The Area Partitioning Problem

Hannah Bast & Susan Hert

{hannah,hert}@mpi-sb.mpg.de
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, GERMANY

Abstract

Given an arbitrary polygon with n vertices, we wish to partition it into p connected pieces of given areas. The problem is motivated by a robotics application in which the polygon is a workspace that is to be divided among p robots performing a terrain-covering task. We show that finding an area partitioning with minimal cut length is NP-hard in the number of pieces and that it is even hard to approximate to within any factor that is independent of the shape of the polygon. We then present a simple $O(pn)$-time algorithm that produces non-optimal, but often quite reasonable, area partitionings for arbitrary polygons.

1 Introduction

We consider here the problem of partitioning a simple polygon into a given number of smaller connected polygons, each of a given area. This problem, which we call the *area partitioning problem*, has applications in robotics, where the polygon to be partitioned is a workspace for a number of robots that must perform a terrain-covering task such as cleaning, lawn mowing, or milling [1, 10].The workspace is to be partitioned in order to divide the work among the robots.

For terrain-covering tasks, the goal is to ensure that each point of the workspace is visited at least once. For such tasks, the area of the workspace provides a reasonable estimate for the work to be done. Thus, dividing a polygonal workspace into p connected, equal-area pieces provides a reasonably balanced division of labor for p robots of equal abilities. When the robots are of different abilities (*e.g.*, different sizes or different speeds), a division into polygons with different areas is obviously called for. Though most other works that have considered the area partitioning problem consider specifically the case of producing a cut that bisects the area of the polygon [2, 17] (or simultaneously bisects the area of two disjoint polygons [4, 18]), the methods generalize for producing partitions of other proportions as well. However, the methods that are formulated for convex polygons [3, 4, 8, 18] do not generalize for nonconvex polygons and the methods developed to handle nonconvex polygons or polygons with holes [2, 17] do not guarantee that the cuts will result in connected polygons. This is obviously an important consideration for the terrain-covering application. Covering many disconnected pieces of the workspace generally requires more work than covering a single connected piece since the robot must travel among the disconnected pieces.

An ideal partitioning algorithm for this application would create polygons that are optimal for the particular terrain-covering algorithm to be used by the robots, where "optimal" means that the robots must do the least amount of work for a given region. However, a quantification of precise criteria that would lead to these optimal results is difficult to develop due to the dependence on the algorithm the robots will implement. A reasonable criterion that is independent of the robots' algorithm is the minimization of the total cut length used to create the partition. This is a criterion that has also been used in previous works on polygon partitioning (*e.g.*, [15]) and makes the partitioning problem NP-hard in many cases [14, 16]. We show here in Section 2 that this criterion also makes our area partitioning problem NP-hard, even when the cut length is allowed to deviate from the optimum by an arbitrary factor that is independent of the shape of the polygon.

Following this, in Section 3 we present a simple, linear-time algorithm for solving the general area partitioning problem. Though formulated for simply connected polygons, the algorithm can easily be applied to nonsimply connected polygons when each hole is removed by joining it to the outer boundary of the polygon. The algorithm is similar to the one presented in [9], which was designed to handle the additional requirement that each partition polygon contain a particular point. By removing this requirement, we are able to employ new heuristics at certain crucial points in the algorithm to produce partitions that in many cases appear quite natural. The algorithm requires $O(pn)$ time to divide a polygon with n vertices into p pieces, compared to $O(p^2n)$ time required for the previous algorithm. This is the only algorithm we are aware of for solving the general area partitioning problem that guarantees connected partition polygons.

The algorithm has been implemented using the CGAL software library [5]. In Section 4, we show a number of examples of the partitions produced by the algorithm and discuss the results and some open problems.

2 Hardness results

Koutsoupias *et al.* [14] showed that finding a minimum-cut partitioning of a polygon with n vertices into $p = 2$ connected, equal-area pieces can be done in $O(n^2)$ time. While their algorithm can be easily modified to produce partitionings for arbitrary areas, it is not at all clear what an extension that can cope with an arbitrary number of pieces would look like. As the following result shows, this general case is much harder. In particular, a simple divide-and-conquer approach, which would lead to a running time polynomial in p, is inappropriate, even if we are content with approximately optimal solutions. It should be noted that, while the partitioning algorithm presented in Section 3 exclusively employs cuts consisting of straight line segments, the theorem below makes no assumptions whatsoever on the nature of the cuts. For that general case, let us agree to call a cut a minimal connected segment of the boundary of a piece of the partitioning whose endpoints lie on the input polygon.

Theorem 2.1 *For every $\varepsilon \geq 0$, the problem of computing an area partitioning of a given (rectilinear) polygon into a given number of pieces with given areas such that the cut length is within $1+\varepsilon$ of the optimum is NP-hard.*

Proof: The proof is done by a reduction from the following PARTITION problem, which is well-known to be NP-complete [6]: Given p integers $a_1, \ldots, a_p$, decide whether there exists $I \subseteq \{1, \ldots, p\}$ with $\sum_{i \in I} a_i = \sum_{i \in \{1,\ldots,p\} \setminus I} a_i$.

Let $\varepsilon \geq 0$, and consider the following procedure for solving instances of PARTITION by means of an area partitioning algorithm. Given $a_1, \ldots, a_p$, first construct the polygon $\mathcal{P}$ shown in Figure 1(a), where $c = \lceil p \cdot (1+\varepsilon) \rceil$, and $a_{\max} = \max\{a_1, \ldots, a_p\}$. Note that the area of $\mathcal{P}$ is just $c^2 \cdot A$, where $A = \sum_{i=1}^{p} a_i$. Now compute a partitioning of $\mathcal{P}$ into p pieces of areas $c^2 \cdot a_1, \ldots, c^2 \cdot a_p$, determine the total cut length of this partitioning, and output YES if it is less than c and NO otherwise. We will prove in what follows that this always gives the correct answer, provided that the

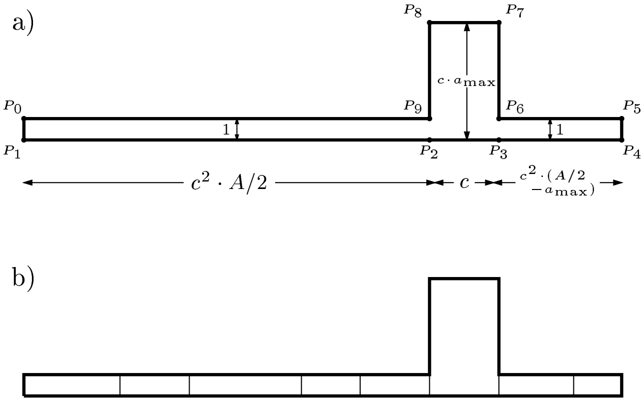

Figure 1: The polygon $\mathcal{P}$, and a partitioning via cuts of length 1.

area partitioning algorithm employed guarantees a total cut length that is at most a factor of $1+\varepsilon$ above the optimum. Since polygon $\mathcal{P}$ can be constructed from $a_1, \ldots, a_p$ in linear time, the claimed NP-hardness result then immediately follows by the NP-hardness of PARTITION.

For the easy part, assume that $a_1, \ldots, a_p$ can be split into two parts of equal sum, *i.e.*, there exists $I \subseteq \{1, \ldots, p\}$ with $\sum_{i \in I} a_i = A/2$. As shown in Figure 1(b), polygon $\mathcal{P}$ can then be partitioned via $p-1$ vertical cuts of length 1, each going either from $\overline{P_1P_2}$ to $\overline{P_0P_9}$ or from $\overline{P_3P_4}$ to $\overline{P_5P_6}$. The total length of these cuts is $p-1$, so any area partitioning algorithm that guarantees a total cut length within $1+\varepsilon$ of the optimum, will yield a partitioning with total cut length at most $(p-1) \cdot (1+\varepsilon) < c$. Our decision procedure therefore answers YES, as it should.

Now coming to the hard part of the proof, assume that our decision procedure has answered YES; that is, a partitioning of polygon $\mathcal{P}$ with total cut length less than c was computed. Certainly then, each individual cut has length less than c, and the remainder of the proof is dedicated to showing that under this condition $a_1, \ldots, a_p$ can be partitioned into two subsets of equal sum.

We start with the observation that a cut (of whatever shape) of length at most c with endpoints on either the same edge or two adjacent edges of an arbitrary rectilinear[1]polygon can cut off an area of at most c^2/π from that polygon; this follows by a simple isoperimetric argument (see, for example, [12]). Since each piece in our partitioning of $\mathcal{P}$ has area at least c^2, this implies that all cuts must be from the edge $\overline{P_1P_4}$ to one of the edges $\overline{P_0P_9}$, $\overline{P_9P_8}$, $\overline{P_6P_7}$, or $\overline{P_6P_5}$ (all other possible pairs of edges are either adjacent or at least at distance c from

[1] In fact, a similar statement holds for any polygon with minimal interior angle bounded away from zero.

each other[2]).

Now, if no cut had an endpoint lying on the segment $\overline{P_2P_3}$, then no cut would intersect the rectangle $P_2P_3P_7P_8$. But this is not possible, since this rectangle alone has area $c^2 \cdot a_{\max}$, which is maximal among the given areas. We may therefore conclude that there is at least one cut with an endpoint lying on the segment $\overline{P_2P_3}$. Of all cuts with this property, consider the cut $\mathcal{C}$ whose endpoints are closest to P_1, and denote by C_L and C_U its lower (*i.e.*, lying on $\overline{P_2P_3}$) and upper (*i.e.*, lying on $P_0P_9P_8$ or $P_7P_6P_5$) endpoints, respectively.

Now if C_U lies on $P_0P_9P_8$, an isoperimetric argument of the same kind as used above shows that the part of polygon $\mathcal{P}$ to the left of cut $\mathcal{C}$ deviates from $c^2 \cdot A/2$ (which is just the area of the rectangle $P_1P_2P_9P_0$) by no more than c^2/π. Consequently, there exists I with $\left|\sum_{i\in I} c^2 a_i - c^2 \cdot A/2\right| < c^2/2$, and hence, since twice the term on the left hand side is divisible by c^2, $\left|\sum_{i\in I} c^2 a_i - c^2 \cdot A/2\right| = 0$.

Similarly, if C_U lies on $P_7P_6P_5$, the area of that part of polygon $\mathcal{P}$ which lies to the right of cut $\mathcal{C}$ deviates from $c^2 \cdot A/2 - c^2 \cdot a_{\max}$ (which is just the area of the rectangle $P_3P_4P_5P_6$) by no more than c^2/π. Since, by the choice of $\mathcal{C}$, there is no further cut to the left of $\mathcal{C}$ intersecting the rectangle $P_2P_3P_7P_8$, we also know that the area of that piece of the partitioning lying directly left of $\mathcal{C}$ deviates from $c^2 \cdot a_{\max}$ by no more than c^2/π, so that it must in fact be of area $c^2 \cdot a_{\max}$ (since all areas are divisible by c^2). Therefore, the total area of this piece together with all pieces to the right of $\mathcal{C}$ deviates by at most c^2/π from $c^2 \cdot A/2$, which, just as above, leaves only the possibility that it is exactly $c^2 \cdot A/2$.

We have thus shown that whenever our decision procedure answers YES, we can construct a partitioning of $a_1, \ldots, a_p$ into two parts of equal sum. Since the reverse implication was already shown to be true, this finishes the proof of Theorem 2.1. ■

The very same construction used in the proof above also shows that area partitioning with minimum cut length is NP-hard even for the seemingly simpler case when the input polygons are restricted to be rectilinear and the cuts are required to be straight-line orthogonal cuts. However, it must also be noted that our proof hinges on the fact that $\mathcal{P}$ is a nonconvex polygon constructed from very thin pieces, *i.e.*, in a sense $\mathcal{P}$ is not very compact. The interesting question of whether an efficient area-partitioning algorithm exists that produces optimal or close-to-optimal cut lengths when the compactness of the input polygons is known to be bounded (in some intuitive sense) thus remains open.

[2]Strictly speaking, this is true only when $a_{\max} > 1$, which however can be assumed without loss of generality.

3 An Algorithm

In this section we present a modified version of the algorithm described in [9]. The original algorithm was designed to produce an anchored area partition of a given polygon, meaning that each partition polygon contained a distinct given point on its boundary. By removing this restriction, we leave room for the use of data structures and heuristics that improve the resulting partition as well as the running time of the algorithm.

To produce an area partition of a nonconvex polygon, we begin with a convex decomposition of the polygon that does not introduce Steiner points [7, 11, 13, 19] and use a sweep-line method similar to the one described in [9] and [18] to divide the convex pieces as necessary. In an attempt to capitalize on the work already done in computing the convex decomposition, we construct the partition polygons by first simply merging adjacent convex pieces. Only when the merging of another piece would result in a polygon that was too large is the sweep-line algorithm employed to divide one or more of the convex pieces. The main difficulty with this sweep is that, as a line is swept through one of the convex pieces, edges to which other convex pieces are attached may be encountered. Care must be taken to ensure that the remaining, unpartitioned part of the polygon does not become disconnected when a portion of one of the convex pieces is carved out to become part of one of the partition polygons. If the unpartitioned polygon becomes disconnected, it may then be impossible to create the remaining partition polygons.

For both the merging phase and the sweep-line phase of the algorithm, simple heuristics are employed with the goal in mind of reducing the number of convex pieces that must be divided, thereby reducing the work to be done and producing partition polygons that more closely resemble the original polygon. The notation we use for the description of the algorithm is presented in Section 3.1. Section 3.2 describes the merging phase of the algorithm and the heuristics employed there, and Section 3.3 describes the sweep-line algorithm and its heuristics.

3.1 Notation

The input to our algorithm is a simply connected polygon $\mathcal{P}$ with area A. We wish to partition it into p (nonoverlapping) pieces with areas equal to a given set of values $a_1, \ldots, a_p$, $\Sigma_{i=1}^{p} a_i = A$. The resulting partition polygon with area a_i is denoted $\mathcal{P}_i$.

The polygon $\mathcal{P}$ is assumed to have been decomposed into a collection of convex polygons without Steiner points. This can be done in time polynomial in the number of vertices of $\mathcal{P}$ [7, 11, 13, 19]. Let $\mathcal{CP}_j, j = 1, \ldots, q$ denote the set of nonoverlapping convex polygons (*pieces*) that result from the convex decomposition. (When $\mathcal{P}$ is convex, $\mathcal{P} = \mathcal{CP}_1$.) Any two pieces $\mathcal{CP}_j$ and $\mathcal{CP}_k$ are *neighbors* if they share an edge. This convex decomposition of $\mathcal{P}$ induces a graph in which the nodes correspond to the convex pieces and there are edges between neighboring convex pieces. Since we begin with a convex decomposition that does not introduce any Steiner vertices and have assumed that $\mathcal{P}$ is simply connected, the connectivity graph induced by the convex decomposition is a tree T. With each node N_j in T, we associate a value $Area(N_j)$, which is the total area of all the convex pieces associated with the nodes in the subtree rooted at N_j. For the sake of brevity, we often refer in the sequel to a node N_j and the convex piece $\mathcal{CP}_j$ associated with that node interchangeably. In particular, we use the term *parent of piece* $\mathcal{CP}_j$ to refer to the convex piece associated with the parent of node N_j in the tree. The vertices of polygons $\mathcal{P}$ and $\mathcal{CP}_j, j = 1, \ldots, q$ are assumed to be in counterclockwise (CCW) order.

In general, the polygons $\mathcal{P}_i$ of an area partition will be nonconvex. We construct each polygon $\mathcal{P}_i$ incrementally as a union of smaller nonoverlapping convex polygons. Each smaller polygon constructed as part of $\mathcal{P}_i$ is said to be *assigned* to $\mathcal{P}_i$. When necessary, a sweep line L is used to divide convex pieces to carve out polygons of the correct areas to assign to a polygon $\mathcal{P}_i$. Generally, the sweep line is based at a vertex of a particular convex piece and then swept CCW around that piece. The sweep line is assumed to be oriented from its base point to the moving point.

3.2 Merging Convex Pieces

An optimal strategy for merging convex pieces to form the partition polygons would result in the fewest number of convex pieces being divided. While this strategy may be optimal in terms of the amount of additional partitioning work to be done, we conjecture that this strategy would likely require that all possible groupings be considered in order to find the optimal solution.

Conjecture 3.1 *Given a set of polygons $\mathcal{CP}$ that form a convex partition of a polygon $\mathcal{P}$, determining which connected sets of these polygons to merge for the purpose of creating an area partition of $\mathcal{P}$ that requires the fewest number of convex pieces to be split is NP-hard.*

Such an exhaustive search technique would lead to an impractically slow partitioning algorithm. We therefore employ a simple heuristic strategy to select the pieces to be merged.

The strategy is based on the construction of a connectivity graph T corresponding to the convex decomposition. Recall that each node N_j in T has a value $Area(N_j)$ associated with it, which is the total area of all the convex pieces associated with the nodes in the subtree rooted at N_j. For any value of a_i, the convex pieces to merge to create a partition polygon $\mathcal{P}_i$ are found by searching through the nodes in T. We search for the largest node N in the tree with $Area(N) \leq a_i$. The nodes in the subtree rooted at N correspond to the convex pieces to be merged. These nodes are then removed from the tree T. If $Area(N)$ is strictly less than a_i or there is no node N for which $Area(N) \leq a_i$, then one or more of the convex pieces must be divided in order to create the remainder of the polygon with area a_i. The sweep-line procedure described in the following section is used for this purpose.

3.3 Dividing Convex Pieces

Though the goal of dividing a convex piece of a nonconvex polygon into pieces of given areas that are to become parts of separate partition polygons sounds similar to the goal of dividing a convex polygon into polygons with given areas, the sweep-line procedure described here differs significantly from the ones for dividing convex polygons described in [9] and [18]. This is due to the two additional constraints that may be present during this second phase of the algorithm:

C1 When pieces are carved out of a convex piece $\mathcal{CP}$, the parts of the polygon $\mathcal{P}$ not yet assigned to a partition polygon must remain connected.

C2 If the piece carved out of $\mathcal{CP}$ is to be merged with other convex pieces in order to create an entire partition polygon, then this piece must be connected to the convex pieces with which it is to be merged.

We can see that the choice of where to start the sweeping process is dictated by these constraints and it is possible that the division of more than one convex piece is required to accumulate enough area for a partition polygon $\mathcal{P}_i$.

The algorithm described here is a modification of the algorithm presented in [9]. The main difference is that we are somewhat more greedy in our sweeping procedure and have fewer constraints on the placements of the endpoints of the sweep line since the partition polygons are not required to contain any specific points, as was the case in [9].

The procedure that implements the algorithm is called *DivideConvexPiece* and there are two situations in which it is called. In the first situation, a_i is smaller than the smallest leaf of the current tree T. In this case, this smallest leaf is the one that is chosen to be divided. Here, constraint C2 does not exist, and, since the convex piece to be divided, $\mathcal{CP}_j$, corresponds to a leaf in T, there is only a single edge that is shared with another convex piece (the parent of $\mathcal{CP}_j$). It is guaranteed to be possible to create the polygon of area

a_i from only this piece. So the only concern is that constraint C1 must be satisfied when the polygon with area a_i is cordoned off. This is assured by simply basing the sweep line at the appropriate endpoint of the edge shared by $\mathcal{CP}_j$ and its parent. When the line is swept in a CCW direction, the appropriate endpoint is the one that comes second in the CCW ordering of the vertices. The sweep stops when the area to the right of the sweep segment, which is directed from its base point to the moving point, is equal to a_i.

In the second situation in which $DivideConvexPiece$ is called, a portion of partition polygon $\mathcal{P}_i$ has been created by merging some set of convex pieces. Assume that node N_j is the node corresponding to the root of the subtree of merged pieces and let N_k denote its parent in the tree.[3] The remaining part of polygon $\mathcal{P}_i$, with area a'_i, must then be constructed by using some portion of the convex piece $\mathcal{CP}_k$ corresponding to N_k and perhaps other adjacent pieces.

The sweep begins in $\mathcal{CP}_k$ with sweep line L based at the endpoint of the edge shared by $\mathcal{CP}_j$ and $\mathcal{CP}_k$ that comes first in the CCW ordering of the vertices. The sweep proceeds CCW. This assures that the piece carved out of $\mathcal{CP}_k$ for $\mathcal{P}_i$ will be connected to the convex pieces already designated as part of $\mathcal{P}_i$; *i.e.*, constraint C2 is satisfied. The CCW sweep stops when either

(a) the portion of the convex piece to the right of the sweep line has area equal to a'_i or

(b) the endpoint of an edge e_{ccw} that is shared with another convex piece is encountered.

In the first case, the polygon $\mathcal{P}_i$ has been fully constructed, so the data structures are updated to reflect the removal of this polygon and then the construction of the next polygon P_{i+1} begins. In the second case, however, more work must be done.

Let $\mathcal{CP}_{ccw}$ denote the convex piece that shares edge e_{ccw} with $\mathcal{CP}_k$ and let N_{ccw} denote the corresponding node in T. There are three general possibilities to be considered here: either all, some, or none of $\mathcal{CP}_{ccw}$ will be used for polygon $\mathcal{P}_i$. However, before considering whether $\mathcal{CP}_{ccw}$ needs to be divided or not, we reverse the sweeping process by anchoring the sweep line at the first endpoint of e_{ccw} and moving the other endpoint of L CW around $\mathcal{CP}_k$. This is done in the hope that deferring the partitioning of convex pieces will result in fewer pieces being divided.

Though the sweep direction is reversed, the sweep line maintains its original orientation, and the sweep in the CW direction stops in the same two cases that the CCW sweep stopped. Again, in the first case, polygon $\mathcal{P}_i$ is finished, while in the second case more work must be done. Now the work to be done includes deciding how much, if any, of the convex pieces adjacent to $\mathcal{CP}_k$ should be assigned to $\mathcal{P}_i$.

[3]Note that the parent node must exist if the problem has been well stated. Otherwise N_j is the root node of T and $\mathcal{P}_i$ therefore has already been assigned all of the remaining pieces, which implies that it is impossible to create a polygon with area a_i.

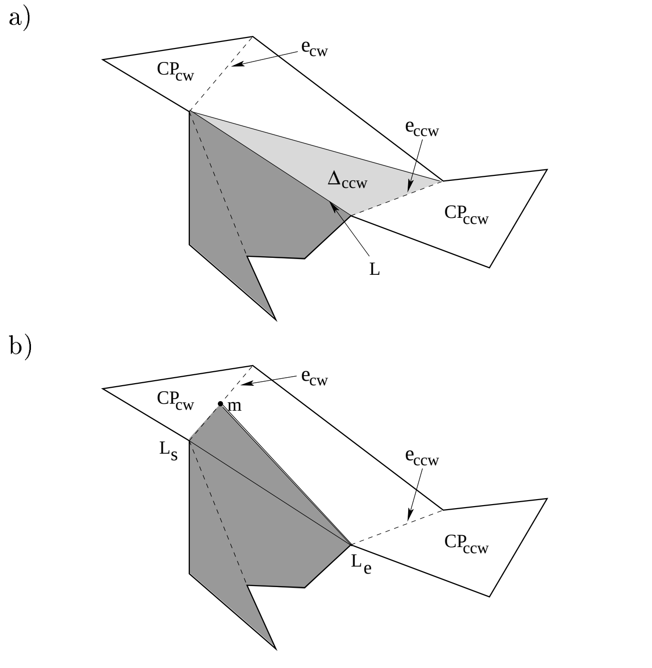

Figure 2: A partition polygon under construction. The dark grey area has already been assigned to the polygon. (a) The light grey triangle is Δ_{ccw}. (b) Midpoint m of edge e_{cw} is computed to create the triangle L_sL_em, which is assigned to the polygon before recursively dividing CP_{cw}.

Let Δ_{ccw} be the triangle defined by the endpoints of the current sweep line and e_{ccw} (Figure 2(a)). Define e_{cw}, $\mathcal{CP}_{cw}$, N_{cw} and Δ_{cw} analogous to e_{ccw}, $\mathcal{CP}_{ccw}$ N_{ccw} and e_{ccw}. Further set a''_i equal to a'_i minus the area of $\mathcal{CP}_k$ to the right of the current sweep line. This is the area that remains to be accumulated. It may be possible that this area can be accumulated without dividing either $\mathcal{CP}_{ccw}$ or $\mathcal{CP}_{cw}$. This is true if either of the triangles Δ_{ccw} or Δ_{cw} has area strictly greater than a''_i or the area associated with the subtree rooted at either N_{ccw} or N_{cw} is strictly less than a''_i.

We consider these four cases, which are not mutually exclusive, in the following order:

1. $Area(\Delta_{ccw}) > a''_i$
2. $Area(\Delta_{cw}) > a''_i$
3. N_{ccw} is not the parent of N_k and $Area(N_{ccw}) < a''_i$
4. N_{cw} is not the parent of N_k and $Area(N_{cw}) < a''_i$

The first and second cases are handled analogously, so we consider only the first here. In this case, we interpolate along edge e_{ccw} to find the point where the area of the triangle defined by this point and the sweep line is equal to a''_i. Since the area of Δ_{ccw} is strictly greater than a''_i (and since a''_i is presumably not equal to 0 at this point), this interpolation point does not coincide with an endpoint of e_{ccw}. This is important, because otherwise the removal of the interpolated triangle to assign to $\mathcal{P}_i$ would disconnect $\mathcal{CP}_{ccw}$ from the rest of the unpartitioned polygon.

The latter two cases are also handled analogously, so we consider only the third. In this case, the sweep simply continues in the CCW direction after assigning the pieces in the subtree rooted at N_{ccw} to $\mathcal{P}_i$.

When none of these four cases is satisfied, we must divide either $\mathcal{CP}_{ccw}$ or $\mathcal{CP}_{cw}$. The division is done by calling *DivideConvexPiece* recursively on one of these two pieces. We choose to divide the piece for which the associated node's subtree has area closer to a''_i in the hope that it will result in fewer pieces being divided since there is a better chance that the pieces connected to the chosen piece can be assigned to $\mathcal{P}_i$ in their entirity as the recursive division progresses. If one of these nodes corresponds to the parent of N_k then the other is chosen without a comparison of the subtrees' areas and a''_i. The area requirement associated with the recursive call is set to a''_i less the area of a triangle that connects the chosen convex piece to the part of polygon $\mathcal{P}_i$ already constructed. This triangle is defined by the current sweep line L and a point on the edge e_{ccw} or e_{cw}, as appropriate. In an attempt to avoid polygons that are close to degenerate, we choose the third vertex of the triangle as the midpoint of either e_{ccw} or e_{cw} (Figure 2(b)).

The procedure *DivideConvexPiece* is summarized below.

DivideConvexPiece$(N_j, a_i, \mathcal{P}_i)$

Input : N_j root node of the subtree to be assigned to polygon $\mathcal{P}_i$; a_i area required for $\mathcal{P}_i$; $\mathcal{P}_i$ the polygon created so far

Output : partition polygon $\mathcal{P}_i$

Procedure :

```
if N_j is a leaf and a_i < Area(N_j) then
  Select the second vertex shared by N_j and its
      parent as the base point of a sweep line.
  Rotate the sweep line CCW around CP_j until
      area to the right of the sweep line equals a_i.
else
  a'_i = a_i − Area(N_j)
  while (a'_i > 0) do
      N_k = parent of N_j
      L_s = L_e = first vertex shared by N_k and N_j
      CCW_Sweep_to_Neighbor(N_k, a'_i, L, P_i)
      if (Area(P_i) = a_i) return

      CW_Sweep_to_Neighbor(N_k, a'_i, L, P_i)
      if (Area(P_i) = a_i) return

      a'_i = a'_i − area to the right of L in N_k
      if Area(Δ_ccw) > a'_i
        pt = exact area interpolation point on e_ccw
        assign triangle (L_s, L_e, pt) to P_i
        return
      else if Area(Δ_cw) > a'_i
        pt = exact area interpolation point on e_cw
        assign triangle (L_s, L_e, pt) to P_i
        return
      else if N_ccw is not the parent of N_k and
              Area(N_ccw) < a'_i
        assign nodes in subtree rooted at N_ccw to P_i
        a'_i = a'_i − Area(N_ccw)
      else if N_cw is not the parent of N_k and
              Area(N_cw) < a'_i
        assign nodes in subtree rooted at N_cw to P_i
        a'_i = a'_i − Area(N_cw)
      else
        if N_ccw is the parent of N_k
          N = N_cw
        else if N_cw is the parent of N_k
          N = N_ccw
        else if (Area(N_ccw) − a'_i) < (Area(N_cw) − a'_i)
          N = N_ccw
        else
          N = N_cw
        endif
        midPt = midpoint of edge btwn N_k and N
        a'_i = a'_i - area of triangle (L_s, L_e, midPt)
        DivideConvexPiece(N, a'_i, P_i)
        return
      endif
  enddo
endif
```

The procedures *CCW_Sweep_to_Neighbor* and *CW_Sweep_to_Neighbor* are variants of the simple sweep-line procedure described in [9, 18] for dividing a convex polygon into pieces of given area. An endpoint of the segment L is swept either clockwise or counterclockwise around the convex piece $\mathcal{CP}_k$ associated with N_k until either the area to the right of L is equal to a'_i or a vertex that is shared with a neighboring convex piece is encountered. As the sweep proceeds, the polygon $\mathcal{P}_i$ is updated to include the vertices of $\mathcal{CP}_k$ that are to the right of the sweep line.

3.4 Runtime Analysis

The correctness of the algorithm follows from the same arguments given in [9]. Here we show that the general-purpose algorithm presented above achieves running time of $O(pn)$, compared to the $O(p^2n)$ time required by the previous algorithm.

Theorem 3.1 *Given a polygon with n vertices, together with a decomposition into q convex pieces, and given p areas, the area partitioning algorithm described in Sections 3.2 and 3.3 requires $O(n + pq) = O(pn)$ time.*

Proof: In the preprocessing phase of the algorithm, the connectivity tree T must be built. This can be done via a simple depth-first or breadth-first traversal of the pieces. It is reasonable to assume that the output of the convex decomposition algorithm includes information about which pieces neighbor which others (*e.g.*, for each edge shared between two pieces, a pointer to the neighboring piece). Thus the traversal can be accomplished in time linear in the total number of edges of the decomposition by simply walking around the edges and following neighbor pointers where they exist. In any convex decomposition of a polygon with n vertices into q pieces that does not use Steiner points, there are $n + q - 1$ edges. Since $q = O(n)$, this means the tree T can be built in $O(n)$ time. (When the connectivity information is not provided as part of the output of the convex decomposition, the preprocessing time increases to $O(n^2)$ since it is necessary to compare each edge to every other edge in the decomposition.)

Once the tree has been built, it is used to identify the convex piece(s) that will serve as the basis for each polygon $\mathcal{P}_i$. Since T has at most q nodes, finding the node corresponding to this basis can be done in $O(q)$ time. Once the proper node N has been identified, then in the worst case the nodes in the subtree rooted at N must be removed from the tree T, and the area values associated with the ancestors of N in T must be updated to reflect the deletion. Since removing a single node from T requires $O(1)$ time, in total these removals require at most $O(q)$ time. Updating the area value for a single node in T can be done in constant time, and since each of the q nodes will be updated at most once for each of the partition polygons, the total time required for updating is $O(pq)$. Thus, for all p pieces, the time required to identify the correct basis pieces for the partition polygons and update the data structure is $O(pq)$.

To compute a bound on the time required to construct the partition polygons once these basis pieces have been determined, we note that the merging and sweeping processes that add vertices to a polygon $\mathcal{P}_i$ require time linear in the number of vertices added. Each vertex that is considered for inclusion in a polygon is either immediately included or never considered for inclusion again. Thus, the only vertices that may be considered more than once are those that are endpoints of edges on which Steiner points are computed. For each Steiner point created, one endpoint will be considered that is not added to $\mathcal{P}_i$, so the extra visit to this endpoint can be charged to the Steiner vertex created. Thus the amount of work done to produce the partition is linear in the total number of vertices in the partition. Since each partition polygon created can add at most one Steiner vertex in each of the q convex pieces, this is at most $O(n + pq)$.

Thus, producing the cuts requires at most $O(n + pq)$ time and maintaining and searching the connectivity tree requires at most $O(pq)$ time. This results in a total time complexity of $O(n + pq) = O(pn)$, since $q = O(n)$. ∎

4 Examples and Discussion

The algorithm described in this paper has been implemented using the CGAL software library [5]. Figures 3 and 4 show a set of examples of the partitions produced by the algorithm.

For all examples, the results were produced beginning with an optimal convex decomposition where the optimality criterion was the number of convex pieces [7]. This criterion was chosen for the convex decomposition instead of the minimum cut length because we have an interest in producing partitions that not only have small cut length but also seem somewhat natural for the robotics application. In particular, we do not want to cut narrow corridors in the polygon into even narrower ones. The first panel in Figure 4 shows that this sometimes happens anyway, but it is more likely to occur if the convex decomposition is more likely to be created using short cuts across the narrow corridors. Using a smaller number of convex pieces also has the advantage that there will generally be fewer recursive calls required to produce a partition and thus less work to do, especially if p is significantly larger than the number of convex pieces.

Whatever the convex decomposition used, we see by these examples, as one should expect, that it greatly influences the partition produced. In particular, it can lead to Steiner vertices in the middle of the polygon that seem somewhat arbitrary. However, the convex decomposition coupled with the heuristic search used in the merging phase of the algorithm to find the best-fitting set of pieces often lead to relatively compact partition polygons. Thus an interesting question for further investigation is whether an algorithm can be developed that does not require this preprocessing phase but produces polygons that are comparably compact. We would also like to further investigate the problem

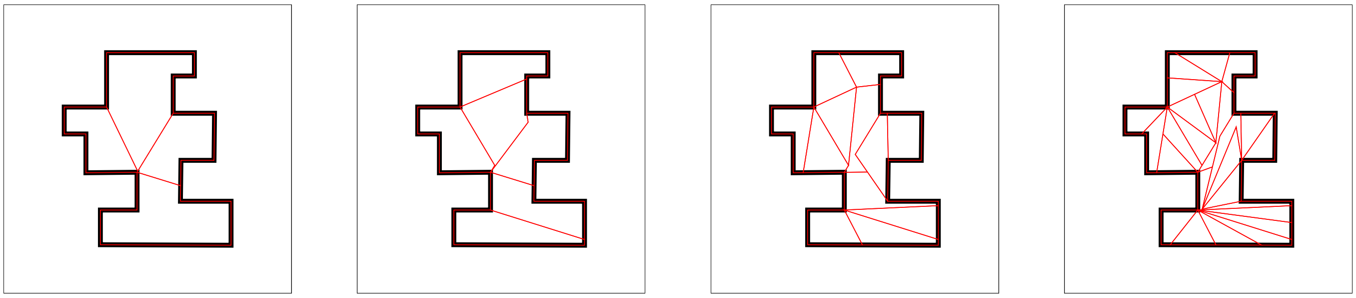

Figure 3: A rectilinear polygon with $n = 20$ vertices cut into $p = 3$, 6, 12, and 24 equal-area pieces.

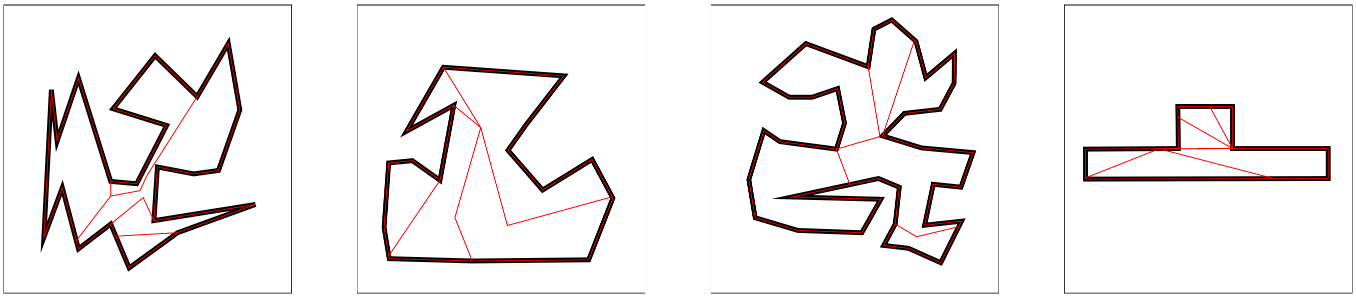

Figure 4: Four different polygons, each cut into 5 pieces of different areas.

of identifying a class of nonconvex polygons for which exactly or approximately optimal area partitions can be computed in polynomial time.

References

[1] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. Technical report, Mathematisches Institut, Universität zu Köln, 1997.

[2] K. F. Böhringer, B. R. Donald, and D. Halperin. On the area bisectors of a polygon. *Discrete Computational Geometery*, 22:269–285, 1999.

[3] P. Bose, J. Czyzowicz, E. Kranakis, D. Krizanc, and D. Lessard. Near-optimal partitioning of rectangles and prisms. In *Proceedings 11th Canadian Conference on Computational Geometry*, pages 162–165, 1999.

[4] M. Díaz and J. O'Rourke. Ham-sandwich sectioning of polygons. In *Proceedings 2nd Canadian Conference on Computational Geometry*, pages 282–286, 1991.

[5] GALIA consortium. `http://www.cgal.org/`.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[7] D. H. Greene. The decomposition of polygons into convex parts. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Advances in Computing Research*, pages 235–259. JAI Press, London, England, 1983.

[8] R. Guàrdia and F. Hurtado. On the equipartitions of convex bodies and convex polygons. In *Proceedings 16th European Workshop on Computational Geometry*, pages 47–50, 2000.

[9] S. Hert and V. Lumelsky. Polygon area decomposition for multiple-robot workspace division. *International Journal of Computational Geometry and Applications*, 8(4):437–466, 1998.

[10] S. Hert, S. Tiwari, and V. Lumelsky. A terrain-covering algorithm for an AUV. *Journal of Autonomous Robots*, 3:91 – 119, 1996.

[11] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proceedings of Conference on Foundations of Computation Theory*, pages 207–218, New York, 1983. Springer-Verlag.

[12] C. Isenberg. The maximum area contained by a given circumference. *The Science of Soap Films and Soap Bubbles*, pages 171–173, 1992.

[13] J. M. Keil. Decomposing a polygon into simpler components. *SIAM Journal on Computing*, 14:799–817, 1985.

[14] E. Koutsoupias, C. H. Papadimitriou, and M. Sideri. On the optimal bisection of a polygon. *ORSA Journal on Computing*, 4(4):435–438, 1992.

[15] C. Levcopoulos and A. Lingas. Bounds on the length of convex partitions of polygons. In *Proceedings 4th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 181 of *Lecture Notes in Computer Science*, pages 279–295. Springer-Verlag, 1984.

[16] A. Lingas, R. Pinter, R. Rivest, and A. Shamir. Minimum edge length partitioning of rectilinear polygons. In *Proc. 20th Allerton Conf. Commun. Control Comput.*, pages 53–63, 1982.

[17] T. C. Shermer. A linear time algorithm for bisecting a polygon. *Information Processing Letters*, 41:135–140, 1992.

[18] I. Stojmenović. Bisections and ham-sandwich cuts of convex polygons and polyhedra. *Information Processing Letters*, 38:15–21, 1991.

[19] S. B. Tor and A. E. Middleditch. Convex decomposition of simple polygons. *ACM Transactions on Graphics*, 3:244–265, 1984.