ELSEVIER

Computational Geometry 6 (1996) 277–290

Computational Geometry
Theory and Applications

Generating random polygons with given vertices

Chong Zhu [a], Gopalakrishnan Sundaram [b], Jack Snoeyink [c,*,1], Joseph S.B. Mitchell [d,2]

[a] *MacDonald Dettwiler and Associates, Richmond, BC, Canada*
[b] *ESRI, Redlands, CA, USA*
[c] *Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1W5, Canada*
[d] *Department of Applied Mathematics and Statistics, SUNY Stony Brook, NY, USA*

Communicated by Mark Keil; accepted 14 September 1995

Abstract

The problem of generating "random" geometric objects is motivated by the need to generate test instances for geometric algorithms. We examine the specific problem of generating a random x-monotone polygon on a given set of n vertices. Here, "random" is taken to mean that we select uniformly at random a polygon, from among all those x-monotone polygons having the given n vertices. We give an algorithm that generates a random monotone polygon in $O(n)$ time and space after $O(K)$ preprocessing time, where $n < K < n^2$ is the number of edges of the visibility graph of the x-monotone chain of the given vertex set. We also give an $O(n^3)$ time algorithm for generating a random convex polygon whose vertices are a subset of a given set of n points. Finally, we discuss some further extensions, as well as the challenging open problem of generating random simple polygons.

"Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin." — *John von Neumann*

1. Introduction

In addition to being of theoretical interest, the generation of random geometric objects has applications that include the testing and verification of time complexity for computational geometry algorithms. In order to have some control over the characteristics of the output, we would like to fix the vertex set and then generate uniformly at random a simple polygon with the chosen vertices.

This paper details some results of our study of generating random simple polygons. In particular, we describe an algorithm for generating, uniformly at random, x-monotone polygons on a given set of n vertices. We also discuss the problem of generating random convex polygons whose vertices

* Corresponding author.

1 Supported in part by an NSERC Research Grant and a fellowship from the British Columbia Advanced Systems Institute.
2 Supported in part by grants from Boeing Computer Services and the National Science Foundation (NSF Grants ECSE-8857642 and CCR-9204585).

SSDI 0925-7721(95)00031-3

are a subset of a given set of vertices and the problem of generating nested monotone polygons. We conclude with a brief discussion of generating random simple polygons, a problem that remains open.

Others have considered generating random simple polygons by various processes that move vertices (e.g., [11]). When vertices are fixed, there are a couple of related works: Epstein and Sack [4] gave an $O(n^4)$ algorithm to generate a triangulation of a given simple polygon at random. Meijer and Rappaport [8] studied monotone traveling salesmen tours and show that the number of x-monotone polygons on n vertices is between $(2+\sqrt{5})^{(n-3)/2}$ and $(\sqrt{5})^{(n-2)}$.

2. Counting and random generation

In this section, we establish an easy connection between the problem of *counting* the number of ways to complete a sequence (polygon) and the problem of randomly *generating* an instance of a sequence (polygon). We conclude that whatever one can count, one can generate uniformly at random.

Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements our goal is to generate an ordered *sequence*, $\sigma = (\sigma_1, \ldots, \sigma_k)$ with $\sigma_i \in S$, for $1 \leqslant i \leqslant k$. Let $\mathcal{S}$ denote the set of all finite sequences, and let $\mathcal{P} \subseteq \mathcal{S}$ denote a given subset of sequences. We think of $\mathcal{P}$ as specifying a set of sequences having a certain property; e.g., if S is a set of n points in the plane, $\mathcal{P}$ may denote the set of sequences that correspond to the upper chain of some x-monotone n-gon having vertex set S.

Let $X \in \mathcal{P}$ be a random sequence. Then we say that sequence X is *chosen uniformly at random* (or, simply, "at random") from $\mathcal{P}$ if

$$P(X = \sigma) = \begin{cases} \dfrac{1}{|\mathcal{P}|} & \text{if } \sigma \in \mathcal{P}; \\ 0 & \text{otherwise.} \end{cases}$$

We say that $\sigma = (\sigma_1, \ldots, \sigma_l) \in \mathcal{S}$ is an *extension* of sequence $\tau = (\tau_1, \ldots, \tau_k) \in \mathcal{S}$ if $l \geqslant k$ and $\sigma_i = \tau_i$ for $1 \leqslant i \leqslant k$. Given a sequence $\tau = (\tau_1, \ldots, \tau_k) \in \mathcal{S}$, we let $\mathcal{P}_\tau$ denote the set of all sequences $\sigma = (\sigma_1, \ldots, \sigma_l) \in \mathcal{P}$ that are extensions of τ. Finally, let () denote the "empty" sequence; any sequence is an extension of ().

Consider the following incremental algorithm, which iteratively extends a sequence, with selection probabilities based on the counts on the number of feasible extensions of the sequence chosen so far. Let $X_0 = ()$. Then, for $i \geqslant 1$, select X_i according to the conditional distribution

$$P(X_i = \sigma \mid X_{i-1} = \tau) = \begin{cases} \dfrac{|\mathcal{P}_\sigma|}{|\mathcal{P}_\tau|} & \text{if } \sigma \text{ is an extension of } \tau; \\ 0 & \text{otherwise.} \end{cases}$$

Since we consider only finite sequences, the algorithm must terminate with a sequence $X = X_k \in \mathcal{P}$. It is an elementary consequence of conditional probability that this incremental procedure yields a random sequence.

Lemma 2.1. *The sequence X obtained by the incremental algorithm above is chosen uniformly at random from the set $\mathcal{P}$.*

Proof. Let $X_0, X_1, \ldots, X_k = X$ be the random variables and let $\sigma^0 = ()$, $\sigma^1, \ldots, \sigma^k$ be the k distinct sequences chosen in the incremental algorithm. By conditional probability,

$$\mathrm{P}(X_k = \sigma^k) = \mathrm{P}(X_k = \sigma^k \mid X_{k-1} = \sigma^{k-1}) \times \cdots \\ \times \mathrm{P}(X_2 = \sigma^2 \mid X_1 = \sigma^1) \times \mathrm{P}(X_1 = \sigma^1 \mid X_0 = ()).$$

But the algorithm gives $\sigma^1, \sigma^2, \ldots, \sigma^k$ according to the conditional probabilities given above, so we get, upon substituting that

$$\mathrm{P}(X = \sigma^k) = \frac{|\mathcal{P}_{\sigma^k}|}{|\mathcal{P}_{\sigma^{k-1}}|} \times \cdots \times \frac{|\mathcal{P}_{\sigma^2}|}{|\mathcal{P}_{\sigma^1}|} \times \frac{|\mathcal{P}_{\sigma^1}|}{|\mathcal{P}_{()}|} = \frac{1}{|\mathcal{P}_{()}|} = \frac{1}{|\mathcal{P}|},$$

where we have used the facts that $|\mathcal{P}_{\sigma^k}| = 1$ (since $\sigma^k \in \mathcal{P}$) and $\mathcal{P}_{()} = \mathcal{P}$. □

The question that remains is how to extend subsequences with the indicated probabilities. This will be answered in specific cases throughout the rest of the paper. In general, we will establish recurrence relations to allow us to count $|\mathcal{P}_\sigma|$. In the next section, on monotone polygons, we explicitly work backwards through the counting recurrence to generate the polygon of a given number; the fact that the resulting polygon is chosen at random from the set of N possible polygons follows from establishing a bijection between the set of polygons and the integers $\{1, \ldots, N\}$. In the case of convex polygons, we will generate the polygon incrementally using the conditional probabilities established in the above algorithm. The randomness of the resulting polygon will then follow directly from the lemma.

We assume a Real RAM model of computation, in which arithmetic operations take constant time [12]. Because we may be counting exponentially many polygons, the more appropriate log-cost RAM increases the running time by a linear factor. The conditional probabilities approach reveals that it is the proportions, and not the exact counts, that are important. If one is lucky enough to have reducible fractions then storing the proportions may require fewer bits.

3. Generating random monotone polygons

Let $S_n = \{s_1, s_2, \ldots, s_n\}$ be a set of n points in the plane, sorted according to their x-coordinate. We assume in this section that no two points have the same x-coordinates. We will generate uniformly at random a monotone polygon with vertex set S_n. "Monotone" in this paper will always mean x-monotone.

Let $S_i = \{s_1, s_2, \ldots, s_i\}$ for $1 \leqslant i \leqslant n$. Any monotone polygon constructed from S_i can be divided into two monotone chains—a *top chain* and a *bottom chain* as depicted in Fig. 1—for which the leftmost vertex is s_1 and the rightmost vertex is s_i. Points s_1 and s_i are on both chains; any other point in S_i is on either the top or bottom chain. Let $N(i)$ denote the number of monotone polygons

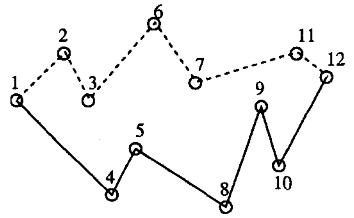

Fig. 1. Monotone chains.

with vertex set S_i. The number $N(i)$ will turn out to be the number of completions of subsequences of monotone polygons that go from s_{i-1} to s_n and back to s_i.

For convenience, we frequently denote the line segment or polygon edge $\overline{s_i s_j}$ by (i,j) and the line $\overleftrightarrow{s_i s_j}$ by $\ell(i,j)$.

We generate a random monotone polygon by scanning S_n forward and counting all monotone polygons, then picking a random number and scanning backward to generate the polygon of that number. We show how to count monotone polygons in Section 3.1 and how to generate one at random in Section 3.3. Our algorithms depend on having the visibility graph of the monotone chain joining the vertices of S_n; this is discussed in Section 3.2.

3.1. Counting completions to monotone polygons

We begin by establishing a recurrence that counts the monotone polygons on S_i in terms of those on S_j for $j < i$.

Note that a monotone polygon with vertex set S_i has edge $(i-1,i)$ as one of the two edges incident to s_i. Let $\mathcal{T}(i)$ be the set of monotone polygons with vertex set S_i that have edge $(i-1,i)$ on their top chain and define the number of these polygons $T(i) = |\mathcal{T}(i)|$. Similarly, let $\mathcal{B}(i)$ be the set of monotone polygons with vertex set S_i that have edge $(i-1,i)$ on their bottom chain and define $B(i) = |\mathcal{B}(i)|$. We will see that our recurrence actually counts $T(i)$ in terms of $B(j)$ for $j < i$. We begin by noting the relationship between $N(k)$, $T(k)$ and $B(k)$ in Lemma 3.1.

Lemma 3.1. *For any point set S_k with $k > 2$, the number of monotone polygons with vertices S_k is*

$$N(k) = T(k) + B(k). \tag{1}$$

Proof. The sets $\mathcal{T}(k)$ and $\mathcal{B}(k)$ are disjoint and cover the set of monotone polygons on S_k, since each monotone polygon has the edge $(k-1,k)$ on either the top or bottom chain and only the degenerate two-vertex polygon has the edge on both chains. □

We say that a point s_i is *above-visible* from s_k if $i < (k-1)$ and s_i is above the line $\ell(j,k)$, for all points s_j with $i < j < k$. Similarly, s_i is *below-visible* from s_k if $i < (k-1)$ and s_i is below $\ell(j,k)$, for $i < j < k$. In other words, if we treat the monotone chain on S_n as an "obstacle", then s_i is above-visible (or below-visible) from s_k if the two vertices are visible to each other, in the usual sense, and the segment (i,j) lies above (or below) the obstacle. Let $V_T(k)$ be the set of points that

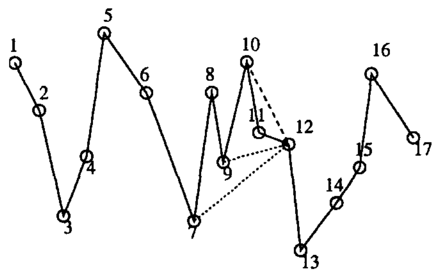

Fig. 2. $V_T(12) = \{10\}$ and $V_B(12) = \{7,9\}$.

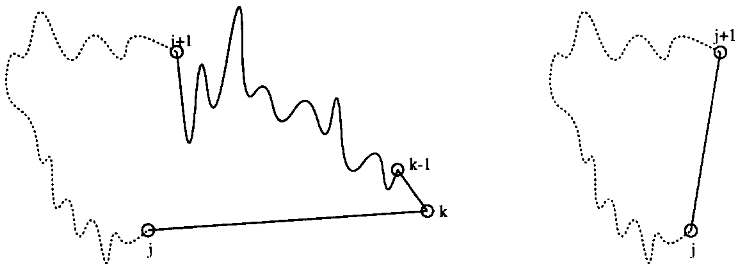

Fig. 3. The original polygon and its set of completions $\mathcal{B}(j+1)$.

are *above-visible* from point s_k, and let $V_B(k)$ be the set of points that are *below-visible* from point s_k, as in Fig. 2. We can now count monotone polygons on S_k where both edges into s_k are specified.

Lemma 3.2. *The number of polygons in $\mathcal{T}(k)$ that contain edge (j,k), for $j \in V_B(k)$, is $B(j+1)$. The number in $\mathcal{B}(k)$ that contain edge (j,k), for $j \in V_T(k)$, is $T(j+1)$.*

Proof. Let $P(j,k)$ be the set of polygons in $\mathcal{T}(k)$ with (j,k) as a bottom edge, for $j \in V_B(k)$. For the polygons in $P(j,k)$, we know that points s_j and s_k are on the bottom chains, and $s_{j+1},\dots,s_k$ are on the top chains. So the path of s_j, s_k, s_{k-1}, $\rightsquigarrow s_{j+1}$ is fixed. We can treat this path as an edge $(j,j+1)$ that is on the bottom chain. Fig. 3 shows an example. Thus, $|P(j,k)|$ equals the number of monotone polygons generated from S_{j+1} with the edge $(j,j+1)$ on the bottom chains, which is $B(j+1)$. □

Theorem 3.3. *For any point set S_k with $k > 2$, we have*

$$T(k) = \sum_{j \in V_B(k)} B(j+1), \tag{2}$$

$$B(k) = \sum_{j \in V_T(k)} T(j+1). \tag{3}$$

Proof. We prove formula (2). According to the definition of below-visible, the bottom edge (j,k) of any $P \in \mathcal{T}(k)$ uses a point $s_j \in V_B(k)$. By Lemma 3.2 there are $B(j+1)$ monotone polygons having edges $(k-1,k)$ and (j,k). Therefore, we have $\sum_{j \in V_B(k)} B(j+1)$ polygons in total. □

This theorem gives us a procedure to calculate $T(n)$ and $B(n)$, assuming that we have $V_B(k)$ and $V_T(k)$. We can start with $T(2) = B(2) = 1$, since in the degenerate case of two vertices the line segment can be considered as the top and the bottom edge of a degenerate polygon. Then we use the recurrence to determine $T(i)$ and $B(i)$ for $i := 3$ to n.

3.2. Computing visibility

The counting in the previous section needed the *above-visible* and *below-visible* sets, $V_T(k)$ and $V_B(k)$ for $k = 1,\dots,n$, which comprise the visibility graph of the monotone chain on S_n. These

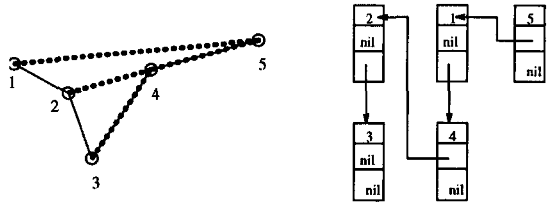

Fig. 4. Point set S_5 and tree(5).

sets can be constructed in time proportional to their total size by the output-sensitive algorithm of Hershberger [7], which works for arbitrary simple polygons. A closer look shows that these sets are needed in increasing order for one index i at a time, which allows us to compute them in $O(n)$ space. Hershberger's algorithm can be simplified in our special case of a monotone chain, so we include details for completeness. We focus on $V_T(k)$ because the computation of $V_B(k)$ is analogous.

Let S_k denote the monotone chain with vertices $s_1, s_2, \ldots, s_k$. If we think of S_k as a fence and compute the shortest paths in the plane above S_k from s_k to each s_i with $i \leqslant k$, then we obtain a tree that is known as the *shortest path tree rooted at* s_k [5,6], which we denote tree(k). The *above-visible* set $V_T(k)$ is exactly the set of children of s_k in tree(k). Thus, we will incrementally compute tree(1), ..., tree(k) to get the *above-visible* sets. We compute $V_T(k)$ from $V_T(k-1)$ by computing a shortest path tree tree(k) from tree($k-1$). The idea is the following.

We represent a shortest path tree tree(k) (in which a node may have many children) by a binary tree in which each node has pointers to its uppermost child and next sibling. For each vertex $j \in [1, n]$, we have a record

j:	*ptr*	*ptr* stores the coordinates of vertex j;
	upc	pointer *upc* points to the upper child of j in tree(k);
	sib	pointer *sib* points to the sibling of j in tree(k).

The initial tree, tree(1), has a single record with $1.ptr = s_1$, $1.upc = nil$ and $1.sib = nil$. We assume that tree($i-1$) has been computed and call the procedure Make_top($i-1, i, tmp$) to calculate the tree(i). The upper child pointer *i.upc* will be set to *tmp.sib*.

```
Make_top(j, k, Var: lastsib)
    While j.upc ≠ nil and k is above ℓ(j.upc, j)
        Make_top(j.upc, k, Var: lastsib); /* Make subtree for this child of j, which is visible from k. */
        j.upc = j.upc.sib;                /* Consider next child of j. */
    End While
    lastsib.sib = j;                      /* Make the connection to j, one of the children of k. */
    lastsib = j;
```

Let r be a record in tree(k). We define $r.sib^0 = r$, and $r.sib^i = r.sib^{i-1}.sib$, for any integer $i \geqslant 0$. Let $\mathcal{CT}(k)$ be the set of points $\{j \mid j = k.upc.(sib)^i \text{ for } i \geqslant 0\}$; these are the children of k in tree(k). We show, in the next theorem, that these are the only vertices visible from k.

Theorem 3.4. *The above-visible vertices* $V_T(k) = \mathcal{CT}(k) - \{k-1\}$.

Proof. If $V_T(k) = \emptyset$ then no point is above line $\ell(k-1,k)$. This means that there is no $\ell(i,k-1)$ that is below k. From `Make_top()`, we know that $\mathcal{CT}(k) = \{k-1\}$, hence $V_T(k) = \mathcal{CT}(k) - \{k-1\}$. Conversely, if $\mathcal{CT}(k) = \{k-1\}$ then no $\ell(i,k-1)$ is below k for $1 \leqslant i < k-1$. So there exists no point that is above $\ell(k-1,k)$. Hence

$$V_T(k) = \emptyset = \mathcal{CT}(k) - \{k-1\}.$$

In the general situation, each $j \in V_T(k)$ is above all $\ell(i,k)$ for $j < i < k$. Thus, k is above all $\ell(j,i)$ for $j < i < k$. Now we prove $j = k.upc.(sib)^i$, for some $i \geqslant 0$. If there is no $j' \in V_T(k)$ and $j' < j$ such that k is above $\ell(j',j)$ then $j = k.upc$. Otherwise, $j = j'.sib$. Similarly this induction can be applied to j', that is, $j' = k.upc.(sib)^{i'}$. Then we have $j = k.upc.(sib)^{i'+1}$. So $V_T(k) \subseteq \mathcal{CT}(k) - \{k-1\}$.

For each $j \in \mathcal{CT}(k) - \{k-1\}$ we know that $j = k.upc.(sib)^i$. Then j is above all $\ell(i,k)$, for $j < i < k$. Otherwise, there exists a point, say j', such that $j' > j$ and j is below $\ell(j',k)$. Then $\ell(j,k)$ is below $\ell(j',k)$, which means that k is below $\ell(j,j')$. From `Make_top()` we know that j can not be expressed as $k.upc.(sib)^i$, for $i \geqslant 0$. This contradiction proves that j is above all $\ell(i,k)$, for $j < i < k$. Therefore $j \in V_T(k)$ and we conclude that $V_T(k) \supseteq \mathcal{CT}(k) - \{k-1\}$. We conclude that $V_T(k) = \mathcal{CT}(k) - \{k-1\}$. □

We observe that `Make_top()` runs in time proportional to the number of edges that it finds, and thus that we can count $\mathcal{T}(k)$ and $\mathcal{B}(k)$ in time proportional to their sizes.

Lemma 3.5. *The runtime of* `Make_top`$(k-1, k,$ `Var`$:t)$ *is* $O(|V_T(k)|)$.

Proof. Each call to `Make_top()`, except the first, implies that the calling procedure found a child of k. All other work in the procedure takes constant time per call. □

Corollary 3.6. *Our algorithm determines* $\mathcal{T}(i)$ *and* $\mathcal{B}(i)$, *for* $1 \leqslant i \leqslant n$, *using* $O(n)$ *space and* $O(T(n) + B(n))$ *time overall.*

3.3. Generating monotone polygons uniformly

Once we have $T(i)$ and $B(i)$, for all $i \leqslant n$, we can generate a monotone polygon on vertex set S_n uniformly at random using the conditional probabilities of Section 3.1. The `Generate()` algorithm extends a subsequence from right to left to generate a monotone polygon. It runs in $O(n)$ time and space by constructing only a linear number of the visibility edges.

```
Generate(S_n)
    Pick x ∈ [1, N(n)] at random;
    Add s_n to top_chain;
    Add s_n to bottom_chain;
    If x ≤ T(n)
        Add s_{n-1} to top_chain;
        Generate_Top(n, x);
    Else
        x = x − T(n);
        Add s_{n-1} to bottom_chain;
        Generate_Bottom(n, x);
```

```
Generate_Top(k, x)
1.  If k ≤ 2 then Add s_1 to top_chain; return;
2.      sum = 0; i = t = k − 1;
3.      Loop                          /* Find partial sum ≥ x. */
4.          i = i − 1;
5.          If s_i is below line ℓ(t, k)
6.              t = i;                /* New visibility edge (i, k) */
7.              sum = sum + B(i + 1);
8.      Until x ≤ sum;
9.      Add s_i to bottom_chain;
10.     Add s_{k-2}, s_{k-3}, ..., s_{i+1} to top_chain;
11.     k = i + 1;
12.     x = x − (sum − B(i + 1));
13. Generate_Bottom(k, x)
```

`Generate_Top()` and `Generate_Bottom()` are two mutually recursive procedures. `Generate_Bottom()` can be obtained from `Generate_Top()` by swapping "top"s and "bottom"s, "T"s and "B"s, "above"s and "belows."

`Generate_Top()` completes a polygon sequence in which s_{k-1} is on the top chain and s_k is on the bottom; thus, $(k-1, k)$ is an edge of the top chain in the completion polygon. It generates the edge of the completion that joins to s_k with the appropriate probability by starting with a random integer $x \in [1 \ldots T(k)]$ and determining which partial sum has

$$x \leqslant \sum_{(j \in V_B(k)) \wedge (j \geqslant i)} B(j+1).$$

These partial sums are evaluated starting with the high indices so that each point can be considered as the left endpoint of a below-visible edge at most once.

Theorem 3.7. *Given $T(i)$ and $B(i)$ for $1 \leqslant i \leqslant n$, one can generate a monotone polygon on S_n uniformly at random in $O(n)$ time and space.*

Proof. We need to argue that the probabilities are correct and that the algorithm runs in linear time.

We sketch the pieces of the induction that proves correctness. The initial computation in `Generate()` insures that x is chosen uniformly at random in $[1 \ldots T(n)]$ before calling `Generate_Top(n, x)`. If we assume that x is chosen uniformly at random in $[1 \ldots T(k)]$, then we know by Eq. (2) that there is an index $i < k$ with

$$\sum_{(j \in V_B(k)) \wedge (j > i)} B(j+1) < x \leqslant \sum_{(j \in V_B(k)) \wedge (j \geqslant i)} B(j+1).$$

The loop in lines 3–8 finds this index i by accumulating the partial sums whenever it finds a new visibility edge in $V_B(k)$. The final edge (i, k) is added to current sequence on the bottom chain and vertices are added to the top chain to catch up. Because x was chosen at random, this new sequence has the correct probability, $B(i+1)/T(k)$. The new value of x computed in line 12 also lies randomly in $[1 \ldots B(i+1)]$.

For the running time of a call `Generate_Top`(k, x), suppose that the loop in lines 3–8 is executed m times. Because $m \geqslant 1$, the amount of work performed in lines 1–12 of the procedure is proportional to m. For the recursive call in line 13, k has decreased to $k - m$. Because the recursion bottoms out when $k \leqslant 2$, the total amount of work is linear. □

3.4. Generating nested monotone polygons

We can modify our algorithm to generate, on a given vertex set S_n, a random x-monotone polygon that is nested inside another x-monotone polygon P. All we need to change is the definition and computation of visibility.

We say that s_i is *below-visible* from s_k if $i < (k-1)$, the line segment (i, k) does not intersect the exterior of P, and s_i is below $\ell(j, k)$, for $i < j < k$. Similarly, s_i is *above-visible* from s_k if $i < (k-1)$, the line segment (i, k) does not intersect the exterior of P, and s_i is above $\ell(j, k)$, for $i < j < k$.

The visible sets $V_T(k)$ and $V_B(k)$ under this new definition of visibility can be computed both forward and backwards in time proportional to their size with a time and space overhead of $O(n+|P|)$. The additional computation is essentially to compute the *relative convex hull* of P [5,13] and S_k up to the vertical line through the point s_k.

Theorem 3.8. *One can count the monotone polygons having vertex set S_n that are nested inside a monotone polygon P in $O(n + |P|)$ space and $O(n + |P| + K)$ time, where K is the total number of above-visible and below-visible points. Thereafter, one can generate these polygons uniformly at random in $O(n + |P|)$ time.*

Remark. Note that there may be *no* polygon within P whose vertex set is S_n; in this case, our algorithm reports that none exists.

4. Generating convex polygons

Researchers have studied several ways to generate random convex polygons, including random point processes [14], random line processes [1,10], and Voronoi cells of random points [2,3]. These approaches, however, generate the polygon vertices at random; they do not allow one to influence the distribution by generating a random n-gon from a given set of n points. Of course, a given n points admit *at most one* convex n-gon. Thus, we consider the problem of generating at random a convex polygon from among all convex polygons whose vertex set is a *subset* of the given n points.

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n distinct points in the plane. We consider the problem of generating a random convex polygon whose vertices are from set S. If $n \geqslant 3$, there is always at least one convex polygon on S. Of course, for $k > 3$, there may not exist a convex k-gon determined by points S. (Consider the example of $n/3$ concentric, homothetic, equilateral triangles; there are no nondegenerate convex k-gons determined by the n corners, for $k \geqslant 4$.) It is known how to count the number of convex polygons determined by n points in time $O(n^3)$ [9]. We now show how this leads to a polynomial-time algorithm for random generation of convex polygons.

A convex k-gon P can be associated uniquely with the sequence $\sigma = (\sigma_1, \ldots, \sigma_k, \sigma_1)$, where $\sigma_i \in S$ $(1 \leqslant i \leqslant k)$ are the vertices of P, and σ_1 is the vertex of P having minimum y coordinate. (Assume for simplicity that no two points of S have the same y coordinate.) Let $\mathcal{P}$ denote the set of all such sequences, for all $k \leqslant n$. We explicitly append σ_1 to the end of σ in order to discriminate between the sequence corresponding to the closed polygon $(\sigma_1, \dots, \sigma_k, \sigma_1)$ and the sequence corresponding to the open convex chain $(\sigma_1, \dots, \sigma_k)$, which may have a completion into a convex polygon with more than k vertices.

Let us be specific about how the incremental construction method applies to this case. We begin with $X_0 = ()$. We then select $\sigma_1 \in S$ according to the rule that $\mathrm{P}(\sigma_1 = s) = 1/L(s)$, where $L(s)$ is the number of convex polygons with vertices among S, such that $s \in S$ is the lowest vertex. Thus, $L(s)$ is the number of elements of $\mathcal{P}$ that are the completion of the one-element sequence (s). Next, we select σ_2 from among $S \setminus \{\sigma_1\}$, according to

$$\mathrm{P}(\sigma_2 = s) = \frac{f(\sigma_1, s; \sigma_1)}{L(\sigma_1)},$$

where $f(p, q; \sigma_1)$ is the number of convex chains from q to σ_1 that lie above σ_1 and to the left of the (directed) segment pq. We select σ_3 from among $S \setminus \{\sigma_1, \sigma_2\}$, according to

$$\mathrm{P}(\sigma_3 = s) = \frac{f(\sigma_2, s; \sigma_1)}{f(\sigma_1, \sigma_2; \sigma_1)}.$$

Continuing, we select σ_i (for $i \geqslant 4$) from among $S \setminus \{\sigma_2, \dots, \sigma_{i-1}\}$, according to

$$\mathrm{P}(\sigma_i = s) = \frac{f(\sigma_{i-1}, s; \sigma_1)}{f(\sigma_{i-2}, \sigma_{i-1}; \sigma_1)}.$$

Note that we allow σ_i to equal σ_1 for $i \geqslant 4$, so that the polygon can close and the algorithm terminate.

It remains to describe how to tabulate the functions $L(s)$ and $f(p, q; s)$. This is done in [9]; we include it here for completeness. Our discussion follows that of [9]. First, we define $H_{p,q}$ to be the open halfplane that lies to the left of the directed line through pq. Next, we fix point s and we restrict ourselves to points of S that lie above s (in y coordinate). Now, visit points $q \in S$ ($q \neq s$) in clockwise order about s, evaluating

$$f(s, q; s) = \sum_{r \in H_{s,q}} f(q, r; s)$$

and

$$f(p, q; s) = 1 + \sum_{r \in H_{p,q} \setminus H_{q,s}} f(q, r; s).$$

The justification of the expression for $f(p, q; s)$ is simple: in any convex chain joining q to s, lying left of pq, either we join q to s directly, thereby closing the polygon, or we join q to a point r that is left of pq and not left of qs. As written, these recursions can be evaluated in $\mathrm{O}(n)$ time for each choice of p, q, s, giving $\mathrm{O}(n^4)$ time overall. This can be improved by noting that, for fixed values of q and s, we can evaluate $f(p, q; s)$ incrementally for points p in clockwise order about q. Specifically, if the points are labelled $p_1, p_2, \dots$ in clockwise order about q (with p_1 being the first point hit by rotating clockwise the ray from q in the direction opposite to s), then we can compute $f(p_i, q; s)$ from $f(p_{i-1}, q; s)$ according to

$$f(p_i, q; s) = f(p_{i-1}, q; s) + \sum_{r \in H_{p_i,q} \setminus H_{p_{i-1},q}} f(q, r; s).$$

The result is that, for the fixed choice of s, the values $f(p, q; s)$ can be tabulated in $O(n^2)$ time using $O(n^2)$ storage. (It is also possible to decrease the storage space to $O(n)$, at the expense of a factor of n in the running time. We omit details here.) Finally, we compute

$$L(s) = \sum_q f(s, q; s),$$

and store this value with point s. As we loop through all choices of s, the total time required is $O(n^3)$. To generate a random polygon, we select the bottom point, σ_1, and compute and store the values of $f(p, q; \sigma_1)$; this takes $O(n^2)$ time and space.

Theorem 4.1. *After $O(n^3)$ preprocessing on a set S of n points, one can in $O(n^2)$ time generate uniformly at random a convex polygon whose vertices are among the points S.*

Remark. Using similar recurrences and an extra factor of n in running time, one can compute the number of convex k-gons, for all $1 \leqslant k \leqslant n$, that are determined by a given set of n points; see [9]. This leads to a method of generating random convex k-gons, for a given value of k, from among those determined by n points.

5. Generating simple polygons

In applications, we frequently want to generate random simple polygons. Unfortunately, the counting problem for simple polygons appears to be quite difficult. It is open whether or not one can compute the number of simple n-gons with a given vertex set in time bounded by a polynomial of n.

One can, of course, generate permutations at random and check for simplicity. The worst-case for this approach occurs when the points are in convex position—only $2n$ of the $n!$ permutations correspond to the convex hull, which is the only simple polygon. In general, for a given vertex set, we would like to count and to enumerate only those permutations that correspond to simple polygons. We know of no efficient enumeration procedure for simple polygons and no polynomial-time algorithm for counting the number of simple polygons on a given vertex set.

One approach that leads to a polynomial-time algorithm is to generate a random permutation and then apply 2-opt moves to pairs of intersecting edges—removing two intersecting edges and replacing them with two non-intersecting edges so as to keep the polygon connected. One can observe that this replacement decreases total length and therefore converges to a simple polygon. (Indeed, this was a Putnam examination problem.) Van Leeuwen and Schoone [15] showed that at most $O(n^3)$ of these "untangling 2-opt" moves can be applied, no matter in what order they are done; the geometric dual of their argument is a good example of the power of duality. Even though this approach does generate each possible simple polygon with some positive probability, it does not generate simple polygons

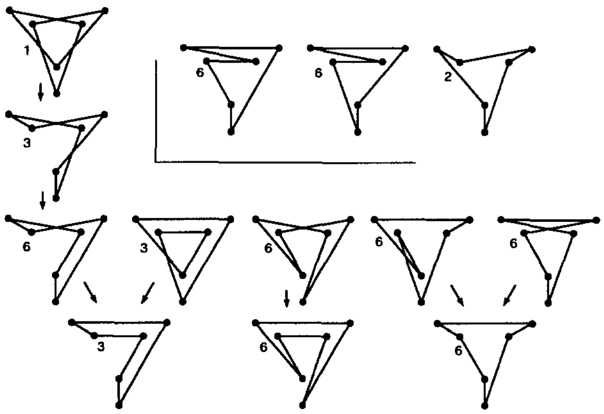

Fig. 5. Polygons obtained by "untangling 2-opt" for the given six vertices. The numbers are how many different polygons are obtained by rotating or reflecting the point set.

uniformly at random: Some polygons in Fig. 5 have a single permutation that generates them, while others have several.

One practical approximation method, suggested by one of the referees, is to start with a monotone simple polygon and apply some simplicity-preserving, reversible operations (including the identity) with the property that any simple polygon is reachable by a sequence of operations. Let $d(P)$ be the number of operations applicable to polygon P. If one randomly chooses an operation, then one obtains an ergodic Markov chain that converges to a stationary distribution where polygon P has probability proportional to $d(P)$. After applying a large number of operations, therefore, one can accept the resulting polygon P with probability $1/d(P)$. It is interesting to study the complexity and convergence rates of different operations.

6. Conclusion

In this paper we have considered the problem of generating a polygon at random, using a *given* vertex set. This definition of "random polygons" separates the choice of vertex set from the choice of edges. As we have shown, the random generation problem is then intimately connected with the counting problem.

For the special case of monotone polygons, we solve the counting and generation problems. Specifically, we have shown how to count and to generate, uniformly at random, the x-monotone polygons that have a given n-point set S_n as their vertices. Counting takes $O(n)$ space and $O(K)$ time, where $n < K < n^2$ is the number of edges of the visibility graph of the monotone chain on S_n. After counting, generation takes $O(n)$ space and time. This algorithm has been implemented using $O(K)$ space; it works well for small values of n, but requires extended precision when the number of polygons on a set exceeds the largest integer that can be stored.

For the special case of convex polygons, we have given an $O(n^2)$ algorithm (after an $O(n^3)$ preprocessing step) for generating a random convex polygon whose vertices are among a given set of n points.

We also gave extensions to some related random generation problems. Our algorithm for monotone polygons generalizes to allow us to generate a random x-monotone polygon that is nested inside a given simple polygon P. This is a useful feature in generating test instances for GIS algorithms on map data that consists of polygonal subdivisions with nesting faces. Other generalizations of our approach leads to polynomial-time methods to

- Generate a random x-monotone polygon that has a given number $k < n$ of vertices from the set S.
- Generate multiply nested hierarchies of a constant number of nested monotone polygons.
- Generate a random simple polygon whose boundary consists of at most a constant number of x-monotone chains using the vertex set S.

In each of these generalizations, the principal idea is to set up recursions to count the number of feasible completions for the given restricted class of polygons. In each case, the fact that there is only a constant-size description of a partial polygon allows the counting to proceed, by recursion, in polynomial time and space. The exponent of the polynomial depends, of course, on the "constant." This fact makes many of these generalizations impractical in most cases.

Finally, we have briefly discussed the difficulty of generating random simple polygons. It is a challenging open problem to determine if a polynomial-time algorithm exists to generate a simple polygon at random, from the set of all simple polygons on a given vertex set.

Acknowledgements

David Kirkpatrick suggested that generation required only $O(n)$ time after counting. Fig. 5 was developed with Sandor Fekete. We thank both of them for discussions on generating random simple polygons and thank the referees for their comments. We thank Thomas Auer and Martin Held for finding and correcting an error in an earlier draft.

References

[1] K. Abrahamson, On the modality of convex polygons, Discrete Comput. Geom. 5 (1990) 409–419.
[2] B.N. Boots, Edge length properties of random Voronoi polygons, Metallography 20 (2) (1987) 231–236.
[3] I.K. Crain, The Monte-Carlo generation of random polygons, Comput. Geosci. 4 (1978) 131–141.
[4] P. Epstein and J. Sack, Generating triangulations at random, in: Proc. 4th Canad. Conf. Comput. Geom. (1992) 305–310.
[5] L.J. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons, Algorithmica 2 (1987) 209–233.
[6] L.J. Guibas and J. Hershberger, Optimal shortest path queries in a simple polygon, J. Comput. Syst. Sci. 39 (2) (1989) 126–152.
[7] J. Hershberger, An optimal visibility graph algorithm for triangulated simple polygons, Algorithmica 4 (1989) 141–155.
[8] H. Meijer and D. Rappaport, Upper and lower bounds for the number of monotone crossing free Hamiltonian cycles from a set of points, Ars Combin. 30 (1990) 203–208.
[9] J.S.B. Mitchell, G. Rote, G. Sundaram and G. Woeginger, Counting convex polygons in planar point sets, Inform. Process. Lett. 56 (1995) 45–49.
[10] R.E. Miles, Random polygons determined by random lines in a plane, Proc. Nat. Acad. Sci. 52 (1964) 901–907 and 1157–1160.

[11] J. O'Rourke and M. Virmani, Generating random polygons, Technical Report 011, Smith College (1991).
[12] F.P. Preparata and M.I. Shamos, Computational Geometry—An Introduction (Springer, New York, 1985).
[13] G.T. Toussaint, Computing geodesic properties inside a simple polygon, Revue d'Intelligence Artificielle 3 (2) (1989) 9–42. Also available as technical report SOCS 88.20, School of Computer Science, McGill University.
[14] P. Valtr, Probability that n random points are in convex position, Technical Report B 94-01, Freie Universität Berlin (January 1994).
[15] J. van Leeuwen and A.A. Schoone, Untangling a traveling salesman tour in the plane, in: J.R. Mühlbacher, ed., Proc. 7th Conf. Graphtheoretic Concepts in Comput. Sci., Linz, 1981 (Hanser, München, 1982) 87–98.