Folding a Triangulated Simple Polygon: Structural and Algorithmic Results

Ali A. Kooshesh and Bernard M. E. Moret

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131
(kooshesh@cs.unm.edu and moret@cs.unm.edu)

ABSTRACT

We describe a linear-time algorithm that folds a triangulated simple polygon into a single triangle. Using this technique, we derive a particularly simple proof of Chvátal's art gallery theorem and improve or simplify a number of algorithms that deal with triangulated simple polygons. We describe two improved algorithms, both based on the degree sequence of the boundary vertices of the given triangulated simple polygon. The first is a linear-time algorithm that, using only the parity of the degree of each vertex, colors the vertices of a triangulated simple polygon using three colors. The second algorithm reconstructs the polygon and its triangulation from the degree sequence. We then show that our results extend to k-trees.

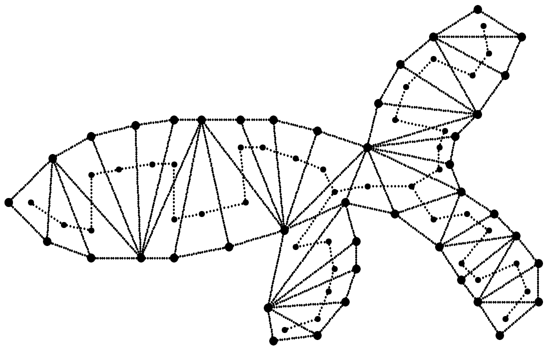
1 Introduction

We describe a linear-time algorithm that folds a triangulated simple polygon (hereafter referred to as a TSP) into a single triangle. Using this technique, we can improve and simplify a number of algorithms that deal with triangulated simple polygons, as well as offer a structural characterization of various attributes of such polygons. Such characterizations include another simple proof of Chvàtal's Watchman Theorem [3]. We describe two improved algorithms, both based on the degree sequence of the boundary vertices of the given triangulated simple polygon. The first is a linear-time algorithm that colors the vertices of a triangulated simple polygon using three colors; this algorithm, which only uses the parity of the degree of each vertex, improves on the $O(n \log n)$ algorithm of Avis and Toussaint [1]. The second algorithm reconstructs the triangulation from the degree sequence in linear time; Beyer *et al.* [2] gave a similar algorithm for maximal outerplanar graphs, but our approach is simpler and more intuitive. We then show that our results extend to k-trees.

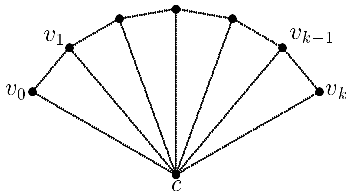
2 Folding a TSP

We begin by introducing some terminology. The *dual tree* of a TSP is the tree formed by associating a vertex with each triangle and connecting two vertices exactly when their associated triangles share an edge. A *fan* is a sequence of adjacent triangles with a common apex, the *center* of the fan; the *base-path* of a fan is the simple path $v_0, v_1, \ldots, v_k$ defined by the traversal of vertices adjacent to the center of the fan. Note that the degree of vertices v_1 through v_{k-1} must be exactly three; a fan in which neither v_0 nor v_k has degree three is called *maximal.* A *sleeve* is the union of adjacent maximal fans, the end-fans of which each have a vertex of degree two; the dual tree of a sleeve is a simple path. Figure 1 illustrates these concepts.

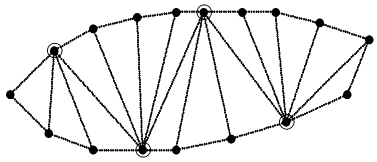
We fold a sleeve by folding one of its end-fans, thereby reducing the number of fans by one, and repeating the process until a single triangle results; we fold a TSP by folding subpolygons that form sleeves. More precisely, we proceed as follows. Let an end-fan of a sleeve have center c and base-path $v_0, v_1, \ldots, v_k$, where v_0 has degree two. We fold this fan by identifying vertex v_i with vertex v_{i+2}, for all i, $0 \leq i \leq k-2$, where v_i has degree two; that is, we fold the fan along diagonals $\{c, v_1\}$, $\{c, v_2\}$, $\ldots$, $\{c, v_{k-1}\}$, in this order; this process is illustrated in Figure 2. The fan is thus folded into the triangle $cv_{k-1}v_k$, where, if k is even (respectively, odd), v_k represents the class of vertices with even (respectively, odd) index, while v_{k-1} represents the class of vertices with odd (respectively, even) index. If the just folded fan was not the last fan left in the sleeve, we now fold along $\{c, v_k\}$,



a) A triangulated simple polygon and its dual tree



b) A fan



c) A sleeve with four fans

Figure 1: Dual trees, fans, and sleeves

thereby reducing the sleeve by one fan. We then repeat this process until the entire sleeve is folded into one triangle. To fold a TSP, we effectively traverse its dual tree from its leaves (the dual tree need not be known, but it facilitates the description). When the search reaches a vertex, y, of degree three, we fold the sleeve, the dual tree of which is the simple path from the leaf to y. As a result of this folding, the degree of y becomes two; thus, after folding the sleeve, the dual tree contains one fewer vertex of degree three; repeating this process eventually folds the TSP into one triangle (see Figure 3).

Lemma 1. The folding algorithm just described partitions the vertex set of any TSP into three equivalence classes (induced by the relation "is identified with") in such a way that, for any three consecutive vertices, p_i, p_{i+1}, and p_{i+2} (indices are modulo n), selected on the boundary of the polygon, p_i and p_{i+2} are in the same equivalence class if and only if the degree of p_{i+1} is odd. □

Proof: That the relation "is identified with" on the vertex set of any TSP is an equivalence relation is immediate. Vertex p_{i+1} is the center of a fan with base-path $p_i = v_0, v_1, \ldots, v_k = p_{i+2}$. For $0 \leq i < k$, if $\{v_i, v_{i+1}\}$ is a diagonal, we fold the section of the polygon that it cuts off into the triangle $v_i v_{i+1} p_{i+1}$. The result is a maximal fan in which v_0 and v_k both have degree two and the other vertices v_i, $1 \leq i \leq k-1$ have degree three. Folding this fan identifies vertices v_0 and v_k if and only if the degree of p_{i+1} is odd.

Q.E.D.

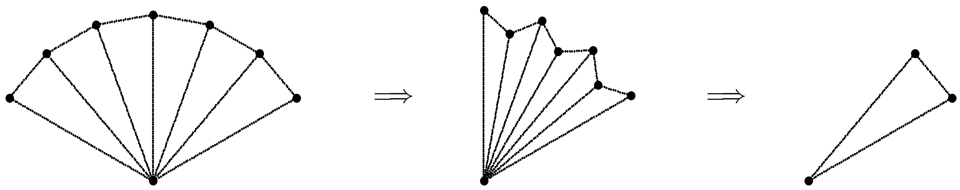
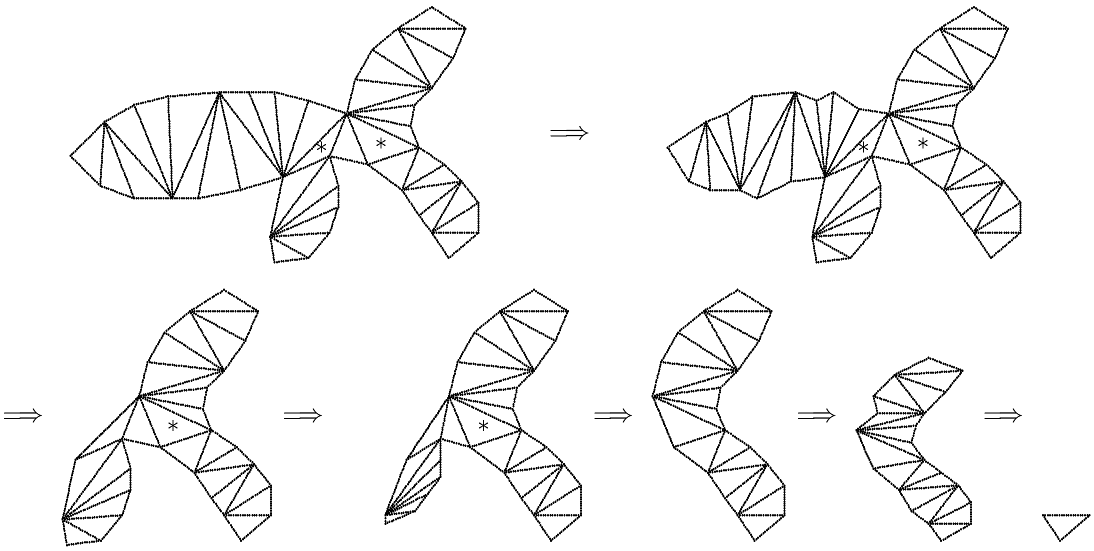


Figure 2: Folding a fan



$*$ denotes branching nodes in the dual tree

Figure 3: Folding a triangulated simple polygon

Let $\{C_1, C_2, C_3\}$ be the partition induced by the relation "is identified with" on the vertex set of a TSP. Because we never fold an edge, but fold along edges, we have the immediate corollary.

Corollary 1. No two vertices of C_i, $1 \leq i \leq 3$, are adjacent. In particular, the vertex set of any triangulated simple polygon is three-colorable. $\square$

This corollary allows us to give a particularly simple proof of a theorem of Chvàtal [3]; Fisk [4] also gave a simple proof based on colorability, but our folding technique allows us to reduce the problem of watching over the interior of a TSP to that of watching over the interior of a triangle.

Corollary 2. [Chvàtal's Watchman Theorem] Let $P = (V, E)$ be a simple polygon with $|V| = n$. There exists a subset $V' \subset V$, with $|V'| \leq \lfloor \frac{n}{3} \rfloor$, such that any point internal to P is visible from at least one vertex in V'. $\square$

Proof: Fold the triangulated polygon into one triangle and choose the vertex that represents the equivalence class with fewest members. This class contains at most $\lfloor \frac{n}{3} \rfloor$ vertices. Any point interior to the polygon is in a triangle, and every triangle has one vertex in the chosen class; therefore, any interior point is visible from at least one vertex.

Q.E.D.

As we fold a triangle along an edge, we can only fold it in one way; indeed, no matter where we start the folding process, the resulting partition is unique.

Corollary 3. The partition of the vertices of the TSP induced by the folding procedure is unique. $\square$

Proof: Assume, to the contrary, the existence of a different partition. Then there must exist vertices p_{i-1}, p_i, p_{i+1}, for some i, such that the folding procedure puts p_{i-1} and p_{i+1} in two different classes (implying that the degree of p_i is even), while the adversary puts them in the same class. But, since the length of the base-path of the fan centered at p_i is even, there must be two adjacent vertices in the same class, contradicting Corollary 1.

Q.E.D.

In fact, producing a three-coloring for a TSP is even simpler than folding the polygon: it suffices to know the parity of each vertex in order to produce a legal 2-coloring in linear time.

Corollary 4. Let T be a TSP given by its perimeter, $p_0, p_1, \ldots, p_n$. The following linear-time algorithm colors the vertex set of T using three colors.

```
Color(p_0) ← 1; Color(p_1) ← 2
for i = 1 to n − 1 do
   if odd(deg(p_i))
      then Color(p_{i+1}) ← Color(p_{i−1})
      else  Color(p_{i+1}) ← 6 − Color(p_{i−1}) − Color(p_i)
```

Proof: We use induction on the number of vertices. The algorithm obviously colors the vertices of a triangle with three colors. Assume that we add vertex p, external to T, between vertices p_i and p_{i+1}, with edges $\{p, p_i\}$ and $\{p, p_{i+1}\}$. Although this addition changes the parity of p_i and p_{i+1}, still the algorithm assigns the same color to vertices p_{i+1} and p_{i+2}. Since the degree of p_{i+2} has not been changed, the algorithm proceeds as in the case of inductive hypothesis and thus produces a legal coloration.

Q.E.D.

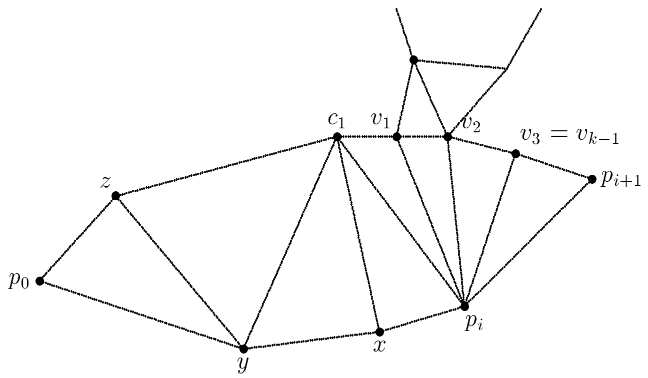
This algorithm improves on the $O(n \log n)$ algorithm of Avis and Toussaint [1], both in its running time and in its simplicity.

Assume that $v_0, v_1, \ldots, v_k, v_{k+1}$ is a sequence of vertices on the boundary of a TSP colored by the preceding algorithm, such that vertices v_0 and v_{k+1} have been assigned color 1 and vertices $v_1, \ldots, v_k$ have been assigned alternately colors 2 and 3. It is easy to see that: if k equals 1, then the degree of v_1 is odd; if k equals 2, then the degrees of v_1 and v_2 are both even; and, if k exceeds 2, then the degrees of v_1 and v_k are both even while the degree of each v_i, $1 < i < k$, is odd. Also note that, if the degree of every vertex of a TSP is even, then the number of vertices must be divisible by three.

3 Finding the Branching Triangles

A triangle is *branching* if all three of its edges are diagonals of the triangulation. The branching triangles in a triangulation give rise to vertices of degree three in the dual tree; in other words, a TSP is composed of sleeves attached to branching triangles. Since reconstructing a sleeve is very simple, the identification of branching triangles (or, equivalently, the identification of branching nodes when reconstructing the dual tree of the triangulation) is the crucial part of an algorithm that reconstructs the TSP from its degree sequence. Beyer *et al.* [2] presented a linear-time reconstruction algorithm for maximal outerplanar graphs. We present below a linear-time version, based on folding, that walks the perimeter of the TSP and, on the basis of the degree sequence, identifies all branching triangles; in addition, our algorithm (with a very small modification) builds the dual tree in linear time from the degree sequence.

Our algorithm is based on balancing; it starts at a vertex of degree two (corresponding a leaf in the dual tree) and walks the perimeter until it encounters another vertex of degree two, recording on the way (in terms of degrees) what it sees of the (maximal) fans to which the vertices belong. On reaching the second vertex of degree two, it starts matching the information it has accumulated with what it encounters on the other side of the perimeter. If no branching triangle exists, the TSP is a simple sleeve and the degrees predicted from the information gathered on one side must exactly match the actual degrees of the vertices on the other side. When a mismatch is found (a vertex with actual degree larger than its predicted degree), it indicates the presence of a branching triangle; by keeping a stack of the vertices on the first side and popping the stack as matches are made from the second side, we can identify the vertex on the first side that belongs to the branching triangle and thus reconstruct the TSP from its degree sequence.



	Degree Stack	Vertex Stack
before processing x	$[3, 3]$	$[y, y]$
after processing x	$[3, 4]$	$[y, y, x]$
after processing p_i	$[3, 5, 3, 3, 3]$	$[y, y, x, p_i, p_i, p_i, p_i]$
after processing p_{i+1}	$[3, 5, 4, 3]$	$[y, y, x, p_i, p_i, v_2, v_2]$

Figure 4: Reconstructing a TSP from its Degree Sequence

When used for reconstruction, our algorithm uses a vertex-stack and a degree-stack. Let s_k^d denote the kth entry from the top of the degree stack and s_k^v the kth entry from the top of the vertex stack; our algorithm proceeds as follows. Let p_0 be a vertex of degree 2; we initialize the degree stack by pushing a 2 on it and call the procedure illustrated below to work on vertex p_0. (Refer to the accompanying illustration, Figure 4.)

`Reconstruct`(p_{i-1})
Label the next vertex of the polygon p_i.
Set p_{i-1} boundary-adjacent to p_i.

($d_i = 3$) In this case, p_i (represented by vertex x in the figure) is on the base-path of some fan with center c_1 and s_1^d is the degree of c_1 in the sleeve p_0–p_i. Since p_i is diagonal-adjacent to c_1, we increment s_1^d by one and push a copy of p_i on the vertex-stack.

($d_i > 3$) In this case, p_i (p_i itself in the figure) is the center of a fan. Thus p_i is also an end-vertex of the base-path of some other fan, say centered at c_1, such that c_1 lies on the left-chain and is diagonal-adjacent to p_i. To reflect this adjacency, we increment s_1^d by one. Then we push $d_i - 3$ threes on the degree-stack to account for vertices $v_{k-1}, \ldots, v_1$ (the vertices on the base path of p_i, excluding c_1), that are expected to be of degree three. Finally, we push $d_i - 2$ copies of p_i on the vertex-stack to be later identified with the vertices of its base-path.

($d_i = 2$) In this case, p_i (represented by p_{i+1} in the figure) is the last vertex on the base path of some fan. Let k be the smallest natural number such that $d_{i+k} > s_k^d$; if such a vertex does not exist, let $k = n-i$. For $j = 1, \ldots, k$, we do the following: (i) we label the next vertex p_{i+j}; (ii) we make p_{i+j-1} and p_{i+j} boundary-adjacent; (iii) we make p_{i+j} diagonal-adjacent to $s_n^v, \ldots, s_{n'+1}^v$, where $n' = \sum_{t=1}^{j-1}(s_t^d - 2)$ and $n = n' + s_j^d - 2$; and finally, (iv) we discard entries $s_k^d, \ldots, s_1^d$ and $s_n^v, \ldots, s_1^v$. If the stacks are empty, we set p_n and p_0 boundary-adjacent and stop. Otherwise, we increment s_1^d to account for $\{p_{i+k}, q\}$, where q is the vertex whose degree is represented by s_1^d and then push $d_{i+k} - s_k^d - 1$ threes on the degree-stack, push $d_{i+k} - s_k^d$ copies of p_{i+k} on the vertex-stack, and call `Reconstruct`(p_{i+k}). Note that the triangle $p_{i+k}s_1^vq$, where q is the vertex whose estimated degree is on the top of the degree-stack before the adjustments, is a branching triangle. Thus the algorithm identifies a branching triangle as it visits its second vertex.

4 Generalization to k-Trees

We extend our coloring procedure for TSPs by showing that linear time also suffices to color the more general k-trees [6], for each fixed k. A $k-$tree is recursively defined as follows.

1. The complete graph on k vertices (a k-clique) is a $k-$tree.

2. If G is a $k-$tree on n vertices, we construct a $k-$tree on $n+1$ vertices by adding to G a vertex, v_{n+1}, and connecting it to all vertices of some k-clique in G. (By construction, G must contain at least one such clique.)

3. No other graph is a k-tree.

Hare *et al.* [5] have shown that a number of problems that are $\mathcal{NP}$-complete on general graphs become solvable in linear time on generalized series-parallel graphs, a family that includes k-trees, and gave some examples. We present a linear-time algorithm for coloring k-trees with the minimal number of colors. Our algorithm runs in time $O(k|V|)$—linear time for each fixed k—and colors the vertices of a $k-$tree using $k+1$ colors (with the exception of the trivial k-tree consisting of a single k-clique, which it colors with k colors only); since a $k-$tree has cliques of size $k+1$, the number of colors is minimal. Our algorithm takes advantage of the fact that any $k-$tree with at least $k+1$ vertices has at least two vertices of degree k. With each vertex, $v \in V$, we associate a list, L_v, that is originally empty and a field, $\deg(v)$, that contains the degree of v. Our algorithm, using a stack, orders the vertices of G so as to allow each vertex to be colored in time $O(k)$—constant for each fixed k. In the description below, the operation Pop( ) removes and returns the top entry of the stack, and AdjList(v) is the adjacency list of vertex v in the input graph G.

```
procedure Order (G = (V, E); stack; AdjList)
    stack ← φ
    Let A be the set of k + 1 tuples (v, v_1, ..., v_k) such that deg(v) = k
        and {v, v_i} ∈ E for 1 ≤ i ≤ k.
    while stack contains fewer than |V| − (k + 1) entries do
        Let (v, v_1, ..., v_k) be a member of A
        Set A ← A − {(v, v_1, ..., v_k)}
        Push ((v, v_1, ..., v_k))
        Set L_{v_i} ← L_{v_i} ∪ {v} for 1 ≤ i ≤ k
        Set deg(v_i) ← deg(v_i) − 1 for 1 ≤ i ≤ k
        for 1 ≤ i ≤ k, if deg(v_i) = k do
            Let {v'_1, ..., v'_k} = AdjList(v_i) − L_{v_i}
            Set A ← A ∪ {(v_i, v'_1, ..., v'_k)}
        endfor
    endwhile
    At this point, the k + 1 members of A form a complete graph on k + 1 vertices;
    color these k + 1 vertices using colors 1, ..., k + 1.
endprocedure

procedure Assignment (G = (V, E), stack)
    while stack is not empty do
        Set (v, v_1, ..., v_k) ← Pop( )
        Set Color(v) ← Σ_{i=1}^{k+1} i − Σ_{i=1}^{k} Color(v_i)
    endwhile
endprocedure
```

Let s_i denote the ith entry of the stack from the bottom and define $G_i = G - \{s_1, s_2, \ldots, s_i\}$ so that, in particular, $G_0 = G$. It is easy to see that s_i in G_{i-1} has degree k. Thus, if G_{i-1} already has a legal coloration, then giving to s_i the color that is not been used by its neighbors results in a legal coloration of G_i. The fact that a k-tree always has at least two vertices of degree k guarantees that G_i can always be constructed from G_{i-1}.

To see that the algorithm runs in linear time for each k, we need to explain how the set difference $\mathrm{Adjlist}(v_i) - L_{v_i}$ is computed in time proportional to the number of members in $\mathrm{AdjList}(v_i)$. We copy the members of $\mathrm{AdjList}(v_i)$ into a lazy array; then, for each member of L_{v_i}, we mark, in constant time, its corresponding copy in the lazy array as "deleted." Now the k unmarked members can be found by a scan of the lazy array. Since we perform such operation at most once for each vertex, the total time spent for difference is proportional to the number of edges in G; but $E \in O(k|V|)$ for a k-tree, proving our claim. (Reconstruction of a k-tree from a degree sequence would pose no particular problem, if it were not for the fact that such a graph is, in general, not Hamiltonian, thereby preventing us from defining a natural sequence of its vertices.)

5 Conclusion

We have introduced the idea of folding triangulated simple polygons, which has led us to some simple characterizations of such polygons and then to particularly simple formulations of algorithms to color or reconstruct such polygons from their degree sequence. The same coloring strategy can be extended to the family of k-trees, yielding a new, linear-time algorithm for finding a minimal coloring for these graphs. Thus the folding paradigm is seen as a useful approach to several problems in computational geometry and algorithmic graph theory.

References

[1] D. Avis and G. T. Toussaint, "An efficient algorithm for decomposing a polygon into star-shaped pieces," *Pattern Recognition* **13** (1981), 395–398.

[2] T. Beyer, W. Jones, and S. Mitchell, "Linear algorithms for isomorphism of maximal outerplanar graphs," *J. ACM* **26** (1979), 603–610.

[3] V. Chvàtal, "A combinatorial theorem in plane geometry," *J. Combin. Theory Series B* **18** (1975), 39–41.

[4] J. Fisk, "A short proof of Chvàtal's watchman theorem," *J. Combin. Theory Series A* **24** (1978), 374.

[5] E. Hare, S. Hedetniemi, R. Laskar, K. Peters, and T. Wimer, "Linear-time computability of combinatorial problems on generalized series-parallel graphs," in *Discrete Algorithms and Complexity*, D.S. Johnson, T. Nishizeki, A. Nozaki, and H.S. Wilf, eds., Academic Press, New York (1986), 437–457.

[6] D.J. Rose, "On simple characterizations of k-trees," *Discrete Mathematics* **7** (1974), 317–322.