Algorithmica (1990) 5: 561-571

Algorithmica


# Searching for Empty Convex Polygons[1]

David P. Dobkin,[2] Herbert Edelsbrunner,[3] and Mark H. Overmars[4]

**Abstract.** A key problem in computational geometry is the identification of subsets of a point set having particular properties. We study this problem for the properties of convexity and emptiness. We show that finding empty triangles is related to the problem of determining pairs of vertices that see each other in a star-shaped polygon. A linear-time algorithm for this problem which is of independent interest yields an optimal algorithm for finding all empty triangles. This result is then extended to an algorithm for finding empty convex $r$-gons ($r > 3$) and for determining a largest empty convex subset. Finally, extensions to higher dimensions are mentioned.

**Key Words.** Computational geometry, Empty convex subsets, Analysis of algorithms, Combinatorial geometry.

**1. Introduction.** A fundamental problem in geometric complexity concerns the counting and reporting of objects from a collection that have certain desirable properties. We consider here instances of this problem involving subsets of a finite set of points in Euclidean space that form the vertex sets of convex polytopes which are empty, that is, the polytopes contain no other points of the set in their interiors. We refer to such subsets as *empty convex subsets*. If the point set $S$ is given in the Euclidean plane, then we call an empty convex subset of size $r$ an *empty convex $r$-gon*. We are specifically interested in the following problem:

> Let $S$ be a set of $n$ points in general position in the plane, that is, no three points are collinear. For a given positive integer $r$, $3 \leq r \leq n$, find all empty convex $r$-gons of $S$.

A solution of this problem is called an enumeration of the set $\Gamma_r(S)$. The cardinality of this set is represented by $\gamma_r(S)$. The reader should note that the naive algorithm consists of enumerating all subsets of $r$ points, determining whether they all lie on their convex hull and finally determining whether any other point is interior to this convex hull. This algorithm runs in time $O(n^{r+1} \log r)$

[1] The first author is pleased to acknowledge support by the National Science Foundation under Grant CCR-8700917. The research of the second author was supported by Amoco Foundation Faculty Development Grant CS 1-6-44862 and by the National Science Foundation under Grant CCR-8714565.
[2] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA.
[3] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA.
[4] Department of Computer Science, University of Utrecht, P.O. Box 80012 NL-3508 TA Utrecht, The Netherlands.

Received March 20, 1988; revised March 23, 1989. Communicated by C. K. Wong.

and can have output size as large as $\binom{n}{r}$ or as small as 0 as $n$ and $r$ vary. This range of possible values is an interesting problem and is briefly addressed below.

We present a solution with running time proportional to $\gamma_r(S)$ when $r=3,4$ and $\gamma_3(S)+r\gamma_r(S)$ when $r\geq 5$. As an intermediate result we get an algorithm that determines a largest empty convex subset in time proportional to $\gamma_3(S)$, which compares favorably with the $O(n^3)$-time algorithms given in [1] and [4]. Indeed, Bárányi and Füredi [2] show that the expected value of $\gamma_3(S)$ is quadratic in the size of $S$ if the points are chosen uniformly in the unit-square. The amount of space of the algorithms in [1] and [4] is $O(n^2)$ and $O(n)$, respectively. The space needed by our algorithm is somewhere in between, that is, proportional to the maximum number of empty triangles with common leftmost vertex.

Our algorithms are based on a result for computing the visibility of vertices of a star-shaped polygon. Hershberger [8] has previously given a linear-time algorithm for this problem—his algorithm is more general in that it also works for simple polygons that are not star-shaped. We present a different approach that leads to a considerably simpler (although highly recursive) algorithm which is readily applicable to our more general problem. In all of our algorithms, we assume that the points lie in general position. If this is not the case, it is possible to find modifications of our algorithms with the same running times.

In the next section we give an overview of the algorithm to come. The following three sections then fill in the details with Section 3 describing the visibility algorithm, Section 4 showing how to find the longest convex chain, and Section 5 dealing with reporting empty convex $r$-gons. Section 6 summarizes the results and comments upon the range of values possible for $\gamma_r(S)$ as $S$ ranges over all point sets of size $n$. The final section describes extensions to higher dimensions.

The problem of finding empty convex $r$-gons has a long history. Erdös [6] asked whether there was a value $f(r)$ such that all sets of at least $f(r)$ points in general position in the plane determine an empty convex $r$-gon. It was shown that $f(3)=3$, $f(4)=5$, and $f(5)=10$ by Harborth [7]. Horton [9] has shown that $f(r)$ is infinite for $r>6$. The value of $f(6)$ remains open, although Overmars *et al.* [11] detected a set of 26 points without empty convex hexagon using an incremental version of the algorithm described in this paper. This implies $f(6)\geq 27$.

**2. The Basic Algorithm.** Let $S$ be the set of $n$ points in the plane for which we want to find all subsets of $r$ points that form a convex $r$-gon that is empty, i.e., does not contain any other point in the set. We assume that the points in $S$ lie in general position and that no two points lie on a common vertical line.

To find all empty convex $r$-gons we locate for each point $p$ all empty convex $r$-gons that have $p$ as leftmost vertex. In this way, each empty convex $r$-gon is reported exactly once.

Globally, the algorithm works as follows:

1. For each $p\in S$, sort all other points by angle around $p$, resulting in an ordered

set $S_p$. From $S_p$ remove all the points to the left of $p$ and add $p$ instead. This results in a star-shaped polygon $P_p$. The *kernel* of $P_p$ is the set of all points from which every edge of $P_p$ is visible; obviously, $p$ belongs to the kernel of $P_p$.

2. For each $p \in S$, compute the visibility graph $VG_p$ inside $P_p$, including the edges of $P_p$, not including the visibility edges involving $p$.
3. For each $p \in S$, compute all convex chains in $VG_p$ of $r-2$ edges. Each of these forms, together with $p$, an empty convex $r$-gon.

The correctness of the method follows from the following observations:

1. Any convex empty $r$-gon has a unique leftmost point $p$. This follows from the fact that no two points lie on a vertical line.
2. Whether $r$ points form an empty convex $r$-gon with $p$ as leftmost vertex is independent of the points to the left of $p$. Hence, these points can be discarded (as happens in step 1).
3. Any convex empty $r$-gon with $p$ as leftmost vertex must lie inside $P_p$ and has edges of $VG_p$ as edges.
4. Point $p$ can see any vertex of $P_p$ (it lies in the kernel). Hence, if $p_1, \ldots, p_{r-1}$ form a chain in $VG_p$, the $r$-gon $p, p_1, \ldots, p_{r-1}$ is empty.
5. Point $p$ lies to the left of the other points and the other points are sorted by angle about $p$. Hence, if $p_1, \ldots, p_{r-1}$ is a convex chain, the $r$-gon $p, p_1, \ldots, p_{r-1}$ is convex.

We now describe the three steps of the algorithm in more detail. Step 1 asks for each point $p$ of $S$ to sort the other points around it. Using standard sorting methods this can be done in time $O(n^2 \log n)$. Using the results of [3] and [5] it is possible to do the sorting around all the points simultaneously in time $O(n^2)$. Removing the points to the left of $p$ and forming the polygons $P_p$ can easily be done in time $O(n^2)$.

Step 2 of the algorithm asks for computing the visibility graph inside a polygon $P$. There is an algorithm for this problem which runs in time linear in the output size [8]. In our case the polygon $P$ has a particular shape. First, it is star-shaped and, secondly, one of its vertices lies inside the kernel. For this input there is a simpler algorithm with the same asymptotic running time. This algorithm is presented in the next section.

Step 3 is split into two steps. In the first step, described in Section 4, we determine the longest convex chain in the visibility graph. In fact, we determine for each edge of the visibility graph the longest convex chain that starts there (and goes counterclockwise). In the second step we use this information to determine all the chains of length $r-2$. Both parts run in time proportional to the size of the visibility graph.

**3. The Visibility Graph of a Star-Shaped Polygon.** We are now given a star-shaped polygon $P$ of $N$ vertices with one vertex $p$ that lies in the kernel. We are interested in obtaining the visibility graph inside $P$, denoted as $VG$. For a pair of vertices of $P$ we say that they are *visible within* $P$ if the line segment joining

them lies entirely in $P$ (including its boundary). Note that, because of our assumption that the points lie in general position, the line segment will either intersect the boundary of $P$ in its two endpoints or is an edge of $P$. The *visibility graph* inside $P$ consists of all pairs of vertices of $P$ that are visible within $P$.

The vertices of $P$ are ordered counterclockwise around $p$, numbered $p_1, \ldots, p_{N-1}$. Because of the requirements we have later we compute the visibility graph as a directed graph in which edges run from lower-indexed to higher-indexed vertices. (Moreover, we do not include the visibility edges involving $p$.) The edge (in the visibility graph) between $p_i$ and $p_j$ $(i<j)$ is denoted by $\overline{ij}$.

We construct the visibility graph $VG$ during one counterclockwise scan around the polygon. When we visit $p_i$ we construct all incoming edges of $p_i$. With each vertex $p_\ell$, $\ell \leq i$, we maintain a queue $Q_\ell$ that stores the starting points of some of the incoming edges of $p_\ell$ in counterclockwise order. It contains those points $p_j$ such that $\overline{j\ell}$ is an edge of the visibility graph and we have not yet reached another point $p_k$ with $k>\ell$ such that $\overline{jk}$ is an edge of the visibility graph. Hence, $Q_\ell$ is a kind of waiting list. It contains those points that can be seen by $p_\ell$ but could not be seen since, because $p_\ell$ blocks their view. The required operations that can be performed in constant time are the following:

1. ADD($\overline{ij}$): it creates an edge from $i$ to $j$. This edge is stored at both $p_i$ and $p_j$ for later use.
2. TURN($\overline{ij}$, $\overline{jk}$): it returns *left* or *right* depending on whether $p_k$ lies to the left or to the right of the directed line passing through $p_i$ and $p_j$ in this order. (Note that it cannot lie on the line.)
3. FRONT($Q$): it returns the index of the first point in queue $Q$.
4. DEQUEUE($Q$): it removes the first point from queue $Q$.
5. ENQUEUE($k$, $Q$): it adds the point $p_k$ to queue $Q$.

The algorithm now looks as follows:

```
procedure VISIBILITY;
  for i := 1 to N−1 do Q_i := ∅ end;
  for i := 1 to N−2 do PROCEED(i, i+1) end.

procedure PROCEED(i, j);
  while Q_i ≠ ∅ and TURN(FRONT(Q_i)i, ij) = left do
    PROCEED(FRONT(Q_i), j);
    DEQUEUE(Q_i);
  end;
  ADD(ij);
  ENQUEUE(i, Q_j).
```

PROCEED adds an edge from point $p_i$ to $p_j$. It also checks whether any of the points in the waiting queue of $p_i$ are visible from $p_j$ and, if so, recursively calls PROCEED. Because the points in the queue are sorted counterclockwise only a first portion of the queue needs to be checked.

Note that the new edge is added after all the recursive calls. This guarantees that the points in the queues are indeed sorted counterclockwise. Also, for each node we collect the incoming and outgoing edges sorted counterclockwise.

The correctness of the method follows from the following lemma:

LEMMA 1. *Let $j > i$. $\overline{ij} \in VG$ if and only if $j = i+1$ or there is a vertex $p_k$, $i < k < j$, such that $\overline{ik} \in VG$, $\overline{kj} \in VG$, and $TURN(\overline{ik}, \overline{kj}) = left$.*

PROOF. First note that because $p$ lies inside the kernel, $\overline{ij} \in VG$ if and only if triangle $pp_ip_j$ is empty. Now the proof goes as follows:

only if: If $j \neq i+1$ take the point $p_k$ between $p_i$ and $p_j$ that lies nearest to the line $\overline{ij}$. As the triangle $pp_ip_j$ is empty, TURN$(\overline{ik}, \overline{kj})$ = left. Moreover, obviously $\overline{ik} \in VG$ and $\overline{kj} \in VG$.

if: If $j = i+1$ $\overline{ij}$ clearly is in $VG$. If $j > i+1$ both the triangles $pp_ip_k$ and $pp_kp_j$ are empty. Moreover, $p_k$ lies beyond $\overline{ij}$ since $p_i$, $p_k$, $p_j$ form a left turn. Hence, the triangle $pp_ip_j$ must be empty and therefore $\overline{ij} \in VG$. □

LEMMA 2. *Finding the visibility graph takes time $O(|VG|)$.*

PROOF. This follows immediately from the fact that with every call of PROCEED an edge is added to the visibility graph. □

**4. Finding the Longest Convex Chain.** Given the visibility graph as a directed graph in which edges run from lower-indexed vertices to higher-indexed vertices (as produced by the algorithm described above), we now determine a longest convex chain. This is equivalent to finding a largest empty convex subset with a fixed leftmost vertex. In fact, we determine for each edge $e$ of the visibility graph the length, $L_e$, of the longest convex chain starting with $e$ going counterclockwise.

To this end we treat the vertices clockwise, starting at the highest-indexed vertex. We take care that after treating some vertex $p_i$ all incoming edges of $p_i$ have their $L$ field set to the right value.

The method works as follows: Assume we are at some vertex $p$. Let the incoming edges of $p$ be $i_1, \ldots, i_{imax}$ and the outgoing edges $o_1, \ldots, o_{omax}$ both ordered counterclockwise by angle. Note that the algorithm for computing the visibility graph inside $P$ gives us the edges in this order. For all outgoing edges we know the length of the longest convex chain that starts there.

We treat the incoming edges in the reversed order, starting at $i_{imax}$. For this first incoming edge we look at all outgoing edges that form a convex angle with it. Let these edges be $o_l, \ldots, o_{omax}$. If they do not exist, we set $L_{i_{imax}}$ to 1. Otherwise, let $m$ be the maximal value of the $L$ fields of them. Then $L_{i_{imax}} = m+1$. Clearly, all outgoing edges that form a convex angle with $i_j$ form a convex angle with $i_{j-1}$. But we do not have to check these outgoing edges again. We already know that $m$ is the maximal length among them. Hence, for the next incoming edge we know that the length of the chain is either $m+1$ or there is an outgoing edge with index smaller than $l$ with larger $L$ field. Hence, starting at $l-1$ we look at

preceding edges that form a convex angle with the incoming edge. Thus $l$ will become the new minimal index and $m$ the new maximal $L$ field (if any). In this way we continue.

The following procedures state the algorithm precisely:

```
procedure MAXCHAIN;
  for i := N − 1 downto 1 do TREAT(p_i) end.

procedure TREAT(p);
  Let i_1, ..., i_imax be the incoming edges of p, and let
  o_1, ..., o_omax be the outgoing edges of p, both ordered counterclockwise.
  l := omax; m := 0;
  for j := imax downto 1 do
    L_{i_j} := m + 1;
    while l > 0 and TURN(i_j, o_l) = left do
      if L_{o_l} > m then
        m := L_{o_l};
        L_{i_j} := m + 1
      end;
      l := l − 1
    end
  end.
```

The correctness of the method easily follows from the above discussion. (Note that in the algorithm the function of $l$ is slightly different than described. Rather that being the minimal index that does form a convex angle it is the first one that does not form a convex angle and, hence, the first one that has to be checked with the next incoming edge.)

LEMMA 3. *Finding the maximal chain and filling in the L fields takes time* $O(|VG|)$.

PROOF. For each vertex $p$ we look at every incoming edge and every outgoing edge once. So in total we look at each edge twice. As the size of the visibility graph is larger or equal to the number of vertices of the polygon, the bound follows.

**5. Reporting the Empty Convex $r$-Gons.** We now have, for each edge in the visibility graph, the length of the longest convex chain starting there. We now use this information to determine all the chains of some given length $r-2$ (resulting in empty convex polygons of $r$ vertices). We do so during one scan of the vertices in counterclockwise order.

For each edge $e$ we maintain a set $C_e$ of all chains of length less than $r-2$ ending on $e$ for which we know (using the $L$ field) that they can be extended to a chain of length $r-2$. A chain is stored as a sequence of points. Moreover, with the chain its length is also stored. We use the following operations on chains. The first three require constant time and the fourth takes time linear in its output size.

1. LENGTH($ch$): returns the length of chain $ch$.
2. EXTEND($ch, e$): returns chain $ch$ extended with edge $e$.
3. CREATE($e$): creates a chain of length 1 starting with edge $e$.
4. REPORT($ch$): reports chain $ch$ as an answer.

To be able to form and extend chains in an efficient way, for each point $p_i$ we sort the outgoing edges by decreasing the $L$ field. As we know that each $L$ value lies between 1 and $N-2$ we can do this during one global radix sort in time $O(|VG|)$. For a point $p$ let $S'_o = o'_1, \ldots, o'_{omax}$ be this sorted list of outgoing edges. As before $S_o = o_1, \ldots, o_{omax}$ is the list of outgoing edges sorted counterclockwise. We assume that we have pointers from the elements in $S_o$ to the elements in $S'_o$ such that given a point in $S_o$ we can remove it in time $O(1)$ from $S'_o$. The algorithm looks as follows:

```
procedure CHAINS;
  for i := 1 to N-2 do TREAT(p_i) end.

procedure TREAT(p);
  Let S_i = i_1, ..., i_imax be the incoming edges of p.
  Let S_o = o_1, ..., o_omax be the outgoing edges of p.
  Let S'_o = o'_1, ..., o'_omax be the outgoing edges of p sorted by L.
  for j := 1 to omax do
    if L_oj ≥ r-2 then C_oj := {CREATE(o_j)} else C_oj := ∅ end
  end;
  m := 1 { m is the index so that S'_o contains edges o_m, ..., o_omax (these
           edges are also denoted as o'_1, ..., o'_m') };
  m' := omax { m' keeps track of the size of S'_o, in fact,
                m' = omax - m + 1 };
  for j := 1 to imax do
    while m ≤ omax and TURN(i_j, o_m) = right do
      Delete o_m from S'_o;
      m' := m' - 1; m := m + 1
    end;
    for each ch ∈ C_ij do
      t := 1; l := LENGTH(ch)
      while t ≤ m' and L_o't ≥ r-2-l do
        ch' := EXTEND(ch, o'_t);
        if l = r-3 then REPORT(ch') else C_o't := C_o't ∪ {ch'} end;
        t := t+1
      end
    end
  end.
```

The routine TREAT creates the sets of chains for all outgoing edges of the point $p$. All incoming edges will have their sets of chains ready. As a first step for each outgoing edge with an $L$-field greater or equal to $r-2$ we create a chain consisting only of the edge. We know for sure that this chain can be extended

to a chain of length $r-2$. Next we extend chains on incoming edges by moving them to the appropriate outgoing edges. This is done by first removing all the outgoing edges that do not form a convex angle with the current incoming edge. (Because of the order in which we treat the incoming edges they also do not form a convex angle with the other incoming edges.) Now we know that all outgoing edges form a convex angle with the current incoming edge. For each chain *ch* on this incoming edge we extend it with all outgoing edges it can be extended with (note that there is at least one such outgoing edge). To this end we treat the outgoing edges by decreasing length. As long as the $L$-field of the outgoing edge is at least $r-2-\text{LENGTH}(ch)$ we can extend the chain *ch* with it. If it gets length $r-2$ we report it, otherwise it is stored on the outgoing edge. An extended chain is represented by its last edge and a pointer to the old chain. We can thus extend a chain in constant time without destroying the old chain. In effect, this is an implicit representation of all chains in trees.

LEMMA 4. *Reporting the chains of length r takes time and space* $O(|VG|+rk)$ *where k is the number of reported chains.*

PROOF. For each point $p$ the following operations are performed: (i) Initializing the sets of chains for each outgoing edge. This obviously takes time $O(|VG|)$ in total. (ii) For each incoming edge we remove some outgoing edges. As each outgoing edge is removed at most once, this takes time $O(|VG|)$ in total. (iii) For each chain on an incoming edge we try to find edges with which it can be extended. We know that there must be at least one such edge. Per chain we spend an amount of time proportional to the number of extending edges found. Hence, in total, this will add up to $O(rk)$ time. □

**6. The Result in the Plane.** Before stating the implications of the previous sections, we state a few results on the behavior of the $\gamma_r(S)$. The following results on the minimum of $\gamma_r(S)$, denoted by $g_r(n)$, over all sets $S$ of $n$ points in the plane (assuming that no three points are collinear) are essentially due Horton [9] and to Bárányi and Füredi [2]. There is a positive constant $c$ such that

$$n^2/2+cn \le g_3(n) \le 2n^2, \qquad n^2/4+cn \le g_4(n) \le 3n^2,$$
$$n/6+c \le g_5(n) \le 2n^2, \quad \text{and} \quad g_6(n) \le n^2/2.$$

Furthermore, Bárányi and Füredi [2] prove that the expected number of empty triangles is $O(n^2)$ if the points are uniformly distributed in the unit square. To present our time bounds we also need the following result which is a lower bound on $\gamma_4(S)$ in terms of $\gamma_3(S)$:

$$\gamma_4(S) \ge \gamma_3(S) - \binom{n-1}{2}.$$

This inequality can be seen if we consider the visibility graph of a star-shaped polygon $P$ as constructed in Section 3. Let $p$ be the leftmost vertex of $P$. For every visibility edge of $P$ which is not at the same time a boundary edge we have at least one empty convex quadrilateral with this visibility edge as a diagonal. The visibility edge is also an edge of the empty triangle whose third vertex is $p$. Thus, for every empty triangle (except for those defined by $p$ and a boundary edge of $P$) we have at least one empty convex quadrilateral and no quadrilateral is counted twice. The result follows since the star-shaped polygons have a total of $\binom{n-1}{2}$ boundary edges.

Combining the results from the previous sections and applying these combinatorial results yields:

THEOREM 5. *Given a set $S$ of $n$ points in the plane, in general position, all subsets of $r$ points that form an empty convex $r$-gon can be determined in time and space $O(\gamma_3(S)+r\gamma_r(S))$. For $r=3, 4$, this simplifies to $O(\gamma_r(S))$.*

PROOF. This follows from the preceding sections, noting the following: The sorting takes time $O(n^2)$ and as $\gamma_3(S)=\Omega(n^2)$ we do not have to include it in the bound. Any edge in one of the visibility graphs computed corresponds to a unique empty triangle. Hence, the total sum of the number of edges of the visibility graphs is equal to $\gamma_3(S)$. The second statement follows since $\gamma_4(S)$ is at least proportional to $\gamma_3(S)$. □

Using the algorithms of Sections 3 and 4 we have also established the following result:

THEOREM 6. *Given a set $S$ of $n$ points in the plane, it is possible to determine a largest (in terms of number of sides) empty convex polygon in time and space $O(\gamma_3(S))$.*

Since the expected size of $\gamma_3(S)$ is $O(n^2)$ we have an algorithm that runs in expected quadratic time, assuming uniform distribution in the unit-square.

**7. Extensions to Higher Dimensions.** The construction of empty simplices in $d$ dimensions (that is, empty convex subsets of size $d+1$) can be reduced to maximal (or minimal) vector computation. Here a vector $a=(\alpha_1, \alpha_2, \ldots, \alpha_d)\in V$ is a *minimum* if there is no other vector $b=(\beta_1, \beta_2, \ldots, \beta_d)\in V$ with $\beta_i \le \alpha_i$ for $1 \le i \le d$. The reduction can be done as follows. Take any $d$ points $p_1, p_2, \ldots, p_d$, construct their convex hull $s_{d-1}$ which is a $(d-1)$-dimensional simplex, and consider all points $p$ on one side of the hyperplane spanned by $s_{d-1}$. We can represent each such point $p$ by the vector whose components are the $d$ angles defined by $s_{d-1}$ and the $d$ facets of the simplicial pyramid with base $s_{d-1}$ and apex $p$. (In three dimensions these angles are the three dihedral angles at the edges of the base triangle.) Clearly, $\{p, p_1, p_2, \ldots, p_d\}$ is empty if and only if there is no point $q$ whose $d$ angles are componentwise smaller than $p$'s angles.

Thus, the points that together with $p_1, p_2, \ldots, p_d$ form empty convex simplices correspond to the minimal angle vectors. Finding all minima of a set of $n$ vectors in $d$ dimensions can be done in $O(n \log n + n \log^{d-2} n)$ time (see [10]). Since we do the minimal vector computation for every $d$ points in the set we get an $O(n^{d+1} \log^{d-2} n)$-time algorithm which improves the trivial $O(n^{d+2})$-time method for finding all empty simplices.

The minima finding step can be used to construct convex subsets of sizes larger than $d+1$ incrementally. This can be done as follows. Consider the convex hull of an empty convex subset of size $r-1 \geq d+1$. This is a convex polytope with $r-1$ vertices and no points of the set inside. If we assume that no $d+1$ points lie on a common hyperplane, then all facets of the polytope are $(d-1)$-dimensional simplices. Using minimal vector computation we can now try to erect a simplicial pyramid at any such facet—in order to maintain convexity we can allow only points on top of such a facet whose angles are sufficiently small as determined by the neighboring facets. Note that the newly added point is connected to $d$ other points.

This approach has the disadvantage that not all convex subsets can be constructed this way. In three dimensions, the smallest counterexample is the octahedron which has six vertices and is simplicial (as required by assumption) but has no vertex of degree 3. In four dimensions the cyclic polytope with six vertices (one more vertex than the simplex) is a counterexample since every vertex has degree equal to 5.

Still, the idea of extending the convex set by finding minimal vectors should not be abandoned yet. In three dimensions, every convex polytope has at least one vertex whose degree is at most 5. This implies that every convex subset can be constructed by raising pyramids on top of triangles, adjacent pairs of triangles, and chains of three triangles (see Figure 1). More specifically, for a given convex subset of size $r-1$ we construct the convex hull and do the following:

1. For every triangle we solve a three-dimensional minimal vector problem.
2. For every pair of adjacent triangles we solve a minimal vector problem which is four-dimensional since every point above two triangles is represented by the vector of the four dihedral angles it defines at the edges where the two triangles do not touch.
3. For every triplet of triangles such that the second is adjacent to the first and the third triangle but the first and the third are not adjacent, we solve a five-dimensional minimal vector problem.

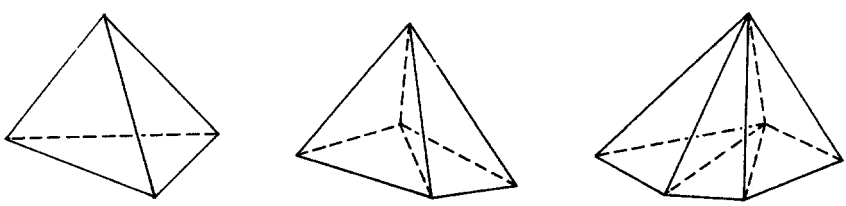

**Fig. 1.** A three-, four-, and five-sided pyramid raised on top of one, two, and three triangles.

The number of triangular facets, pairs of adjacent triangles, and triangle chains of length three is linear in $r-1$. For constant $r$, this process yields all empty convex subsets of size $r$ (each one at most $r!$ times) with $O(n \log^3 n)$ time per set.

The difficulty in extending this approach even to four dimensions is that cyclic polytopes with $r-1$ vertices have minimum vertex degree $r-2$. Thus, the dimensionality of the minimal vector problems are not bounded by any constant independent of $r$. We conclude this section with an open problem. Is there a polynomial-time algorithm for finding a largest empty convex set of $n$ points in three dimensions?

## References


[1] Avis, D., and Rappaport, D., Computing the largest empty convex subset of a set of points, *Proc. 1st Ann. ACM Sympos. Comput. Geom.*, 1985, pp. 161–167.

[2] Bárányi, I., and Füredi, Z., Empty simplices in Euclidean space, *Canad. Math. Bull.*, **30** (1987), 436–445.

[3] Chazelle, B., Guibas, L., and Lee, D., The power of geometric duality, *BIT*, **25** (1985), 76–90.

[4] Edelsbrunner, H., and Guibas, L., Topologically sweeping an arrangement, *J. Comput. System Sci.*, **38** (1989), 165–194.

[5] Edelsbrunner, H., O'Rourke, J., and Seidel, R., Constructing arrangements of lines and hyperplanes with applications, *SIAM J. Comput.*, **15** (1986), 341–363.

[6] Erdös, P., Combinatorial problems in geometry and number theory, *Proc. Sympos. Pure Math.*, **34** (1979), 149–162.

[7] Harborth, H., Konvex Fünfecke in ebenen Punktmengen, *Elem. Math.*, **33** (1978), 116–118.

[8] Hershberger, J., Finding the visibility graph of a simple polygon in time proportional to its size, *Proc. 3rd Ann. ACM Sympos. Comput. Geom.*, 1987, pp. 11–20.

[9] Horton, J. D., Sets with no empty convex 7-gons, *Canad. Math. Bull.*, **26** (1983), 482–484.

[10] Kung, H. T., Luccio, F., and Preparata, F. P., On finding the maxima of a set of vectors, *J. Assoc. Comput. Mach.*, **22** (1975), 469–476.

[11] Overmars, M. H., Scholten, B., and Vincent, I., Sets without empty convex 6-gons, *Bull EATCS*, **37** (1989), 160–168.