**Proof of EE1, EE2.** We first prove EE1. Assume $A \leq H_{j,k}$, we shall prove that $(e_{j-1}, e_{k'})$ is G-dead for $k' \in [k, r - 1]$. Assume that $(e_{j-1}, e_{k-1})$ is legal and $e_{j-1} \parallel e_{k-1}$. Otherwise $(e_{j-1}, e_{k})$, $(e_{j-1}, e_{r-1})$ are all illegal and thus G-dead (due to Observation 15.2).

Now, we argue that $(e_{j-1}, e_k)$ is G-dead even if it is legal. See Figure 11 (a). Clearly, $Q(e_{j-1}, e_k)$ lies above the line $l$ at $H_{j,k}$ that is parallel to $\overrightarrow{P_{j,k}}$. However, since $A \leq H_{j,k}$ in the distance to $\overrightarrow{P_{j,k}}$, polygon $P$ lies below line $l$. Therefore, $Q(e_{j-1}, e_k)$ does not intersect $P$. Further applying Observation 15.1, $(e_{j-1}, e_k)$ must be G-dead. The same argument holds for any pair $(e_{j-1}, e_{k'})$ where $k' \in [k, r - 1]$, so EE1 holds.

We then prove EE2. Here $A > H_{j,k}$, so $H_{j,k} \neq \infty$. So $(e_{j-1}, e_{k-1})$ is legal and $e_{j-1} \parallel e_{k-1}$. See Figure 11 (b). Since $A > H_{j,k}$, clearly $A \geq J_{j,k}$ in the distance to $\overrightarrow{P_{j,k}}$. Applying Observation 15.2, $(e_{j-1}, e_{k-1})$ is G-dead. Moreover, since $A \geq J_{j,k}$ in the distance to $\overrightarrow{P_{j,k}}$, we can easily obtain that $A \geq J_{j+1,k}$ in the distance to $\overrightarrow{P_{j+1,k}}$. (The proof is trivial and is simply illustrated in Figure 11 (c).) According to $A \geq J_{j+1,k}$ and by using Observation 15.3, $(e_{j+1}, e_{k-1})$ is G-dead. By induction, $(e_{j'}, e_{k-1})$ is G-dead for $j' \in [j, t - 1]$, so EE2 holds. ▲

**Proof of VE1, VE2.** We first prove VE1. Assume $A < J_{j,k}$, we shall prove that $(v_j, v_{k'})$ is G-dead for $k' \in [k, r]$. Assume $(v_j, v_{k-1})$ is legal and $v_j \parallel v_{k-1}$. Otherwise, $(v_j, v_k), \ldots , (v_j, v_r)$ are all illegal and hence G-dead. We now argue that $(v_j, v_k)$ is G-dead even if it is legal. See Figure 11 (d). Let $l$ denote the parallel line of $\overrightarrow{P_{j,k}}$ at $J_{j,k}$. It separates $Q_{j,k}$ from $P$ when $A < J_{j,k}$. Therefore, $Q_{j,k} \cap P = \varnothing$, which implies that $(v_j, v_k)$ is G-dead. Similarly, every vertex pair $(v_j, v_{k'})$ where $k' \in [k, r]$ is G-dead. So VE1 holds. We then prove VE2. Here $A \geq J_{j,k}$, so $J_{j,k} \neq \infty$. So $(e_{j}, e_{k-1})$ is legal and $e_{j} \parallel e_{k-1}$. See Figure 11 (e). By the same inductive analysis as the EE case, $(e_{j'}, e_{k-1})$ is G-dead for $j' \in [j, t - 1]$, i.e. VE2 holds. ▲

![Figure 11](https://via.placeholder.com/150) Illustration of the proof of EE1, EE2, VE1, VE2, EV1, and EV2.
Proof of EV1, EV2. Similarly as before, we can assume that \((e_{j-1}, e_k)\) is legal and \(e_{j-1} \not\parallel e_k\).

When \(A \leq K_{j,k}\), we can see \((e_{j-1}, e_k)\) is G-dead because \(Q(e_{j-1}, e_k) \cap P = \emptyset\) (because \(Q(e_{j-1}, e_k)\) and \(P\) are separated by the line \(l\) at \(K_{j,k}\) that is parallel to \(\overrightarrow{vjvj}\)). See Figure 11(f). Moreover, for the same reason, \((e_{j-1}, e_k)\) is G-dead for \(k' \in [k, r - 1]\), i.e. EV1 holds.

Next, assume that \(A > K_{j,k}\). This implies implies that \((v_j, v_k)\) is G-dead by Observation 7.3. We then claim that \((v_{j+1}, e_k)\) is also G-dead. This reduces to prove that \(A > K_{j+1,k}\) in the distance to \(\overrightarrow{vjvj}\), which can be easily observed in the figure (the omitted proof is easy and similar as before). By induction, \((v_j, v_k)\) is G-dead for \(j' \in [j, t]\), i.e. EV2 holds. ▷

4.4 Final step for computing all generally 3-stable triangles

Let \([X \cup Y]\) denote the portion of \(\partial P\) starting from \(X\) and clockwise to \(Y\). We describe our first algorithm for computing the generally 3-stable triangles in Algorithm 4.

Notation used in Algorithm 4. Denote by \(A_{j,k}\) and \(A_{j,k}^*\) the vertex with the largest distance to \(\overrightarrow{vjvk}\) on the right of \(\overrightarrow{vjvk}\). When two vertices have the same distance to \(\overrightarrow{vjvk}\), we choose the clockwise first one to be \(A_{j,k}\) and choose the next one to be \(A_{j,k}^*\). (In most cases, \(A_{j,k} = A_{j,k}^*\).)

A unit pair is called visited if it will be visited when we call Algorithm 3 with parameters \((r, s, t)\) or \((s, t, r)\) or \((t, r, s)\). Observing Algorithm 3 we know that for any visited vertex pair \((v_j, v_k)\), \(A_{j,k}\) and \(A_{j,k}^*\) are easy to compute. Moreover, we claim that for any visited edge pair \((e_j, e_k)\), both \(A_{j,k}\) and \(A_{j+1,k+1}^*\) can be computed in amortized \(O(1)\) time. This is because all the visited unit pairs form a monotonic path in the table as discussed.

```plaintext
1 foreach visited vertex pair \((v_j, v_k)\) do
2     foreach unit \(u\) in \([A_{j,k} \cup A_{j,k}^*]\) do
3         FindGenarally3stable(\((u, v_j, v_k)\));
4     end
5 end
6 foreach visited edge pair \((e_j, e_k)\) do
7     foreach unit \(u\) in \([A_{j,k} \cup A_{j+1,k+1}^*]\) do
8         FindGenarally3stable(\((u, e_j, e_k)\));
9     end
10 end
```

Algorithm 4: Compute generally 3-stable triangles with \(\geq 2\) vertex-type corners

Note that we omit the details of the procedure \(FindGenarally3stable()\). It can be easily implemented in \(O(1)\) time, as partially discussed in Observation 10 and 11.

![Figure 12 Correctness, bad efficiency, and optimization by batch technique.](image)

2017
The correctness of Algorithm 4 is based on Lemma 18.

**Lemma 18.** Let \((e_j, e_k)\) be a visited edge pair. Assume that there is a generally 3-stable \(\triangle ABC\) such that \(B \in e_j, C \in e_k\). Then, point \(A\) lies in \([A_{j,k} \cap A'_{j+1,k+1}]\).

**Proof.** We prove it by contradiction. Consider an opposite case where \(A\) lies between \(v_{k+1}\) and \(A_{j,k}\), as shown in Figure 12 (a). Another opposite case where \(A\) lies between \(A^*_{j+1,k+1}\) and \(v_j\) is symmetric. Note that \(A < A_{j,k}\) in the distance to \(v_jv_k\). By comparing the slope of \(v_jv_k\) and \(BC\), we have \(A < A_{j,k}\) in the distance to \(BC\), thus \(A\) is not stable in \(\triangle ABC\).

The current algorithm is inefficiency and runs in \(\Omega(n^2)\) time in worst case. However, by using a batch technique shown in the next subparagraph, it can be optimized to \(O(n)\) time. Figure 12 (b) draws an example to illustrate that Algorithm 4 could cost \(\Omega(n^2)\) time. In this example, \((e_j, e_k), \ldots, (e_j, e_{k'})\) are \(\Omega(n)\) visited edge pairs. In addition, \(A_{j,k} = \ldots = A_{j,k'} = v_i\) and \(A^*_{j+1,k+1} = \ldots = A^*_{j+1,k'+1} = v_{i'}\), and that there are \(\Omega(n)\) units in \([v_i \cup v_{i'}]\).

**The batch technique.** An idea to optimize the algorithm is to restrict the enumerating units \((u)\) for \((e_j, e_k)\) and handle those enumerations for \((e_j, e_k), \ldots, (e_j, e_{k'})\) in a batch. To be more specific, make parallel lines of \(e_j\) at the endpoints of \(e_k\). This defines a stripe region (denoted by \(S_k\)) where \(u\) must intersect so that \((u, e_j, e_k)\) may accommodate a generally 3-stable triple (due to Observation 15). Notice that these stripe regions \(S_k_1, \ldots, S_k_n\) do not overlap, therefore, by handling \((e_j, e_k), \ldots, (e_j, e_{k'})\) together, we can enumerate the associated units for all of them in \(O(k' - k + \sum|A^*_{j+1,k'+1} - A_{j,k}| + 1)\) time, where \(|A^*_{j+1,k'+1} - A_{j,k}|\) denotes the number of vertices in \([A_{j,k} \cap A'_{j+1,k'+1}]\). Similarly, we handle the visited edge pairs in a column (for example, \((e_j, e_k), \ldots, (e_{j'}, e_{k'})\)) in a batch, in \(O(j' - j + \sum|A^*_{j'+1,k'+1} - A_{j,k}| + 1)\) time. The entire batch method is illustrated in Figure 13.

**Lemma 19.** The running time of the optimized Algorithm 4 is \(O(n)\). Moreover, our entire algorithm for computing all generally 3-stable triangles runs in \(O(n)\) time.

![Figure 13](image13.png)

**Summary and an alternative algorithm for the last step.** To enumerate the generally 3-stable triangles \(ABC\), we first settle the units containing \(B\) and \(C\) and then the unit containing \(A\). In the above, we apply an additional assumption that \(B, C\) are both at vertices or both in edges. Correspondingly, we enumerate all the edge pairs and vertex pairs that are not \(G\)-dead (by three calls of the generalized Rotate-and-Kill process). Alternatively, we may not apply this assumption but assume that the unit pair containing \(B, C\) is in \(U(r, s, t)\). Due to \((**\)) these unit pairs can be enumerated via one call of the generalized Rotate-and-Kill process - so the entire efficiency may be slightly better. However, in this way we need to consider one more case in finding \(A\), i.e. \(B, C\) is in an edge and a vertex.
4.5 Applications - compute the smallest triangle enclosing $P$

In this subsection, we assume $T = \triangle abc$ is a local minimum among triangles enclosing $P$. Denote the midpoint of the three sides of $T$ (i.e., $ab, bc, ca$) by $C, A, B$. We know that $A, B, C$ lie in $\partial P$ according to the following lemma (it is rediscovered in many places).

Lemma 20. \[10] The midpoint of each side of $T$ touches $P$.

Moreover, $A, B, C$ are clearly stable in $\triangle ABC$. Therefore, $\triangle ABC$ is (roughly) generally 3-stable. (Note: in degenerate cases as shown in Figure \[2] (a), $ABC$ itself may not be generally 3-stable, but there is a generally 3-stable triangle equivalent to $\triangle ABC$.)

It is easy to compute $a, b, c$ in $O(1)$ time from $A, B, C$. So finding the local minimums among triangles enclosing $P$ reduces to computing the generally 3-stable triangles in $P$.

Theorem 21. Given a convex polygon $P$ with $n$ vertices, we can compute all local minimums among its enclosing triangles in $O(n)$ time.

Acknowledgement

The author thank Professor Zhiyi Huang for discussion. We also thank the authors of \[15\] for giving us the opportunity to revisit such an interesting and classic problem.

References

1. A. Aggarwal, J.S. Chang, and C.K. Yap. Minimum area circumscribing polygons. The Visual Computer, 1(2):112–117, Aug 1985. doi:10.1007/BF01898354
2. A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. Algorithmica, 2(1-4):195–208, 1987.
3. A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum-weight k-link path in graphs with the concave monge property and applications. Discrete & Computational Geometry, 12(1):263–280, 1994.
4. B. Bhattacharya and A. Mukhopadhyay. On the Minimum Perimeter Triangle Enclosing a Convex Polygon, pages 84–96. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-540-44400-8_9
5. J. E. Boyce, D. P. Dobkin, R. L. (Scot) Drysdale, III, and L. J. Guibas. Finding extremal polygons. In 14th Symposium on Theory of Computing, pages 282–289, 1982.
6. P. Brab, G. Rote, and K. J. Swanepoel. Triangles of extremal area or perimeter in a finite planar point set. Discrete & Computational Geometry, 26:51–58, 2001. doi:10.1007/s00454-001-0010-6
7. S. Chandran and D. M. Mount. A parallel algorithm for enclosed and enclosing triangles. International Journal of Computational Geometry & Applications, 02(02):191–214, 1992. doi:10.1142/S0218195992000123
8. J. S. Chang and C. K. Yap. A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems. In 25th Annual Symposium on Foundations of Computer Science, pages 408–416, Oct 1984. doi:10.1109/SFCS.1984.715942
9. M. M. Day. Polygons circumscribed about closed convex curves. Transactions of the American Mathematical Society, 62(2):315–319, 1947.
10. D. P. Dobkin and L. Snyder. On a general method for maximizing and minimizing among certain geometric problems. In 20th Annual Symposium on Foundations of Computer Science, pages 9–17, Oct 1979. doi:10.1109/SFCS.1979.28
11. A. Dumitrescu, M. Sharir, and C.D. Tóth. Extremal problems on triangle areas in two and three dimensions. Journal of Combinatorial Theory, Series A, 116(7):1177 – 1198, 2009. doi:https://doi.org/10.1016/j.jcta.2009.03.008
23:18 Maximal Area Triangles

12. T. Hausel, E. Makai, and A. Szucs. Polyhedra inscribed and circumscribed to convex bodies.
13. K. Jin. Maximal parallelograms in convex polygons - a novel geometric structure. CoRR, abs/1512.03897, 2015. URL: http://arxiv.org/abs/1512.03897
14. K. Jin and K. Matulef. Finding the maximum area parallelogram in a convex polygon. In 23rd Proceeding of Canadian Conference on Computational Geometry, 2011.
15. V. Keikha, M. Löffler, J. Urhausen, and I. v. d. Hoog. Maximum-area triangle in a convex polygon, revisited. CoRR, abs/1705.11035, 2017.
16. V. Klee. Facet Centroids and Volume Minimization. 1986.
17. V. Klee and M. C. Laskowski. Finding the smallest triangles containing a given convex polygon. Journal of Algorithms, 6(3):359 – 375, 1985. doi:https://doi.org/10.1016/0196-6774(85)90005-7
18. M. Levi. Minimal perimeter triangles. The American Mathematical Monthly, 109(10):890–899, 2002.
19. E. A. Melissaratos and D. L. Souvaine. Shortest paths help solve geometric optimization problems in planar regions. SIAM Journal on Computing, 21(4):601–638, 1992. doi:10.1137/S0097539796301038
20. A. Miernowski, W. Mozgawa, and W. Rzymowski. Minimal area n-simplex circumscribing a strictly convex body in $r^n$. Journal of Geometry, 87(1):99–105, Nov 2007. doi:10.1007/s00022-006-1905-4
21. J. S.B. Mitchell and V. Polishchuk. Minimum-perimeter enclosures. Information Processing Letters, 107(3):120 – 124, 2008. doi:https://doi.org/10.1016/j.ipl.2008.02.007
22. J. O’Rourke, A. Aggarwal, S. Maddila, and M. Baldwin. An optimal algorithm for finding minimal enclosing triangles. Journal of Algorithms, 7(2):258 – 269, 1986. doi:http://dx.doi.org/10.1016/0196-6774(86)90007-6
23. O. Pârvu and D. Gilbert. Implementation of linear minimum area enclosing triangle algorithm. Computational and Applied Mathematics, 35(2):423–438, Jul 2016. doi:10.1007/s40314-014-0198-8
24. Arun K. Pujari and A. Nataraj. Linear algorithm to find the largest intriangles of a planar convex polygon. Kybernetes, 25(5):53–59, 1996. doi:10.1108/03684929610124122
25. B. Schieber. Computing a minimum-weight k-link path in graphs with the concave monge property. In Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’95, pages 405–411. Society for Industrial and Applied Mathematics, 1995.
26. G. Toussaint. Solving geometric problems with the rotating calipers. In In Proc. IEEE MELECON’83, pages 10–02, 1983.
27. F. Vivien and N. Wicker. Minimal enclosing parallelepiped in 3d. Computational Geometry, 29(3):177 – 190, 2004. doi:https://doi.org/10.1016/j.comgeo.2004.01.009
28. Y. Zhou and S. Suri. Algorithms for a minimum volume enclosing simplex in three dimensions. SIAM Journal on Computing, 31(5):1339–1357, 2002. doi:10.1137/S00975397993683992
A  Any two generally 3-stable triangles are interleaving

![Figure 14](image-url) Any two generally 3-stable triangles must be interleaving.

**Proof of Lemma 12.** Suppose \( \triangle A_1B_1C_1, \triangle A_2B_2C_2 \) are generally 3-stable and they do not interleave. There are four essentially-different cases: (as shown in Figure 14(a),(b),(c),(d))

Case a. \( A_2 \) is between \( A_1, B_1 \) meanwhile \( B_2, C_2 \) are between \( B_1, C_1 \).

Case b. \( A_2 \) is between \( A_1, B_1 \) meanwhile \( B_2 = B_1 \) meanwhile \( C_2 \) is between \( B_1, C_1 \).

Since \( C_1 \) is stable in \( \triangle A_1B_1C_1 \), it has the largest distance to \( \overrightarrow{A_1B_1} \) among all points of \( P \) on the right of \( A_1B_1 \). It implies \( C_1 > C_2 \) in the distance to \( \overrightarrow{A_2B_2} \), by comparing the slope of \( A_1B_1 \) and \( A_2B_2 \). Therefore, \( C_2 \) is not stable in \( \triangle A_1B_2C_2 \), which is contradictory.

Case c. \( A_2, C_2 \) are between \( A_1, B_1 \) meanwhile \( B_2 = B_1 \).

Case d. \( A_2, B_2, C_2 \) are all between \( A_1, B_1 \) and \( A_1, C_2, A_2, B_2, B_1 \) lie in clockwise order.

Since \( A_1 \) is stable in \( \triangle A_1B_1C_1 \), it has the largest distance to \( \overrightarrow{B_1C_1} \) among all points of \( P \) on the right of \( B_1C_1 \). It implies \( A_1 > C_2 \) in the distance to \( \overrightarrow{A_2B_2} \), by comparing the slope of \( B_1C_1 \) and \( A_2B_2 \). Therefore, \( C_2 \) is not stable in \( \triangle A_2B_2C_2 \), which is contradictory.

Note: here, the vertices between \( A,B \) means the ones in set \( \{A+1, \ldots, B-1\} \).

B  A sufficient and necessary condition for \( \triangle ABC \) to be LMAT

Recall that 3-stable is a necessary condition for a triangle to be LMAT. As we claimed, this is condition not sufficient. We state the full condition in the following (proof omitted).

**Lemma 22.** Let \( d_{X,Y}(Z) \) denote the distance from \( Z \) to \( \overrightarrow{XY} \). Given a 3-stable triangle \( \triangle ABC \). It is an LMAT if and only if none of the following holds.

1. \( (d_{A,C}(B) = d_{A,C}(B-1) \) and \( d_{A,B}(C) = d_{A,B}(C-1) \) or \( d_{A,B}(B) = d_{A,B}(B-1) \) and \( d_{A,B}(C) = d_{A,B}(C+1) \));
2. \( (d_{B,A}(C) = d_{B,A}(C-1) \) and \( d_{B,C}(A) = d_{B,C}(A-1) \) or \( d_{B,A}(C) = d_{B,A}(C+1) \) and \( d_{B,C}(A) = d_{B,C}(A+1) \));
3. \( (d_{C,B}(A) = d_{C,B}(A-1) \) and \( d_{C,A}(B) = d_{C,A}(B-1) \) or \( d_{C,B}(A) = d_{C,B}(A+1) \) and \( d_{C,A}(B) = d_{C,A}(B+1) \));

![Figure 15](image-url) A case where \( \triangle ABC \) is 3-stable but not an LMAT. In this case, we can find another triangle \( AB'C' \) with a area larger than that of \( ABC \) such that \( |B'B| = |C'C| < \varepsilon \)
C  More omitted proofs

Proof of Lemma 16 Since \((u_1, u_2)\) is not G-dead, there exists a generally 3-stable \(\triangle ABC\) such that \(B \in u_1\) and \(C \in u_2\). Notice that \(\triangle v_r v_s v_t\) is 3-stable, and so generally 3-stable. By Lemma 12 \(\triangle ABC\) interleaves \(\triangle v_r v_s v_t\). Thus we have three possibilities regarding the positions of \(B, C\). \(B \in [v_s \cup v_t]\) and \(C \in [v_t \cup v_s]\), or \(B \in [v_t \cup v_r]\) and \(C \in [v_s \cup v_r]\). Correspondingly, \((u_1, u_2)\) belongs to \(U(r, s, t), U(s, t, r)\), or \(U(t, r, s)\). Here, \([X \cup Y]\) indicates the portion of \(\partial P\) from \(X\) and clockwise to \(Y\).

D  An example for the generalized Rotate-and-Kill process

![Figure 16 Example of the generalized Rotate-and-Kill process.](image-url)