DECOMPOSING A POLYGON INTO ITS CONVEX PARTS*

Bernard Chazelle
Department of Computer Science
Yale University
New Haven, Connecticut

David Dobkin
Department of Computer Science
University of Arizona
Tucson, Arizona

ABSTRACT

A common operation in geometric computing is the decomposition of complex structures into more basic structures. Since it is easier to apply most algorithms to triangles or arbitrary convex polygons, there is considerable interest in finding fast algorithms for such decompositions. We consider the problem of decomposing a simple (non-convex) polygon into the union of a minimal number of convex polygons. Although the structure of the problem led to the conjecture that it was NP-complete, we have been able to reach polynomial time bounded algorithms for exact solution as well as low degree polynomial time bounded algorithm/or approximation methods.

1. Introduction

Most of the recent work in computational geometry [D,DS,S] has centered on algorithms for performing operations on convex polygons. Unfortunately, in many applications areas [*DT,V*], we must deal with non-convex designs. To remedy this, we are faced with a choice of extending current algorithms to non-convex objects or finding methods of rewriting non-convex polygons as unions of convex polygons. The latter course is followed here, motivated by applications to problems of tool design and pattern recognition [*FP,V*] where one wishes to transform among various representations of an object using its representation as the union of convex objects as a base. Previous works in this area involve finding efficient triangulations of point sets and polygons and showing that a variation of that problem is NP-complete [*GJPT,L*]. Previous work on this problem has consisted of algorithms which solve either approximately or exactly simplifications of the problem [*FP,P*]. In each of these papers, the introduction of new points to the decomposition has been forbidden. These algorithms follow trivially from the algorithm given here, since their structure does not require much of the detailed analysis we give.

The problem which we are studying can be simply stated as:

> Given a simple polygon P, what is the smallest number of pairwise disjoint convex polygons, whose union is exactly P?

We first observe that any decomposition must consider all vertices of P at which the interior angle is reflex. Such vertices will be called notches, with N(P) denoting the number of notches. Each notch can be removed by the addition of a polygon to the decomposition. Also, at most 2 notches can be removed with the addition of a single polygon. Letting S(P) be the minimum number of polygons in a convex decomposition of P implies:

$$1+N(P)/2 \leq S(P) \leq 1+N(P)$$

However, to extend these simple observations is a difficult mathematical problem. To form minimal decompositions, additional (Steiner) points must be introduced as vertices of newly generated polygons. This removes the obvious finiteness of the problem and makes simple enumerative procedures impossible. Furthermore, the problem can't be treated in a local manner. These observations led to the conjecture that the problem was NP-complete.

Our analysis of the problem begins with the introduction of X-patterns from which all decompositions may be generated. An Xk-pattern is an interconnection of k notches forming a set of k polygons which remove all reflex angles occuring at the k notches and create no new notches. A complete decomposition will consist of a set of k X_{i_j}-patterns along with k' notches used to remove

*This research was started when the second author was at Yale. Portions of the research were supported by the Office of Naval Research under Grant N00014-75-C-0450 and the National Science Foundation under Grant MCS76-11460.

This research was also facilitated by the use of theory net, NSF Grant MCS78-01689.

See page ii

the remaining notches ($i_1+i_2+\ldots+i_k+k'=N(P)$),

yielding a decomposition into $N(P)+1-k$ polygons. It is clear that the decompositions which use the most X-patterns also minimize the number of convex polygons, suggesting a dynamic programming approach to the problem's solution. We define $S_j(i)$ as the maximal set of X-patterns which can be applied together to notches between v_i and v_{i+j-1} for $1\leq i,j\leq N(P)$, with subscripts taken modulo N(P). We then find $S_j(i)$ from $\{S_k(\ell) \mid i\leq \ell<\ell+k-1\leq i+j-1\}$.

There is exactly one type of X2-pattern and one type of X3-pattern, as shown in Figure 1. Since this is not the case for all k, dynamic programming does not yield a polynomial-time algorithm. We use structural facts about the decomposition to limit the types of interconnections occurring at notches yielding a polynomial-time bounded dynamic programming algorithm. The facts we need involve showing that an optimal decomposition can be assumed to have no polygon which is completely interior to the decomposition and no Steiner vertex of degree more than 3 or notch of degree more than 4..

These observations limit the types of interconnections which can occur at notches. The introduction of Y-patterns and Y-decompositions which are reduced forms of X-decompositions allow for a polynomial-time algorithm. Informally, a Y-pattern represents the skeleton of the decomposition as a forest with the leaves corresponding to notches. In what follows, we begin by showing the necessary facts about optimal decompositions, and proceed to describe formally X-patterns, Y-patterns, and Y-decompositions, showing that all optimal decompositions can be made to have the latter form. This leads to a polynomial-time algorithm for solving the problem. At present, the degree of this algorithm is 6, which we do not believe to be optimal. However, since the number of reflex angles in most practical polygons is small, the algorithm is of use in its current form. Furthermore it gives rise to subalgorithms which in time $O((N(P))^k)$ provide a solution within $1/(k+1)$ of optimal, for k=2,3,4.

2. Basic Definitions

Let P be a polygon with vertices $w_1,w_2,\ldots,w_n$ and reflex angles occurring at N(P) of the vertices $v_1,v_2,\ldots,v_{N(P)}$, with the v_i forming a subset of the w_j. All indices of vertices (resp. notches) are taken modulo n (resp. modulo N(P)). Vertices will always be taken to occur in clockwise order with angles given positive orientation if directed in this manner. All motion will occur in clockwise directions. A decomposition of P is a set of polygons $P_1,P_2,\ldots,P_k$ whose union is P and such that the intersection of any two if nonnull consists totally of edges and vertices. The goal of this research is to minimize k, given the constraint that all polygons should be convex. A decomposition realizing a minimum value of k will be called an optimal convex decomposition (OCD) of P.

We define the range of a notch v, R(v), as the set of points u on the boundary of P such that the segment vu divides P into two polygons having between them N(P)-1 notches. We also include in R(v) the edges vu_1 and vu_2, where u_1 and u_2 are the two end-points of this set. Thus, R(V) forms a subpolygon of P. Connecting each notch v to a point on the boundary of R(v) yields a decomposition of P into 1+N(P) convex polygons referred to as the naive decomposition. In this decomposition, each notch can be made to belong to exactly 2 polygons and have degree at most 3, the latter property being also shared by the vertices of the decomposition which are not vertices of P. These facts will become useful later. It is easy to see that this upper bound of 1+N(P) is tight. However, the total number of convex parts can be strictly less than 1+N(P) making the problem non-trivial.

We begin our discussion by defining a type of decomposition to be used in our analysis.

Definition: An X-decomposition is any convex decomposition containing no interior polygon and such that no more than 3 segments, those of the boundary of P included, intersect at a single point.

Theorem 1: Any OCD can be transformed into an OCD which is also an X-decomposition.

Proof: Let P be a polygon for which an OCD has been generated. We first show that it is possible to eliminate all interior polygons from the OCD without creating interior vertices of degree higher than 3. Then we show that all vertices of degree higher than 3 can be replaced by vertices of degree 3 or less. Here, we may create interior polygons which are then eliminated by the first process.

We regard the segments of the decomposition (excluding the boundary of P) as forming a graph. Next, we pick an interior polygon of the OCD and consider the connected component G to which the edges of this polygon belong (see Fig. 2-a). Note that G may contain other interior polygons. Let $a_1,\ldots,a_k$ denote the vertices of G which lie on the boundary of P. Clearly, the introduction of G in the decomposition creates an least k+1 polygons. Now, we remove G from the decomposition while leaving the rest of the decomposition unchanged. This is likely to exhibit notches but since G is a connected component, all of them are also notches of P, that is, are some of the a_i's. These notches are removed by applying the naive decomposition locally. This will introduce at most k+1 polygons all of which will not be interior to P. Repeating for all remaining interior polygons, eliminates them.

Let x be a vertex of degree greater than 3 as illustrated in Figure 2-b. Assume that vertex x is connected to vertices $y_1,\ldots,y_k$ for k>3. Assume further that x is a notch of P with xy_1 and xy_2 on the boundary of P (the case where it is not being treated similarly). Now we iterate on the following procedure. Let xy_i be such that the angle $y_{i-1}xy_{i+1}$ is not reflex and move

xy_i along xy_{i-1} or xy_{i+1} to form a new line $x'y_i$, with x' chosen close enough to x so as to preserve convexity. Continuing on this step, we reach a point where at most 4 lines $xy_1,\ldots,xy_4$ are left (see Figure 2-c). We now move both xy_3 and xy_4 to introduce a line segment xx'. As mentioned above, this procedure will most likely introduce interior polygons, however, their removal by the procedure given above does not increase the number of edges adjacent to any vertex of the decomposition.

When applying X-decompositions, we will consider each decomposition as a union of Xk-patterns along with possibly some line segments drawn to finish off the decomposition by the naive algorithm. By Xk-patterns, we mean that unit used to remove a set of k notches while introducing k-1 additional polygons to the decomposition. More precisely, if we regard an X-decomposition as a forest of trees (which is legitimate since there is no interior polygon), then an Xk-pattern is a tree which has **k** leaves, all of which are notches of P (see Figure 3 consisting of an X2- and an X3-pattern). Since the smallest Xk-pattern has k=2, we have the result:

Corollary: For any polygon P, an optimal decomposition consists of at least 1+N(P)/2 convex polygons.

An OCD will be obtained by maximizing the number p of X-patterns which can be applied together on P. More precisely, we have:

$$S(P) = N(P)+1-p$$

We omit the proof which proceeds by an easy induction on p. Unfortunately, the difficulty of determining whether, given k notches, there exists an Xk-pattern connecting them makes the existence of an efficient algorithm based on these patterns extremely unlikely. Therefore, we extend our work to Y-patterns which allow for such an algorithm.

Definition: A Yk-pattern is a tree involving k notches and made up of vertices of type N_1,N_2, and N_3 (see Figure 4-a) such that:

1) No edge joins nodes of type N3-N3

2) No edge joins nodes of type N1-N2

3) In any path containing 3 consecutive nodes of respective type N2,N3,N2, the N2-nodes lie on opposite sides.

A Y7-pattern and its representation are given in Figure 4-b.

A Y-decomposition is an X-decomposition with the possible addition of Y-patterns.

Informally, a Yk-pattern is an Xk-pattern where some edges emanating from notches have their length reduced to 0 in order to meet the specifications given above. In this regard, it is natural that Xk-patterns can often be transformed geometrically into Yℓ-patterns (ℓ>k) by slight perturbations involving stretching, shrinking, or rotating the lines of the original pattern. Indeed:

Theorem 2: Any X-pattern which is not geometrically reducible to an X2-,X3-, or X4-pattern, can be reduced to a Y-pattern:

Proof: The method involves applying continuous transformations called reductions to the lines of an X-pattern, which preserve its convexity as well as the convexity of all its angles. Before proceeding further, we have to ensure that reductions can be carried out freely without merging two patterns in the process, thus possibly increasing the number of polygons in the decomposition. To see this, consider an X-pattern T. By definition, no line of T can intersect any other line which does not belong to T or P. We claim that any reduction of T will preserve this property. Suppose that, in the course of a reduction, a point of T gets to intersect with a line L not in T, then this point has to be a vertex of T, therefore, 3 edges of T emanate from it and all 3 lie on the same side of L, thus exhibiting a reflex angle, and showing that the reduction was illegal. Now we can focus on our prime goal:

Given any X-pattern not reducible to X2, X3, or X4, we show that we can successively reduce it to a tree satisfying conditions 1),2),3) in the definition of Y-patterns.

1) Figure 5-I represents 2 consecutive N3-nodes which we want to split. We move one of them as indicated by the arrow. Then, either we get to 1) and we are finished or we get to 2) and another reduction leads to 2-1), 2-2), or 2-3). We then iterate on this process as indicated in the figure. Convergence is guaranteed since each reduction adds a different N2-node. We note that the figure investigates all cases except those representing extreme instances of Y-patterns, namely, the cases illustrated in Figure 4-c (knowing that case 2 represents 2 edges emanating from a N2-notch, one of which lies in the range of the notch, the figure should be self-explanatory). To handle these 4 cases, it suffices to note that, in each of them, we can prune one edge along with the adjoining subtree and still preserve the non-reflexivity of all the angles (we then iterate on the same procedure described above). This operation removes at least one N1-node from the pattern. Since pruning the pattern never adds any (except when dealing with case 2), convergence is guaranteed. The result also holds for case 2 where we actually remove at least two N1-nodes.

Figures 5-II and -III handle all the remaining cases in a similar manner. This procedure is to be applied as long as there are edges between two N3-nodes, with each application decreasing by 1 the number of such edges.

2) We note that the previous transformations never rotate the edges between an N1-node and an N3-node. Therefore, even if an N3-node is removed in the previous process, a connection N1-N2 will never occur.

3) Finally, we satisfy condition 3 by treating the 2 possible cases -IV and -V as indicated. Note that if we fall into one of these 4 cases of Figure 4-C, chopping an edge off will make condition 3 hold or will recreate a problem of type IV or V treated in a similar manner. The same remark as above guarantees convergence.

This theorem sets the stage for a polynomial-time algorithm.

3. The polynomial-time algorithm

So, a polynomial-time algorithm will result from polynomial-time algorithms for detecting X2-,X3-, and X4-patterns, for finding Y-patterns, and for combining these results to obtain an optimal decomposition. We begin with the first task.

Theorem 3: Testing for the possibility of an Xk-pattern connecting k notches of P can be done in constant time for k=2,3 and in linear time for k=4, after doing some preprocessing taking on the order of $N(P)^3$ time.

Proof: (See Figure 6 for details). The preprocessing consists of considering all the notches and for each triple v_i,v_j,v_k, defining ℓ_{ik} and r_{ij} as follows: For all notches t of P between v_k and v_i (clockwise order), consider the t which maximizes the oriented angle (v_iv_k,v_it); then ℓ_{ik} is v_it. Similarly, for all notches t between v_i and v_j, r_{ij} is v_it for the t which maximizes (v_it,v_iv_j). Now let ℓ and r be the two edges of $R(v_i)$ intersecting in v_i. If (ℓ_{ik},ℓ) is positive, we define ℓ'_{ik} as ℓ, else as ℓ_{ik}. Likewise, if (r,r_{ij}) is positive, we define r'_{ij} as r, else as r_{ij}. We can compute all of these lines in time $O(N(P)^3)$ operations. We also define M_{ij} as the intersection between r'_{ij} and ℓ'_{ji} (note that $M_{ij} \neq M_{ji}$ in general) and for each M_{ij} we record whether both segments v_iM_{ij} and v_jM_{ij} lie within P or not. A flag can be sent to 1 in the former case, 0 in the latter.

1) An X2-pattern can connect v_i and v_j if and only if v_iv_j lies within the two edge shaped regions (ℓ'_{ij},r'_{ij}) and (ℓ'_{ji},r'_{ji}) and the two corresponding angles are positive.

2) Given 3 notches v_i,v_j,v_k, if any of the 3 angles of the kind (ℓ'_{ik},r'_{ij}) is negative or one of the points M_{ij},M_{jk},M_{ki} does not exist or has a zero flag, no X3 is possible. Otherwise, v_i,v_j,v_k form an X3 is and only if the polygon $v_iM_{ij}v_jM_{jk}v_kM_{ki}v_i$ is simple and all of its interior angles alternate between >180 and <180 degrees.

3) Given 4 notches v_i,v_j,v_k,v_ℓ associating v_i with v_j and v_k with v_ℓ yields an X4 if and only if M_{ij} and $M_{k\ell}$ are defined, both flags are positive, and the segments $v_iM_{ij},v_jM_{ij},v_kM_{k\ell},v_\ell M_{k\ell}$, and $M_{ij}M_{k\ell}$ form an X4-pattern (this last condition involving just a test on the angles). Similarly, we can test for the associated pairs v_j with v_k and v_ℓ with v_i. Note that this procedure will not detect some X4-patterns which are reducible to Y5-patterns, yet all of the others. These patterns will be detected in a general procedure for Y-patterns, though. Likewise, by our method, an X3 reducible to X2 will not be detected as an X3 but as an X2.

It is clear that the necessary and sufficient conditions given can be checked in constant time for any set of 2 or 3 notches and in linear time for a set of 4 notches, since in the latter case we have to determine whether the "middle" edge of the X4-pattern lies within P or not. However, we believe that it is possible to determine all possible X2-, X3- and X4-patterns in less than $O(N(P)^5)$ operations by batching the computations.

We may now generate our algorithm, assuming that we have available lists of all sets of notches for which X2-,X3-, and X4-patterns are feasible (with the restriction mentioned above on X4). We define $V_j(i)$ as the set of j notches beginning at and including i and $S_j(i)$ as the maximal set of compatible patterns of type Xk ($k\leq 4$) as well as Y-patterns applicable to this set of notches. $S_{N(P)}(1)$ is now found by a dynamic programming approach. We find $S_j(i)$ from $\{S_k(\ell) \mid k\leq i+j-\ell$ and $i\leq\ell\leq i+j\}$. In applying this algorithm to X-decompositions, no polynomial-time algorithm could be derived because it was necessary to preserve all possibilities within each $V_j(i)$ for possible future interactions. Here, consider all Y-patterns having v_i as an N2-notch. v_i splits each of them into two subtrees, called Y-subtrees. We are interested in the set of all of those subtrees which have all of their notches in $V_j(i)$. We associate with each of them a pair (u,C) with u defined as the angle (T_u,L_i) where T_u is the edge adjoining v_i to the subtree lying in $V_j(i)$, L_i is the edge forming the counter-clockwise boundary of $R(v_i)$ and C is defined as the maximum number of X- or Y-patterns applicable in $V_j(i)$ and compatible with the Y-subtree currently considered (i.e. the savings, see Figure 7). More precisely, if $i,i_1,\ldots,i_k$ are the indices of the notches of the Y-subtree, C is defined as $|S_{i,i_1,\ldots,i_k}|$ where $S_{i,i_1,\ldots,i_k}$ is the maximum set of compatible patterns, that is,

$$S_{i,i_1,..,i_k} = \bigcup_{\ell=1}^{k-1} S_{i_{\ell+1}-i_\ell-1}{}^{(i_\ell+1)} U S_{i_1-i-1}{}^{(i+1)}$$

For obvious reasons, if two Y-subtrees in $V_j(i)$ have values (u,C) and (u',C') with $u \leq u'$ and $C \leq C'$, the first can be dropped. This means that in our set of (u,C), we keep only the maxima. Recall that in a subset of A of R^2, (a,b) is a maximum if there is no element (c,d) in A with $c \geq a, d \geq b$ and $(c,d) \neq (a,b)$. We call SKIM(A) the set of maxima in A. SKIM(A) can be computed in O(NlogN) operations if $|A| = N$[KLP]. It is this observation which guarantees a polynomial-time algorithm, since, for all i and j, $S_j(i)$ has at most j/2 members.

Let $B_j(i)$ be the set of pairs (u,C) so obtained with u<180 degrees. Initially, $B_1(i)$ is the singleton set (u,0) where $u=(R_i,L_i)$, R_i being the edge of P intersecting with L_i in v_i. Let $F_j(i)$ be defined in a similar fashion about v_i with the Y-subtree here being constructed in $V_j(i-j+1)$. $F_1(i)$ can be formed as $B_1(i)$ was. The $B_j(i)$ and $F_j(i)$ sets are then formed iteratively. To do this computation, we need a method for "patching" subtrees corresponding to parts of Y-decompositions together. This is done by a function we shall call $Y(v_i,ARG)$ where ARG is an argument of the form $B_b(a)$, or $F_b(a)$, or $(B_b(a),B_d(c))$, or $(F_b(a),F_d(c))$. We outline the algorithm for the first and third cases with the other cases easily following from this description. The operation of the following algorithm is shown in Figure 8.

Case 1: $Y(v_i,B_b(a))$

-$Y(v_i,B_b(a))$ is initially empty.

-IF (R_i,v_iv_a) and (R_a,v_av_i) are <180 degrees. THEN FOR all (u,C) in $B_b(a)$ such that $(v_av_i,T_u) < 180$ degrees:

BEGIN

$Y(v_i,B_b(a)) = Y(v_i,B_b(a)) U ((R_i,v_iv_a), S_{a,i_1,..,i_\ell,i})$ where $v_a,v_{i_1},..,v_{i_\ell}$ are the notches of the Y-subtree which the segment T_u connects to v_i.

END

ELSE $Y(v_i,B_b(a))$ remains empty.

Case 2: $Y(v_i,(B_b(a),B_d(c))$

-$Y(v_i,(B_b(a),B_d(c)))$ is initially empty.

-FOR all (u,C) in $B_b(a)$ and (u',C') in $B_d(c)$:

BEGIN

Apply the algorithm for detecting the possibility of an X3-pattern connecting (v_i,v_a,v_c) with the geometric notch formed at v_a being (L_a,T_u) instead of (L_a,R_a) (same for v_c). As to v_i, since the edge emanating from it does not have to lie within $R(v_i)$, we don't consider the notch formed by (ℓ'_{ic},r'_{ia}) but by (ℓ_{ic},r_{ia}).

-IF an X3-pattern between these new notches is possible.

THEN

Form the intersection of the regions formed by these 3 notches and let M be the vertex of this polygon which maximizes the angle $w=(R_i,v_iM)$.

IF w < 180 degrees

THEN

$(w,|S_{a,i_1,..,i_\ell,c,j_1,..,j_m,i}|)$ is added to $Y(v_i,(B_b(a),B_d(c)))$, where $v_a,v_{i_1},..,v_{i_\ell}$ and $v_c,v_{j_1},..,v_{j_m}$ are the notches involved in the Y-subtree corresponding to u and u'.

We can illustrate what the function Y() actually computes, on a particular value of the argument, say, $(B_b(a),B_d(c))$. The result is the set of all possible pairs (u,C) obtained by linking v_i to two Y-subtrees in $B_b(a)$ and $B_d(c)$ respectively. Every time such a link is possible, it is made so that u is maximum (Remark: Actually, in $B_j(i)$ and $F_j(i)$, in addition to (u,C), we should also record the corresponding subtree, but we will not mention it for the sake of simplicity).

We also define $Y'(v_i,B_b(a))$ just like Y() with the same argument, with the only difference that all statements on (R_i,v_iv_a) now bear on (v_iv_a,L_i) - see Figure 8 for an illustration of the difference. (We define $y'(v_i,F_b(a))$ by a similar process).

From previous results, we know that each step can be computed in constant time. Therefore, the running time of the whole algorithm (Y or Y') is $O(|ARG|)$, where $|ARG|=|a|$ if ARG=a and $|ARG|=|a||b|$ if ARG=(a,b).

Thus, within a set of notches $V_j(i)$, we have a way of determining all the Y-subtrees which, later, may give rise to Y-patterns, all having either v_i or v_{i+j-1} as an N2-node. In fact, we keep only those that are candidates for an OCD. Now, it is natural to wonder how to detect all Y-patterns lying within $V_j(i)$ which are worth considering.

To cope with this problem, we define another function <ARG> which takes 2 or 3 Y-subtrees and constructs a Y-pattern if these subtrees can be patched together. More precisely, ARG is any argument of the kind: $(B_b(a),B_d(c))$, $(F_b(a),F_d(c))$, $(B_b(a),F_d(c))$, $(B_b(a),B_d(c),B_f(e))$, $(F_b(a),F_d(c),F_f(e))$. We outline the algorithm for only some of these cases with the other cases directly following from this description.

Case 1: $<B_b(a),B_d(c),B_f(e)>$

Let (u,C) be in $B_b(a)$. We define a new notch at $v_a,(L_a,T_u)$. However, if b=1, we keep the regular one (L_i,R_i). Similarly, for all (u,C) in $B_b(a)$, (u',C') in $B_d(c)$, (u",C") in $B_f(e)$, we compute these notches and apply the algorithm for detecting the possibility of an X3-pattern between them. With the same preprocessing set out above, this detection is still done in constant time. If the result is yes, the 3 subtrees corresponding to u',u',u" can be linked together to form a Y-pattern which we record with its savings, that is C+C'+C". The function <ARG> returns the value of the maximum C+C'+C" recorded, as well as the corresponding Y-pattern (or nothing if no record has been made).

Case 2: $<B_b(a),B_d(c)>$ or $<B_b(a),F_d(c)>$

It is strictly similar to the previous case with the algorithm for X2-patterns.

As in the case of function Y(), the running time of these algorithms is $O(|ARG|)$, with $|ARG|=|a||b|$ if ARG=(a,b) and $|ARG|=|a||b||c|$ if ARG=(a,b,c). An improvement can be made here which turns out to cut down the total running time by a factor of N(P).

The idea is, in computing $<B_b(a),B_d(c),B_f(e)>$, not to record all possible patchings but only the best (same with the F). Assume that the pairs (u,C) are stored sorted. We have for $B_b(a)$: $(u_1,C_1),(u_2,C_2),\ldots$ with the C_i (resp. u_i) in increasing (resp. decreasing) order. For all pairs (u_i,C_i) in $B_b(a)$, (u_j,C_j) in $B_d(c)$, (u_k,C_k) in $B_f(e)$, we define f(i,j,k) = 0 if the corresponding T_{u_i},T_{u_j},T_{u_k} cannot be patched to form a Y-pattern, or if one of the indices i,j,k is out of range, 1 otherwise. Let m be the maximum value that i,j,k can take on. We proceed as follows:

-i=j=m; k=1; iterate for i=m,m-1,..,1.

-Decrement j until f(i,j,k)=1.

-As long as f(i,j,k)=1, increment k by 1. When f(i,j,k)=0, record the triple (i,j,k). Decrement j by 1 as long as f(i,j,k)=0. When f(i,j,k)=1, increment k by 1 and iterate.

-Finally, keep the maximum $C_i+C_j+C_k$ among all recorded triples as well as the corresponding Y-pattern.

The running time of this algorithm will be $O(m^2)$. Its correctness stems from the fact that we compute exactly all the possible patchings (i,j,k) such that (C_i,C_j,C_k) are maxima in the sense defined above. Also, the computation is valid since the 3 sets of (u,C) are sets of maxima, that is, distinct (u,C) and (u',C') with u<u' and C<C' cannot be found in any of them.

As will soon become apparent, the advantage of Y-patterns over X-patterns is that we have a relatively simple function, <>, which detects all possibilities of Y-patterns which are candidates for an OCD. Finally, we are ready to assemble the pieces of the algorithm to give our final result:

Theorem 4: It is possible to find an optimal convex decomposition of a simple polygon in polynomial time.

Proof: We set out the algorithm and give its running time.

Preprocessing

Check that P is simple and not convex. Make a list of the notches in the usual order. For each v_i, compute R_i and L_i as defined above. Also compute ℓ_{ij} and r_{ij} for all pairs of notches (v_i,v_j). (Recall that all these notations were defined in the section related to the detection of small X-patterns).

Initialisation of B(i),F(i),S(i)

For i=1,..N(P)

BEGIN

-$B_1(i)=F_1(i)=\{((R_i,L_i),0)\}$

-$S_1(i)$=empty set

END

Detection of small X-patterns

Apply the algorithm for detecting X2,X3, and X4-patterns, for all pairs, triples, and quadruples

of notches, and record all possible patterns, forming 3 sets: X2,X3,X4.

Dynamic programming detection of Y-patterns

FOR j=2,..,N(P) BEGIN

FOR i=1,..,N(P) BEGIN "step 1 through step 3"

Step 1:

"Computes $B_j(i)$ and $F_j(i)$ from the previous values. In the case of $B_j(i)$, we look for all the possible Y-subtrees merging in v_i, which lie between v_i and v_{i+j-1} and may, later, give rise to a Y-pattern. Because of the nature of Y-patterns, the 3 cases of Figure 9-a exhaust all possibilities. All the new pairs (u,C) thus obtained are combined with $B_{j-1}(i)$ and only the maxima of the resulting set are finally kept. The case $F_j(i)$ is exactly symmetric."

-B1 $=Y(v_i,F_{j-1}(i+j-1))$

-B2 $=U_u Y'(v_i,B_u(i+j-u))$

-B3 $=U_u Y(v_i,F_u(i+u),F_{j-u-1}(i+j-1))$

-Bj(i)=SKIM($B_{j-1}(i)$U B1 U B2 U B3)

-F1 $=Y(v_i,B_{j-1}(i-j+1))$

-F2 $=U_u Y'(v_i,F_{j-u}(i+1-u))$

-F3 $=U_u Y(v_i,B_u(i-u),B_{j-u-1}(i-j+1))$

-$F_j(i)$=SKIM($F_{j-1}(i)$U F1 U F2 U F3)

for all u; $1\le u<j$

Step 2:

"Computes $Y_j(i)$ from the previous values. By using <>, it tests for all possible patchings of 2 or 3 Y-subtrees, so far unexplored. Figure 9-b gives all possible cases. Recall that when we test for patchings between, say, three B(), we only keep the "best" patching if there is any. In the following, MAX stands for: 'of maximal cardinality'."

-Y1 $=U_u <B_u(i),F_{j-u}(i+j-1)>$

-Y2 $=U_u <B_u(i),B_{j-u}(i+u)>$

-Y3 $=U_{u,v} <B_u(i),B_v(i+u),B_{j-u-v}(i+u+v)>$

-Y4 $=U_{u,v} <F_u(i+u-1),F_v(i+u+v-1),F_{j-u-v}(i+j-1)>$

for all u,v; $1<u+v<j$

-$Y_j(i)$=MAX(Y1,Y2,Y3,Y4)

Step 3:

Computes $S_j(i)$ from the previous values. It finds an OCD of the part of P taken between v_i and v_{i+j-1} from OCDs of parts stretching over fewer notches. It first finds the best decomposition without any pattern connecting both v_i and v_{i+j-1}, then it looks for the best when allowing such a pattern of type Xi for small i. Finally it allows for a Y-pattern."

-A $=MAX\ \{S_k(i)US_{j-k}(i+k)\}$ for all k; $0<k<j$

-B $=MAX\ \{\{x_{i,a_1,..,a_\ell,i+j-1}\}US_{i,a_1,..,a_\ell,i+j-1}\}$

for all $a_1,..,a_\ell$, integers between i and i+j-1, with $\ell>1$, such that $x_{i,a_1,..,a_\ell,i+j-1}$ is a X2-,X3-, or X4-pattern.

-C =(the Y-pattern $y_{u,..,v}$ in $Y_j(i)$)$US_{u,..,v}$

-$S_j(i)$=MAX {A,B,C}

END END

Step 4:

Apply the naive algorithm on the set of remaining notches.

Correctness

The computation of $B_j(i)$ proceeds from $B_{j-1}(i)$ along with all the Y-subtrees with 2 nodes at v_i and V_{i+j-1}, the former being of type N2. The computation is feasible since we only use $B_u(v)$ with u<j (same for F()). The crux is that we can limit ourselves to the subtrees of F() and B(), since all of the others which could be linked and yield a pair (u,C) would be eventually eliminated in the SKIM process. This can be expressed in a crude manner by saying that all that can be done at v_i with (u,C) in $B_j(i)$ can be done at least as well with (u',C') in $B_j(i)$ if u'>u and C'>C. Step 2 computes the best Y-pattern to apply between v_i and v_{i+j-1}. Note that for the sake of simplicity, Y4 is conservative and may even detect Y-patterns which do not have v_{i+j-1} as a node.

Time complexity

Let N denote N(P). The preprocessing can be done in time $O(N^3)$ [5]. The detection of small X-patterns requires $O(N^5)$ time since our detection of X4 is linear. In both $B_j(i)$ and $F_j(i)$, all the pairs (u,C) satisfy $0\le C\le N/2$, and since the C are integers, the cardinality of these sets cannot exceed N/2+1. Using the previous results on the running time of the function Y(), we have that B1,B2,B3 require at most $O(N^3)$ time and contain at most N/2 elements (same for $F_j(i)$). Step 1 being iterated N^2 times will account for $O(N^5)$ in the running

time. From previous results on function <>, it follows that each call on <> can be executed in $O(N^2)$ time. Therefore the N^2 iterations of step 2 will require $O(N^6)$ time. Finally, each pass through step 3 takes time O(N) for A, $O(N^2)$ for B, O(1) for C, the total summing up to $O(N^4)$ time. Thus we have an algorithm with a total running time to $O(N^6)$.

4. Conclusions

We have presented an algorithm which, in polynomial time, determines the minimal number of convex polygons into which a simple non-convex polygon can be decomposed. The algorithm is inefficient in its present state, though we believe improvements to be possible. Variants of the algorithm can be used to yield efficient approximation mechanisms. For example, if we only consider Xi-patterns for i<k, we are guaranteed an algorithm which is within 1/(k+1) of optimal yet requires at most $O(N(P)^k)$ operations. Setting k=2 or 3 makes this an attractive procedure. Furthermore, in actual practice, the number of notches of a polygon is greatly dominated by the total number of vertices, making this algorithm more realistic. An added benefit of this algorithm is that the optimal decompositions are generated. For an application such as pattern recognition, this is especially attractive as it allows for an easy pattern test after the algorithm has been applied to the set of allowable patterns (in a preprocessing step) and then to a new pattern to be tested against this set.

A number of interesting problems remain in this area. For example, there remains the problem of extending this algorithm to 3-dimensional polyhedra, or to decompositions where we can write a polygon as a sum and difference of convex polygons. Finally, there remains the problem of efficiently implementing this algorithm and adding it to existing application-oriented software (see e.g. [*D*]). Empirical data obtained from the use of the algorithm will no doubt effect future design criteria in its extensions.

5. References

[*D*] Dobkin,D., Keyword structures in a language for computer geometry, Bell Laboratories Technical Memorandum, TM 78-1271-9, June 29,1978.

[*DL*] Dobkin,D. and Lipton,R., Multidimensional searching problems, SIAM Journal on Computing, Vol. 5, No. 2, June 1976, pp. 181-186.

[*DS*] Dobkin,D. and Snyder,L., On a general method of maximizing and minimizing among certain geometric problems, in preparation.

[*DT*] Dobkin,D. and Tomlin,D., Cartographic modelling techniques in environmental planning: an efficient system design, submitted for publication.

[*FP*] Feng,H. and Pavlidis,T., Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition, IEEE Transactions on Computers, Vo. C-24, No. 6, June 1975, pp. 636-650.

[*GJPT*] Garey,M., Johnson,D., Preparata,F., and Tarjan,R., Triangulating a simple polygon, Information Processing Letters, Vol. 7, No. 4, June 1978, pp. 175-180.

[*L*] Lloyd,E., On triangulations of a set of points in the plane, 18th Annual IEEE FOCS Conference Proceedings, Providence, R.I., October 1977, pp. 228-240.

[*P*] Pavlidis,T., Analysis of set patterns, Pattern Recognition, Vol. 1, 1968, pp. 165-178.

[S] Shamos,M., Geometric Complexity, PhD Thesis, Yale University, May 1978.

[*V*] Volecker,H. private communication to D. Dobkin, November 14, 1977.

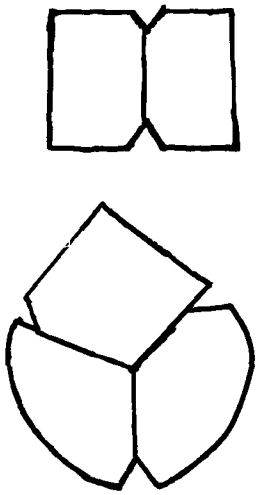
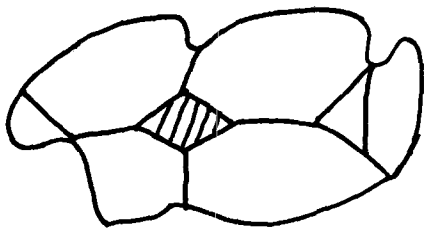
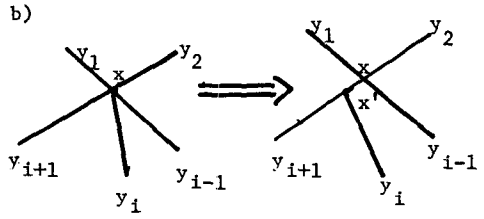


FIGURE 1 An X2-pattern and an X3-pattern

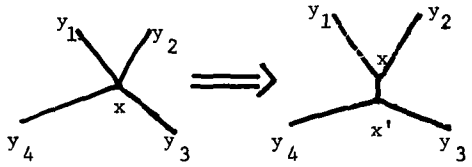


a) Eliminating interior polygons from an Xdecomposition

b)



c)



b,c) Reducing vertex degree in an Xdecomposition

FIGURE 2

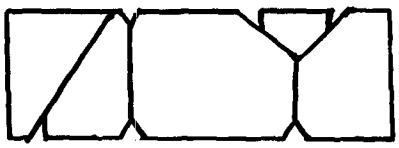
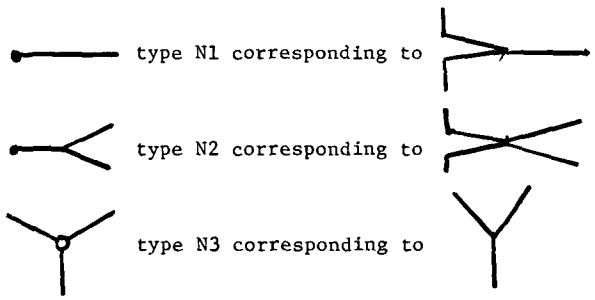
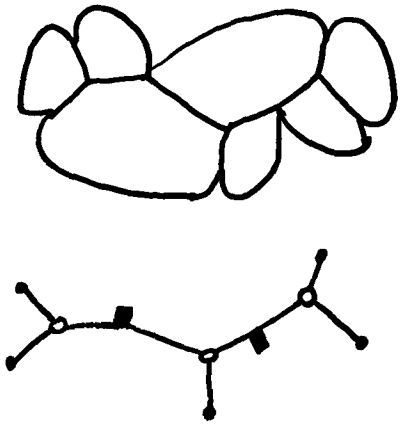


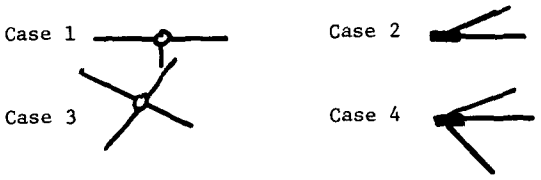
FIGURE 3
An Xdecomposition involving an X2-pattern and an X3-pattern



a) Notch Type



b) A Y7 pattern and its representation



c) Extreme instances of Y pattern vertices

FIGURE 4

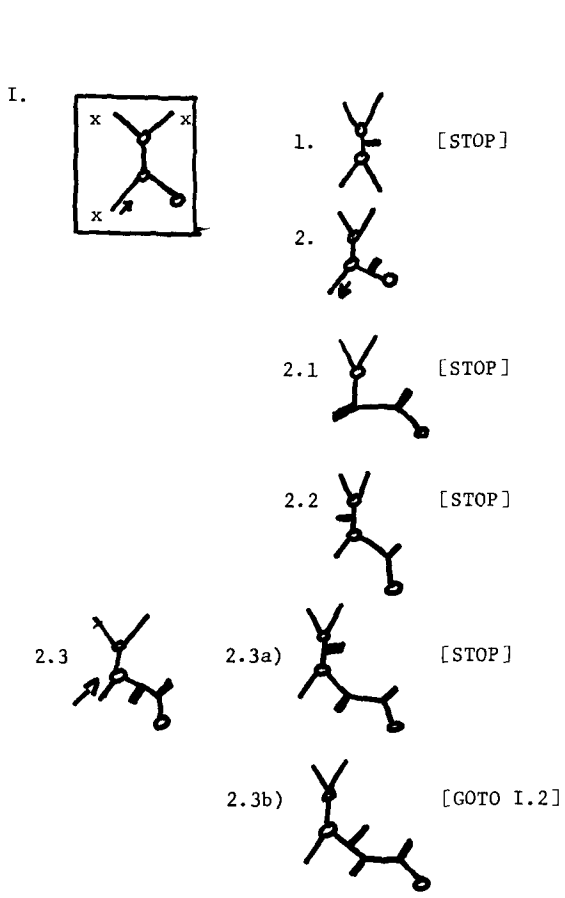


FIGURE 5 The proof of Theorem 2.

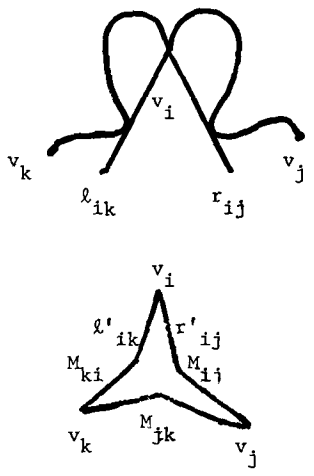


FIGURE 6 Testing for Xk-patterns (k≤4)

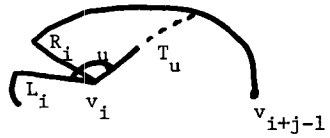


FIGURE 7 Computing the savings in $V_j(i)$

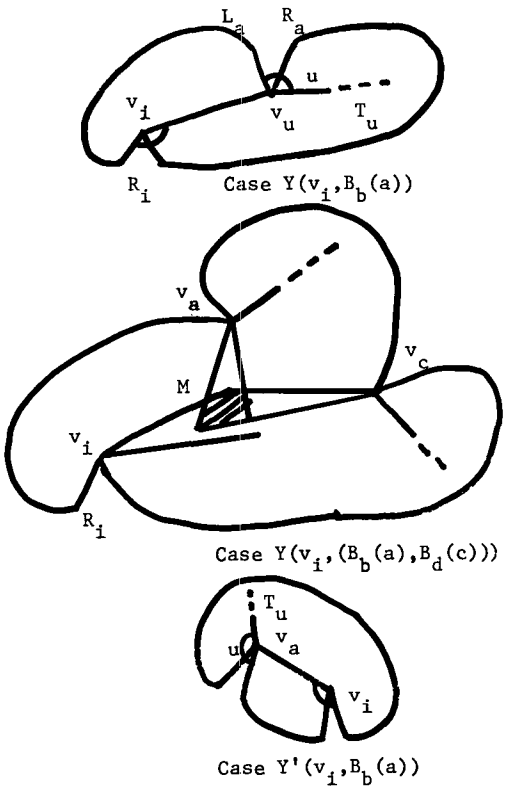
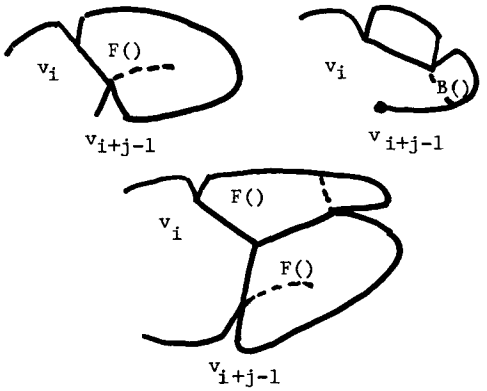
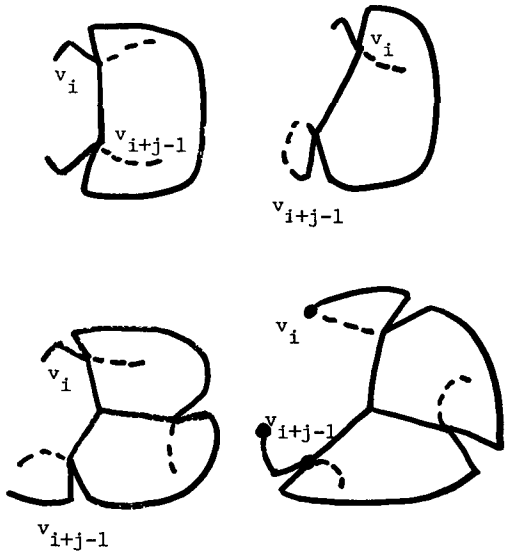


FIGURE 8 Computing $Y(v_i,ARG)$



a) Computing $B_j(i)$ and $F_j(i)$



b) Computing $Y_j(i)$

FIGURE 9