ELSEVIER

European Journal of Operational Research 121 (2000) 330–342

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Theory and Methodology

Algorithms for the decomposition of a polygon into convex polygons

J. Fernández, L. Cánovas, B. Pelegrín *

Departamento de Estadística e Investigación Operativa, Universidad de Murcia, Campus de Espinardo, 30100 Espinardo (Murcia), Spain

Received 1 July 1997; accepted 1 December 1998

Abstract

Decomposing a non-convex polygon into simpler subsets has been a recurrent theme in the literature due to its many applications. In this paper, we present different algorithms for decomposing a polygon into convex polygons without adding new vertices as well as a procedure, which can be applied to any partition, to remove the unnecessary edges of a partition by merging the polygons whose union remains a convex polygon. Although the partitions produced by the algorithms may not have minimal cardinality, their easy implementation and their quick CPU times even for polygons with many vertices make them very suitable to be used when optimal decompositions are not required, as for instance, in constrained optimization problems having as feasible set a non-convex polygon (optimization problems are usually easier to solve in convex regions and making use of a branch and bound process or other techniques, it is not necessary to find the optimal solution in all the subsets, so finding convex decomposition with minimal cardinality may be time-wasting). Computational experiments are presented and analyzed.

Keywords: Polygon decomposition; Approximation; Facility location

1. Introduction

A *convex decomposition* of a polygon P is a set of convex polygons whose union gives P, and such that the intersection of any two of them, if non-empty, consists totally of edges and vertices. The problem of decomposing a simple polygon into convex polygons has been an important theme in the literature due to its many and diverse applications. For instance, it is of great interest in object representation since complex geometric structures are more easily handled when decomposed into simpler structures. Other examples are pattern recognition [6], database systems [14] and graphics [16]. For further motivation on decomposition problems see [2,6,21]. For different but related problems see [17,18] or [20].

The problem that led us to consider it was the definition of the feasible set in constrained planar location problems, i.e., given a set of points in the plane (representing the location of cities or customers) where a new point (representing a facility) should be located in the feasible set so as to

* Corresponding author. Tel.: +34 968 363635; fax: +34 968 364182.

E-mail address: pelegrin@fcu.um.es (B. Pelegrín).

0377-2217/00/$ - see front matter
PII: S0377-2217(99)00033-8

optimize a given function. To model the different aspects that must be taken into account, many facility location problems are being considered nowadays. The interested reader is referred to the excellent books of Drezner [4], Francis et al. [10] or Love et al. [15]. But to adapt a location model to a given region, it is necessary that the points to be located belong to a particular geographical area. That area may have a more or less complicated shape, but it can always be approximated by a polygon, whose number of vertices will depend on the desired accuracy of the approximation. Unfortunately, those polygons, usually non-convex, cannot be written in an analytical way through a set of linear constraints. So, in order to get an analytical expression of the feasible set the polygon must be decomposed into simpler sets. The location problem then, will be solved in some (usually not all) of those sets. The most appropriate kind of sets into which the polygon can be decomposed is convex polygons, for both their easy analytical writing and good optimization properties. Furthermore, from the optimization point of view, it is not advisable to add new (Steiner) vertices in the partition because in many location problems vertices must be treated with special attention since they may be local (sometimes global) optimal points or give information about optimal points, and adding new vertices will cause extra work in fathoming them. There are algorithms to triangulate a polygon, i.e., to subdivide it into triangles without adding new vertices [1] but they produce partitions into many subsets. If, in addition to this, we have procedures that to be applied only require convexity properties (and not necessarily triangles), as is the case in most optimization algorithms, then it is advisable to use decompositions into as big as possible convex sets.

Keil [12] presented an exact algorithm to decompose a polygon into the minimum number of convex polygons without adding new vertices. Nevertheless, it is a dynamic programming algorithm and although he used the concept of equivalent states, as introduced in [5], to reduce the size of a state space in order to achieve a polynomial algorithm, it has like other dynamic programming algorithms the problem known as *the curse of dimensionality* [3], i.e., when run on a computer it is time consuming and needs much storage space, specially for polygons like those which appear in location problems, with many vertices.

On the other hand, we should remember that in order to obtain the decomposition of a polygon into the minimum number of convex polygons additional (Steiner) points must be introduced as vertices of newly generated polygons. This other problem was recently proved in [2] to be doable in polynomial time, contrary to what was thought, although in that article the authors admitted that the algorithm they proposed to show is "inherently intricate and implementing it in its most elaborate form is certainly a formidable task". Unfortunately, that is true for most of the decomposition algorithms in the literature. As pointed out in [19], the researchers involved in this kind of problems have usually pursued asymptotically optimal algorithms and it is now when greater emphasis is being placed on computationally robust algorithms, on approximations and on the need for simple, practical algorithms.

The aim of this paper is to present fast and easy to implement algorithms to do convex decompositions with low cardinality and with the additional requirement that new vertices cannot be introduced in the partition. The algorithms generalize those in [9] and an actual implementation of all of them can be obtained in the O.R.S.E.P. section of the European Journal of Operational Research 102, no. 2 [8]. The algorithms follow a "divide and conquer" scheme. They start with the whole polygon to be decomposed. Then, given an initial vertex, a convex polygon of the partition is generated using a procedure and is cut off from the initial polygon. This process is repeated with the remaining polygon until it is convex, in which case it will be the last polygon of the partition. Although it is not guaranteed that the algorithms yield minimal size decompositions, their quick running times, the low number of convex polygons they produce and their easy implementation make them very suitable to be used when a (non-optimal) decomposition of a polygon is needed. A good example is in constrained optimization problems where the feasible set is a non-convex polygon. A common method to cope with these problems is to solve them in the convex subpolygons (what is

an easier problem), and because of the use of a branch and bound process or other techniques it is usually not necessary to find the optimal solution in all of them, so finding convex decomposition with the minimum number of convex polygons may be time-wasting. For other algorithms with the same aim see [11] or [21].

The rest of the article is organized as follows. In the following section, three different algorithms for the decomposition of a polygon into convex polygons are presented. In Section 3, a procedure to merge all the convex polygons of a partition whose union remains a convex polygon is given. This merging process can be applied not only to our algorithms, which may generate partitions containing unnecessary edges, but to any other partitioning process which produces unnecessary edges (for instance the algorithms given in [21]). In Section 4 the results of some computational experiments are presented and analyzed to show the performance of the algorithms, including a comparison with Hertel and Mehlhorn's algorithm [11], and in the last section we give conclusions and directions for further research.

Throughout this paper, the following convention on the representation of angles is used. Let a, b and c be three different points, $ang(a, b, c)$ denotes the angle between 0° and 360° swept by a counterclockwise rotation from line segment ba to line segment bc. The vertices of a polygon displaying a *reflex* angle, that is, greater than 180° are called *notches*.

2. Algorithms

In this section, three different algorithms for the decomposition of a polygon into convex polygons are presented. All of them follow the divide and conquer scheme, their only difference being the procedure used to generate the convex polygons of the partitions. Since the three procedures, which will be called *MP1*, *MP2* and *MP3*, used in Algorithm 1, 2 and 3, respectively, follow the same ideas only the first one will be described in detail.

Let P denote the simple polygon to which we want to apply the procedure *MP1*, given by its vertices $v_1, \ldots, v_n$, in clockwise order. At the beginning, P is the initial polygon, but as the algorithms go on, P will be the polygon obtained from the initial one by cutting off the convex polygons already generated.

The vertices of the next future convex polygon of the partition are stored in a list, L, initially with only one vertex. From it, the convex polygon with the highest possible number of vertices of P is generated by adding to L consecutive (in clockwise order) adjacent vertices of P while possible, and then removing backward some of them if necessary. The final convex polygon is obtained by drawing the diagonal joining the last and first vertices of L, which is the only artificial edge we add. The detailed process is as follows.

Initially, the list L consists of one vertex, say v_1. Then, we add the next consecutive vertex of P, v_2, to L. If the last vertex we have already added to L is v_i then we *provisionally* add v_{i+1} to L if $ang(v_{i-1}, v_i, v_{i+1})$ is not reflex (to avoid v_i to become a notch) and if both $ang(v_i, v_{i+1}, v_1)$ and $ang(v_{i+1}, v_1, v_2)$ are not reflex (to make sure the diagonal $v_{i+1}v_1$ close in fact a convex polygon, preventing v_{i+1} and v_1 from being notches, respectively). Both conditions are necessary and independent, as shown in the polygons of Fig. 1.

We go on adding new vertices to L until all the vertices of P are in L or until we first find a vertex failing one of the two conditions. In the first case, the convex polygon generated is the whole polygon P, that is, it was convex, and the algorithms stop, being P the last convex polygon of the decomposition of the initial polygon. So, let us suppose that the final provisional list is $L = \{v_1, \ldots, v_r\}, 2 \leqslant r < n$. If $r = 2$ then L produces the edge v_1v_2, which is not considered as polygon of the final partition, and the procedure stops. This happens for instance when both v_i and v_{i+1} are notches. The other possibility is $2 < r < n$. In this case, we have to check whether the convex polygon generated by the diagonal v_rv_1 contains in its interior vertices of $P \setminus L$ (see Fig. 2(a)).

Since there might be many vertices to be checked and those checkings must be done every time the procedure is called in the algorithm and generates a list L as in the case described above, a method to save time is needed. First, notice that if there are vertices of $P \setminus L$ inside the convex poly-

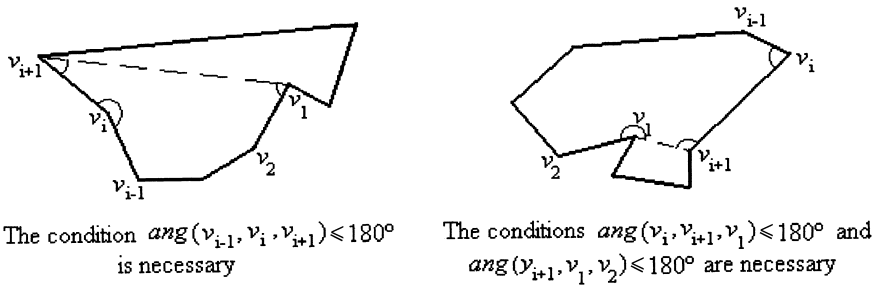

Fig. 1. Non-convex polygons generated when one of the angles is not considered.

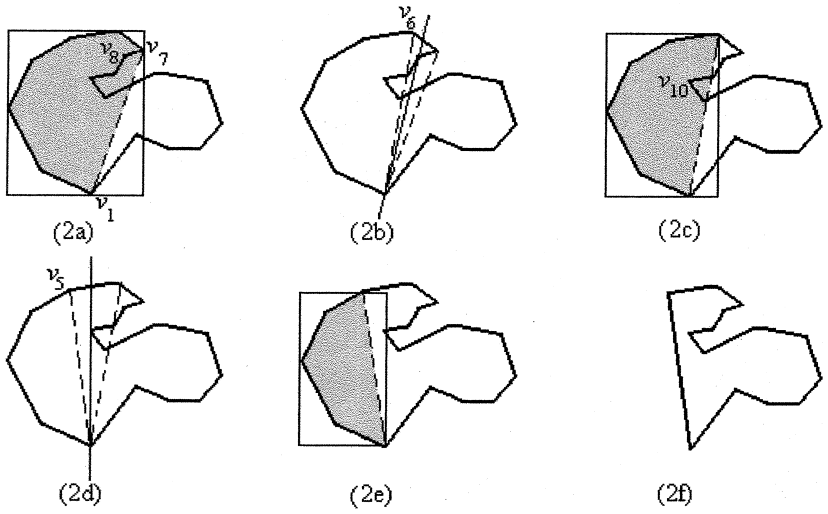

Fig. 2. Removing vertices of L until the convex polygon does not contain vertices of P.

gon, then at least one of them must be a notch. So we can reduce the checking to the notches of $P \setminus L$. When the checking starts, we first generate the smallest rectangle R with sides parallel to the axes containing all the vertices of L, $[\min\{v_{i,x}: v_i \in L\}, \max\{v_{i,x}: v_i \in L\}] \times [\min\{v_{i,y}: v_i \in L\}, \max\{v_{i,y}: v_i \in L\}]$, $v_i = (v_{i,x}, v_{i,y})$, $i = 1, \ldots, r$. If v is a notch of $P \setminus L$ we first check whether v is inside R. Observe that if v is not inside R then it cannot be inside the polygon generated by L either, so the checking to see whether a notch of $P \setminus L$ is inside the convex polygon can be reduced to the notches found in the rectangle. This test can be done very quickly and is specially useful for polygons with many vertices (and notches) since it may avoid the checking of many notches. Finally, if v is a notch inside R the checking to see whether v is inside the convex polygon generated by L is done by studying whether v verifies all the linear constraints defining the polygon.

If a notch v is found to be certainly inside the polygon generated by L, then we remove from L its last vertex v_r, and all the vertices of L in the semiplane generated by $v_1 v$ containing v_r. This process is repeated with the new L until no notch is inside the polygon generated by L. If L has then more than two vertices, it generates one of the polygons of the partition, else the procedure does not generate a polygon in this call. We can see an example of this process in Fig. 2. There, the initial diagonal (see 2a) is $v_7 v_1$, but the notch v_8 is found inside R and inside the convex polygon generated

by L (in grey). So the line v_1v_8 is drawn (see 2b) and after removing the vertices of L in the semi-plane generated by v_1v_8 containing v_7, the new diagonal to be considered is v_6v_1. But the notch v_{10} is found inside the new R (see 2c) and also inside the convex polygon generated by the new L. After drawing the line v_1v_{10} (see 2d) and removing vertices, the new diagonal is v_5v_1. This time no notch is inside the new convex polygon (see 2e), so v_5v_1 is a diagonal of the partition. The convex polygon it generates is cut off from the main polygon and the process is repeated with the remaining polygon (see 2f).

Notice that a similar procedure *MP1′* can be done going counterclockwise instead of clockwise, as we have done. We also want to point out that it is possible to mix both kinds of search to get a more powerful one: we begin generating a list L with *MP1* and then, so as to get a bigger polygon, we try to expand L by adding consecutive, in counterclockwise order, adjacent vertices to it in its first position, i.e., we use *MP1′* with L as initial list. This is in fact our procedure *MP2*. A different procedure *MP2′* can be obtained going firstly counterclockwise with *MP1′* and then clockwise with *MP1*. Our last procedure *MP3* works like *MP2* but only chooses a convex polygon as a convex polygon of the partition if one (or both) of the vertices of the diagonal is a notch; if it is not the case, the convex polygon is not considered as a convex polygon of the partition and the procedure *MP2* is called again considering as initial vertex the last vertex of the previous not considered convex polygon. Using this third procedure, it is guaranteed that the partitions obtained do not contain any diagonal which connects two vertices which are not notches. Those partitions are known to be non-optimal since those diagonals can always be removed without creating non-convex pieces. In the polygons of Fig. 3 is shown the convex polygon generated by the procedures when the initial vertex considered is v_1. Observe that when using MP3 the first diagonal v_4v_{16} and the second one v_7v_4 are not chosen as diagonals of the partition.

Below the steps of the first algorithm are detailed. L^m will denote the list L containing the vertices of the convex polygon at the mth iteration.

Algorithm 1

1. **Read** in clockwise order the vertices v_i and store them in a cyclical list $P = \{v_1, \ldots, v_n\}$.
2. $L^0 \leftarrow \{v_1\}$; $m \leftarrow 1$.
3. **While** $n > 3$ **do** (*Begin MP1*)
   - 3.1. $v^{(1)} \leftarrow Last[L^{m-1}]$; $v^{(2)} \leftarrow Next[P, v^{(1)}]$.
   - 3.2. $L^m \leftarrow \{v^{(1)}, v^{(2)}\}$; $i \leftarrow 2$; $v^{(i+1)} \leftarrow Next[P, v^{(i)}]$.
   - 3.3. **While** $ang\{(v^{(i-1)}, v^{(i)}, v^{(i+1)}), (v^{(i)}, v^{(i+1)}, v^{(1)}), (v^{(i+1)}, v^{(1)}, v^{(2)})\} \leqslant 180^\circ$ **and** $|L^m| < n$ **do**
     - 3.3.1. $L^m \leftarrow L^m \cup \{v^{(i+1)}\}$; $i \leftarrow i + 1$; $v^{(i+1)} \leftarrow Next[P, v^{(i)}]$.
   - 3.4. **If** $|L^m| \neq |P|$ **then**
     - 3.4.1. **Obtain** the list *LPVS* with the vertices $v_i \in P \setminus L^m$ which are notches.
     - 3.4.2. **While** $|LPVS| > 0$ **do**
       - •**Obtain** the smallest rectangle R with sides parallel to the axes containing all the vertices of L^m.
       - •*Backward* $\leftarrow$ *false*.
       - •**While** not *Backward* **and** $|LPVS| > 0$ **do**
         Repeat

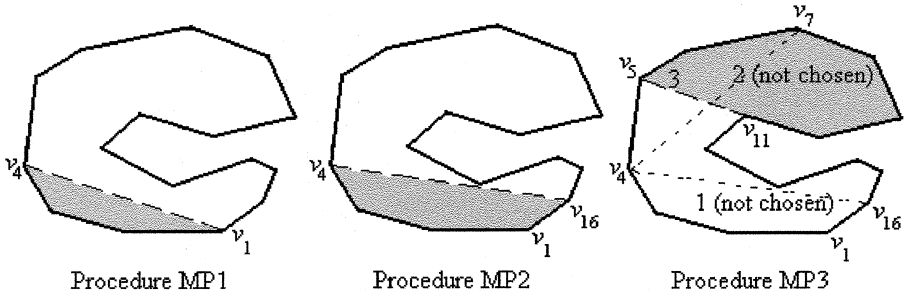

Fig. 3. Polygon generated from vertex v_1 with the procedures *MP1, MP2* and *MP3*.

$v \leftarrow First[LPVS]$.
If $v \notin R$ **then** $LPVS \leftarrow LPVS \setminus \{v\}$
Until $v \in R$ **or** $|LPVS| = 0$.
If $|LPVS| > 0$ **then**
If v is inside the polygon generated by L^m **then**
Obtain the set *VTR* of vertices of L^m in the semiplane generated by $v^{(1)}v$ containing $Last[L^m]$.
$L^m \leftarrow L^m$ *VTR*; *Backward* $\leftarrow$ *true*.
$LPVS \leftarrow LPVS \setminus \{v\}$.(*End MP1*)

3.5. **If** $Last[L^m] \neq v^{(2)}$ **then**
3.5.1. **Write** L^m as a convex polygon of the partition.
3.5.2. $P \leftarrow (P \setminus L^m) \cup \{First[L^m], Last[L^m]\}$; $n \leftarrow n - |L^m| + 2$;
3.6. $m \leftarrow m + 1$.

The algorithm calls the procedure *MP1* until the polygon to be decomposed into convex polygons is in fact convex. The first vertex in the lists storing the vertices of the convex polygons of the partition is chosen as the next vertex (in clockwise order) of P from which we can generate a new convex polygon. Observe that sometimes it may happen that no convex polygon is generated from the initial vertex $v^{(1)}$. In this case, we *skip* that vertex and try to generate a convex polygon considering the next vertex as the first one in the list.

Observe that we initially set $L^0 = \{v_1\}$, but we could also have chosen any other vertex. In fact, this choice affects the final solution, i.e., different initial vertices may produce different partitions, even with different cardinality. To take into account this fact, we can modify the algorithms as follows. Observe that if two vertices generate the same first convex polygon, then the final partition will be the same too. So, we only need to generate a partition with a new initial vertex if the first convex polygon generated (before the merging process) is different from the first polygons of the partitions already generated. After reading the vertices of the polygon, we now do the vertex dependency process described below in which *OS* denotes the set with the minimal decompositions obtained by the algorithm, *CARD* their cardinality, D_s denotes the convex polygon decomposition obtained by the algorithm when the initial vertex considered is v_s and P_s is the first convex polygon obtained in D_s.

Vertex dependency process
1. $OS \leftarrow \emptyset$; $CARD \leftarrow \infty$; $s \leftarrow 1$.
2. **While** $s \leqslant n$ **do**
2.1. Decompose the polygon into convex polygons using v_s as initial vertex.
2.2. **If** $|D_s| = CARD$ **then** $OS \leftarrow OS \cup D_s$ **else if** $|D_s| < CARD$ **then** $OS \leftarrow \{D_s\}$; $CARD \leftarrow |D_s|$.
2.3. $s \leftarrow s + 1$; *NewFirstPolygon* $\leftarrow$ *false*.
2.4. **While** not *NewFirstPolygon* **and** $s \leqslant n$ **do**
2.4.1. **Generate** P_s.
2.4.2. **If** $P_s \neq P_{s-1}$ **then** *NewFirstPolygon* $\leftarrow$ *true* **else** $s \leftarrow s + 1$.

At the end of the process, *OS* contains all the minimal decompositions obtained by the algorithms. Algorithm i, $i = 1, 2, 3$, together with the vertex dependency process, will be called Algorithm i/D.

3. The merging process

A diagonal d is said to be *essential for vertex* v if removal of d creates a piece that is non-convex at v. Observe that if d is essential for v then it must be incident to v and v must be reflex. A diagonal that is not essential for any vertex is called *inessential*.

The algorithms presented above produce good partitions but like other partitioning algorithms (see [21]), the partitions may sometimes contain inessential diagonals, i.e., two convex polygons sharing a diagonal can be merged to generate a single one. To prevent this, we have developed a merging process which can be used after any partitioning process producing inessential diagonals. Everyone of the diagonals of the partition is considered in order and it is checked whether it can be removed, taking into account that if the polygons P_s and P_t sharing a given diagonal have already been merged to other polygons P_s^* and P_t^* then the polygons to be considered when checking whether that diagonal can be removed are the merged

polygons $P_s \cup P_s^*$ and $P_t \cup P_t^*$, and not the single polygons P_s and P_t sharing the diagonal. Algorithm i, $i = 1, 2, 3$, together with the merging process will be called Algorithm i/M. The steps of the merging process are given below.

We will use the following lists: (1) LLE is a list containing the diagonals of the partition, (2) for every vertex v_j, LPv_j is a list containing pairs (k, v_r) where k is the index of a polygon containing v_j as one of its vertices and v_r is the next vertex to v_j in that polygon k, but these pairs are in LPv_j only if v_r is not the consecutive vertex of v_j in the initial polygon, i.e., if $v_j v_r$ is a diagonal, (3) LDP is an ordered list of boolean flags, $LDP[i] = true$ meaning that the polygon with index i is one of the definitive polygons of the partition after the merging process and (4) LUP is an ordered list of integer, $LUP[i] = j$ meaning that the polygon with index i is part of the polygon with index j. Let m be the number of diagonals in the partition to which we want to apply the merging process and NP the number of convex polygons used in the merging process.

Merging process

1. $NP \leftarrow m + 1$.
2. $LDP[i] \leftarrow true$, $LUP[i] \leftarrow i, i = 1, \ldots, NP$.
3. **For** $j = 1$ to m **do**
   - 3.1. $(v^{(s)}, v^{(t)}) \leftarrow LLE[j]$.
   - 3.2. **If** $(|LPv^{(s)}| > 2$ and $|LPv^{(t)}| > 2)$ **or** $(|LPv^{(s)}| > 2$ and $v^{(t)}$ is convex) **or** $(|LPv^{(t)}| > 2$ and $v^{(s)}$ is convex) **or** ($v^{(s)}$ is convex and $v^{(t)}$ is convex) **then**
     - 3.2.1. $j_2 \leftarrow v^{(t)}; i_2 \leftarrow v^{(s)}; j_3 \leftarrow Next[P_j, v^{(t)}]$; $i_1 \leftarrow Previous[P_j, v^{(s)}]$.
     - 3.2.2. **Find** in $LPv^{(t)}$ the only pair $(u, v^{(s)})$ containing $v^{(s)}$.
     - 3.2.3. $j_1 \leftarrow Previous[P_u, v^{(t)}]; i_3 \leftarrow Next[P_u, v^{(s)}]$.
     - 3.2.4. **If** $ang\{(i_1, i_2, i_3), (j_1, j_2, j_3)\} \leqslant 180^\circ$ **then** *(*see Fig. 4*)*
       - $\bullet NP \leftarrow NP + 1$.
       - **Write** as NPth polygon the polygon obtained merging the polygons with indices $LUP[j]$ and $LUP[u]$.
       - $\bullet LDP[j] \leftarrow false; LDP[u] \leftarrow false$; $LDP[NP] \leftarrow true$.
       - $\bullet LUP[j] \leftarrow NP; LUP[u] \leftarrow NP$.

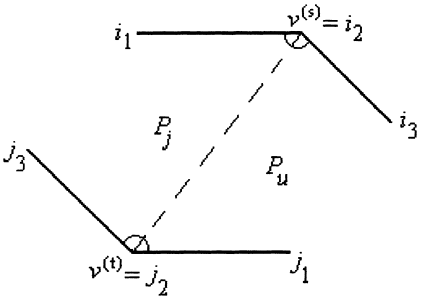

Fig. 4. Check to see whether the diagonal $v^{(s)}v^{(t)}$ can be removed.

- $\bullet$**For** h = 1 to $NP - 1$ **do**
  If $LUP[h] = j$ **or** $LUP[h] = u$ **then** $LUP[h] \leftarrow NP$.

To evaluate the efficiency of the algorithms, we need bounds on the best possible partition by diagonals. Let OPT be the fewest number of convex subpolygons into which a polygon may be partitioned. If the polygon has r notches, then $OPT \geqslant \lceil r/2 \rceil + 1$ since at most two reflex angles can be resolved by a single diagonal (see Fig. 5(a)). To obtain an upper bound, observe that to resolve any reflex angle we need at least one diagonal d incident to the notch v producing the reflex angle. But d resolves the reflex angle if and only if d is in the cone formed by extending the polygonal edges ending at v toward the interior of the polygon (see Fig. 6(a)). However, it is not sure whether for a given notch there will be a visible vertex within the corresponding cone allowing us to draw such a diagonal (see Fig. 6(b)). What we can guarantee in those cases is that v can be resolved with at most two diagonals, the ones obtained connecting v to the visible vertices to the immediate right and left of its cone. So $OPT \leqslant 2r + 1$. As Fig. 5(b)) shows, that bound can be attained for some polygons.

Now, since at most two diagonals are essential for any reflex vertex (see [11]), it follows that any decomposition without inessential diagonals is within four times of the minimum decomposition $(2r + 1 < 2r + 4 \leqslant 4(\lceil r/2 \rceil + 1))$. In particular, this holds for the partitions generated with Algorithm i/M, $i = 1, 2, 3$. Nevertheless, we think that the bound on the absolute quality of an algorithm is not a good representative measure of its performance since the bound must take the worst

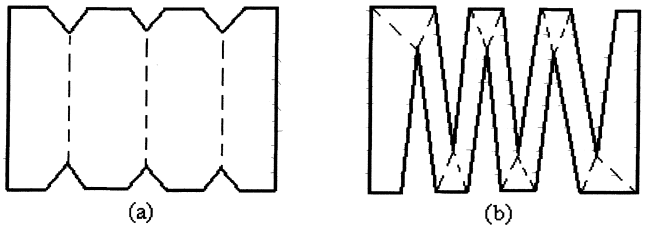

Fig. 5. Optimal convex decompositions. In (a), $OPT = \lceil r/2 \rceil + 1$: $r = 6$, 4 pieces. In (b), $OPT = 2r + 1$: $r = 6$, 13 pieces.

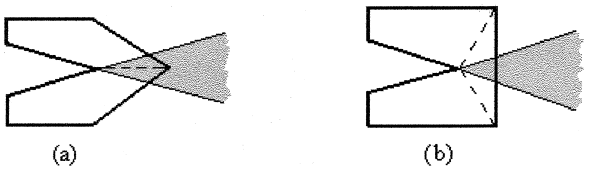

Fig. 6. A reflex angle and its cone. In (a) one diagonal can resolve the reflex angle. In (b) two diagonals are required.

possible case into account, which hardly appears in practice. Computational studies as the one presented in the next section may better help to give an idea of the performance of an algorithm.

4. Computational experiments

To the extent of our knowledge, there are no articles dealing with convex decompositions which include computational experiments. This is due, in part, to the fact that some of the algorithms proposed are *theoretical*, their only purpose being to offer low complexity but that in practice are extremely difficult, if not impossible, to implement in any language programming (see [2]). Another reason is that there are no published test problems with which to compare running times and cardinality of the partitions when the algorithms are not exact. On the other hand, to generate polygons is a tedious task.

To carry out our study, we have generated in all 250 polygons, fifty for everyone of the number of vertices $n = 50$, 75, 100, 125 and 150, utilizing a program that allows to draw the polygon on the PC screen and store the co-ordinates of the vertices in an ASCII file. We implemented that program in such a way that a polygon has to fit on a single screen ($[0, 640] \times [0, 480]$ pixels). So the polygons have complicated shapes, making them appropriate to test decomposition algorithms. Notice that the complexity of a polygon not only depends on its number of notches but also on its shape. This intuitive fact is corroborated by our empirical studies where, for instance, using the same algorithm a polygon with 100 vertices, 43 of them notches, was decomposed into 49 polygons while another polygon with the same number of vertices, 47 of them notches, was decomposed into only 31 polygons. These test problems can be obtained under request [7].

We have decomposed the polygons using the twelve different algorithms obtained by combining the three basic procedures *MP1*, *MP2* and *MP3* to generate the polygons of the partitions and the merging and vertex dependency processes. For the sake of brevity, we do not present the results individually for everyone of the 250 polygons but grouped according to the number of vertices. Furthermore, we only present that sumarized information for 5 of the algorithms since they are sufficient to explain our conclusions. All the codes were implemented on Turbo Pascal V. 6.0 [8] and run on a PC with a processor Intel Pentium with a CPU speed of 133 MHz.

In Table 1 are shown the results obtained with Algorithms 1, 2 and 3. For everyone of the groups of polygons the minimum, mean, maximum and

Table 1
Computational results for Algorithms 1, 2 and 3 (the given values are (minimum, mean, maximum; standard deviation))

Vertices	Notches	Polygons	Time
Algorithm 1			
50	(6, 15.72, 25; 4.84)	(8, 17.96, 31; 5.93)	(0, 0.009, 0.06; 0.02)
75	(16, 28.10, 38; 4.75)	(17, 29.86, 45; 6.38)	(0, 0.017, 0.11; 0.02)
100	(25, 40.08, 50; 5.28)	(22, 41.34, 64; 7.05)	(0, 0.023, 0.06; 0.02)
125	(34, 51.76, 62; 6.25)	(34, 54.80, 69; 8.15)	(0, 0.040, 0.06; 0.02)
150	(48, 64.62, 77; 6.48)	(47, 68.70, 93; 10.22)	(0, 0.061, 0.16; 0.03)
Algorithm 2			
50	(6, 15.72, 25; 4.84)	(5, 15.76, 29; 5.66)	(0, 0.009, 0.06; 0.02)
75	(16, 28.10, 38; 4.75)	(14, 27.06, 40; 5.72)	(0, 0.022, 0.06; 0.02)
100	(25, 40.08, 50; 5.28)	(22, 37.68, 59; 6.73)	(0, 0.028, 0.11; 0.03)
125	(34, 51.76, 62; 6.25)	(24, 48.74, 67; 7.67)	(0, 0.042, 0.11; 0.02)
150	(48, 64.62, 77; 6.48)	(45, 62.12, 82; 9.11)	(0, 0.062, 0.11; 0.02)
Algorithm 3			
50	(6, 15.72, 25; 4.84)	(5, 15.50, 28; 5.44)	(0, 0.014, 0.06; 0.02)
75	(16, 28.10, 38; 4.75)	(14, 26.94, 40; 5.52)	(0, 0.019, 0.06; 0.02)
100	(25, 40.08, 50; 5.28)	(22, 37.66, 58; 6.67)	(0, 0.028, 0.06; 0.02)
125	(34, 51.76, 62; 6.25)	(24, 48.74, 67; 7.67)	(0, 0.039, 0.11; 0.02)
150	(48, 64.62, 77; 6.48)	(45, 61.92, 81; 8.87)	(0, 0.061, 0.11; 0.02)

standard deviation of the number of notches, number of polygons of the decomposition and CPU time are given. Notice that the running times are similar for the three algorithms, but the decompositions obtained by Algorithms 2 and 3 are much more better than those obtained by Algorithm 1. For instance, the improvement when using Algorithm 3 as contrasted with Algorithm 1 attains a considerable

$$\mathrm{Imp}^{50}_{\mathrm{Alg1,Alg3}} = \frac{(\mathrm{No.pol})_{\mathrm{Alg1}} - (\mathrm{No.pol})_{\mathrm{Alg3}}}{(\mathrm{No.pol})_{\mathrm{Alg1}}} 100 = 13.69\%$$

for the polygons with 50 vertices.

Table 2 shows the results obtained when using Algorithm 3/M. As we see, the merging process is very useful since it reduces the number of polygons considerably. For instance, $\mathrm{Imp}^{150}_{\mathrm{Alg3,Alg3/M}} = 16.69\%$. Similar improvements are obtained when Algorithms 2 and 2/M are compared. When Algorithm 1 and 1/M are compared the improvements are higher, around 25%. In fact, Algorithm 1/M is as good as Algorithm 2/M or 3/M, the improvements of one of them as contrasted with one another being under 4%, that is, the merging process makes the use of *MP1* not as bad as it was without the merging process. On the other hand, it is natural for the merging process to be more effective with *MP1* since it is the procedure producing worse partitions. These algorithms are fast. For instance, the mean time for Algorithm 3/M for polygons with 150 vertices is 0.205 seconds, 0.144 of which are for the merging process (although most of that CPU time is employed in opening and closing auxiliary files and not in calculations). For Algorithm 1/M the CPU times are higher, the mean for polygons with 150 vertices being 0.286 seconds, 28.32% slower than Algorithm 3/M. This is due to the fact that the merging process has to check more diagonals when applied to partitions obtained with *MP1*.

Of course, the best partitions of the polygons are obtained when the merging process is used together with the vertex dependency process. Nevertheless, the improvements upon Algorithm *i*/M are slight, varying from $\mathrm{Imp}^{150}_{\mathrm{Alg1/M,Alg1/M/D}} = 2.70\%$ to $\mathrm{Imp}^{100}_{\mathrm{Alg3/M,Alg3/M/D}} = 8.38\%$, whereas the running times increase very much, so their use is not recommended. In Table 3, where the results

Table 2
Computational results for Algorithm 3/M (the given values are (mean; standard deviation))

Vertices	Notches	Polygons	Time	Merg. pol.	Merg. time
50	(15.72; 4.84)	(13.44; 4.39)	(0.038; 0.02)	(2.06; 1.73)	(0.023; 0.02)
75	(28.10; 4.75)	(23.14; 4.44)	(0.063; 0.03)	(3.80; 1.85)	(0.049; 0.03)
100	(40.08; 5.28)	(31.98; 5.22)	(0.097; 0.04)	(5.68; 2.36)	(0.075; 0.03)
125	(51.76; 6.25)	(40.74; 5.65)	(0.147; 0.04)	(8.00; 3.23)	(0.107; 0.04)
150	(64.62; 6.48)	(51.58; 6.33)	(0.205; 0.07)	(10.34; 4.02)	(0.144; 0.05)

Table 3
Computational results for Algorithm 3/M/D (the given values are (mean; standard deviation) or mean)

Vertices	Polygons	Time	Merg. pol.	Merg. time	No. partit.	No. best
50	(12.52; 4.35)	0.48	(1.78; 1.80)	0.28	(10.60; 3.22)	3.82
75	(21.30; 4.25)	1.24	(3.76; 2.03)	0.81	(15.96; 3.75)	3.26
100	(29.30; 5.64)	2.41	(5.24; 2.61)	1.63	(21.08; 3.68)	2.84
125	(37.62; 5.19)	4.16	(7.04; 3.17)	2.87	(26.02; 4.01)	2.22
150	(48.04; 6.81)	7.43	(9.76; 3.93)	5.19	(32.70; 4.15)	2.58

for Algorithm 3/M/D are shown, we can see that the mean time for decomposing polygons with 150 vertices is 7.43 seconds. This is due to the great number of different partitions (the mean for polygons with 150 vertices is 32.7) that the algorithm must carry out, although only 2 or 3 of them (the mean for polygons with 150 vertices is 2.58) are optimal.

Finally, the performance of the algorithms with the vertex dependency process but without the merging process is very bad since they need much more time and produce worse partitions than Algorithm *i*/M. For instance, $\mathrm{Imp}^{150}_{\mathrm{Alg1/D,Alg3/M}} = 22.31\%$ and $\mathrm{Imp}^{150}_{\mathrm{Alg3/D,Alg3/M}} = 9.15\%$.

We have also decomposed the polygons following the idea of Hertel and Mehlhorn's algorithm (see [11]), another simple practical algorithm for decomposing polygons into convex subpolygons without adding new vertices. The idea of that algorithm is basically this: triangulate the polygon to be decomposed and then remove inessential diagonals. To triangulate we have used the quadratic algorithm outlined in [18], chapter 1 and to remove inessential diagonals we have used the merging process described in the previous section. The results are summarized in Table 4. For everyone of the groups of polygons the mean and standard deviation of the number of notches, number of polygons, total CPU time and part of the CPU time employed in the merging process are given. As we can see, the partitions produced by Hertel and Mehlhorn's algorithm (AlgHM) are worse than those produced by the algorithms presented in this paper. Even the algorithms without the merging process and without the vertex dependency process produce partitions with lower cardinality (see Table 1). For instance, the improvement when using Algorithm 3 as contrasted with AlgHM is over 11% for everyone of the groups of polygons. For the algorithms which

Table 4
Computational results using Hertel and Mehlhorn's algorithm (the given values are (mean; standard deviation))

Vertices	Notches	Polygons	Time	Merg. time
50	(15.72; 4.84)	(18.32; 5.34)	(0.069; 0.02)	(0.068; 0.02)
75	(28.10; 4.75)	(31.78; 5.20)	(0.163; 0.02)	(0.158; 0.02)
100	(40.08; 5.28)	(43.88; 6.15)	(0.277; 0.03)	(0.267; 0.03)
125	(51.76; 6.25)	(56.60; 6.92)	(0.416; 0.03)	(0.407; 0.03)
150	(64.62; 6.48)	(70.06; 7.14)	(0.587; 0.04)	(0.564; 0.03)

use the merging process the improvements are higher, always over 25%. This is not surprising. The algorithms presented in this paper try to generate convex polygons with as many vertices as possible and so the partitions they produce have low cardinality. On the other hand, Hertel and Mehlhorn's algorithm does not follow any strategy with that aim apart from the one of the merging process, i.e., to try to remove in chronological order of their generation the inessential diagonals of the triangulation. Concerning the CPU times, although triangulating a polygon is quicker than decomposing it into convex subpolygons using any of the procedures *MP1*, *MP2* or *MP3*, the total CPU times using Hertel and Mehlhorn's algorithm are much higher than those of Algorithm *i*/M, $i = 1, 2, 3$ (and of course, than those of Algorithm *i*, $i = 1, 2, 3$). This is due to the fact that now the merging process has to try to remove the $n - 3$ diagonals required to triangulate any polygon with n vertices, whereas the number of diagonals generated with any of the procedures *MP1*, *MP2* or *MP3* and to be checked then by the merging process is much lower. For instance, the mean number of diagonals for polygons with 150 vertices using the procedure *MP3* is 60.92, whereas the number of diagonals of any triangulation is 147, that is, 58.55% less, and the corresponding savings in time when using Algorithm 3/M as contrasted with AlgHM are 74.48% for the merging process and 65.10% for the total time. So we can conclude that the algorithms presented in this paper are much better than the one of Hertel and Mehlhorn for both the cardinality of the partitions they produce and the CPU times they use.

5. Conclusions

In this paper we have presented different algorithms for decomposing a polygon into convex polygons without adding new vertices. They differ in the way the polygons of the partition are generated and in two additional processes, one of them to improve the partitions by merging the polygons of the partition whose union remains a convex polygon, and the other one to take into account the dependency of the partition on the initial vertex chosen to generate the first polygon of the partition. Computational experiments show that the procedures to generate convex polygons *MP2* and *MP3* produce partitions with similar cardinality and that they both are better than *MP1*. The procedure *MP3* has in addition the property that the partitions it produces have no diagonals joining two vertices which are not notches, which are known to be non-optimal. It is recommended the use of the merging process since it is a quick process and can reduce the number of polygons considerably. Moreover, since after the merging process the partitions do not contain unnecessary edges it is theoretically guaranteed that they are within four times the optimal solution, although the computational studies carried out show that the performance is in practice much better. The merging process described can be applied to any partition, not only to the ones produced by our algorithms. On the other hand, the vertex dependency process is time consuming and the number of polygons it reduces after the merging process is low, varying (from 1 to 3) depending on the number of vertices of the polygon (from 50 to 150). So the user must decide whether the great increase of time compensates the reduction of the number of polygons. A comparison between the algorithms presented in this paper and Hertel and Mehlhorn's algorithm, another simple practical algorithm, shows that the performance of the algorithms presented in this paper is better since the cardinality of the partitions they produce is lower and their CPU times are quicker.

For constrained location problems, where using techniques such as branch and bound the location problem has then to be solved in some (usually not all) of the subpolygons, we recommend the use of Algorithm 3/M, which can decompose a polygon with 150 vertices in 0.2 seconds. As a case study, in Fig. 7 is presented the decomposition given by that algorithm when the polygon considered is a non-convex polygon with 57 vertices, 27 of them notches, approximating the shape of the Autonomous Region of Murcia, a region in the south-east of Spain. The number of polygons of the partition is 20 and the CPU time was 0.047 seconds. When using Algorithm 3/M/D, the number of polygons was 19 and the CPU time 0.710 seconds.

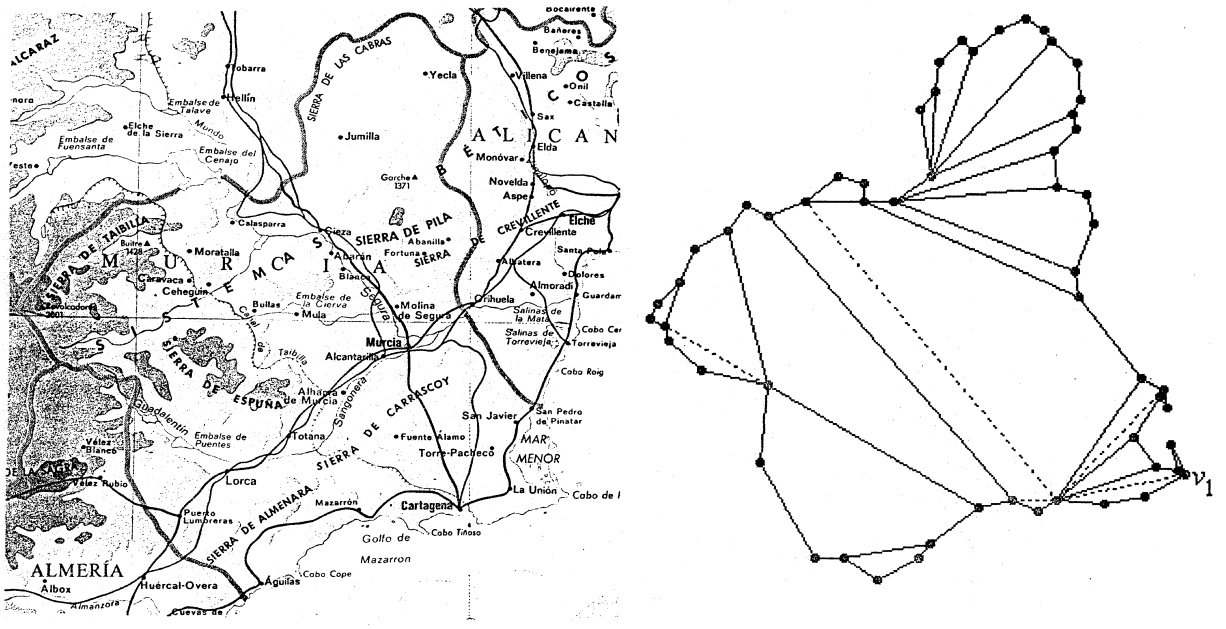

Fig. 7. Partition of the Autonomous Region of Murcia using Algorithm 3/M.

Observe that the algorithms we have presented can only be used to decompose polygons without holes. Nevertheless, in many problems it is necessary to consider holes inside the polygons. Decomposing a polygonal region with polygonal holes into the minimum number of convex subpolygons is known to be a problem NP-hard [13]. A direction for future research is to adapt the procedures generating the convex polygons of the partition and the merging process described in this paper to the problem with holes so as to design a quick practical algorithm producing partitions with low cardinality suitable to be used when optimal decompositions are not required. Another area for future research is to find rules to decide the order in which the diagonals of a partition should be checked by the merging process so as to remove as many diagonals as possible.

Acknowledgements

We wish to thank the referees for making several comments to improve the presentation of the paper. The comparison between our algorithms and Hertel and Mehlhorn's algorithm was suggested by one of the referees. This work has been supported by the Consejería de Cultura y Educación de la Región de Murcia under grant "COM-23/96 MAT. Ayud. Compl. Inv. 1996" and Fundación Séneca under grant PB/11/FS/97.

References

[1] B. Chazelle, Triangulating a simple polygon in linear time, Discrete and Computational Geometry 6 (1991) 485–524.

[2] B. Chazelle, D.P. Dobkin, Optimal convex decompositions. in: Computational Geometry, Elsevier Science Publishers, North-Holland, 1985, pp. 63–133.

[3] E.V. Denardo, Dynamic Programming: Models and Applications. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[4] Z. Drezner (Ed.), Facility Location. A Survey of Applications and Methods, Springer Series in Operations Research, Springer, New York, 1995.

[5] S.E. Elmaghraby, The concept of State in discrete dynamic programming, Journal of Mathematical Analysis and Applications 29 (1970) 523–557.

[6] H. Feng, T. Pavlidis, Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition, IEEE Transactions on Computers 24 (1975) 636–650.

[7] J. Fernández, L. Cánovas, B. Pelegrín, Two hundred and fifty nonconvex polygons, Internal Report, 1997.
[8] J. Fernández, L. Cánovas, B. Pelegrín, DECOPOL – Codes for decomposing a polygon into convex subpolygons, European Journal of Operational Research (O.R.S.E.P. section) 102 (1997) 242–243.
[9] J. Fernández, L. Cánovas, B. Pelegrín, Un algoritmo para la descomposición de polígonos en polígonos convexos, XXIII Congreso Nacional de Estadística e Investigación Operativa, Valencia (1997) 34.9-34.10.
[10] R.L. Francis, L.F. McGinnis, J.A. White, Facility Layout and Location: An Analytical Approach, 2nd ed., International Series in Industrial and Systems Engineering, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[11] S. Hertel, K. Mehlhorn, Fast triangulation of simple polygons, in: Proc. 4th Internat. Conf. Found. Comput. Theory, in Lecture Notes in Computer Science, vol. 158, Springer, New York, 1983, pp. 207–218.
[12] J.M. Keil, Decomposing a polygon into simpler components, SIAM Journal on Computing 14 (1985) 799–817.
[13] A. Lingas, The power of non-rectilinear holes, in: Automata, Languages and Programming (Aarhus, 1982), Springer, Berlin, 1982, pp. 369–383.
[14] W. Lipski, E. Lodi, F. Luccio, C. Mugnai, L. Pagli, On two-dimensional data organization II, Fundamenta Informaticae 2 (1979) 245–260.
[15] R.F. Love, J.G. Morris, G.O. Wesolowsky, Facilities Location: Models and Methods, North-Holland, New York, 1988.
[16] W. Newman, K.J. Sproull, Principles of Interactive Computer Graphics, 2nd ed., McGraw-Hill, New York, 1979.
[17] J. O'Rourke, Art Gallery Theorems and Algorithms, International Series of Monographs on Computes Science, Oxford University Press, New York, 1987.
[18] J. O'Rourke, Computational Geometry in C, Cambridge University Press, Cambridge, 1994.
[19] J. O'Rourke, Computational geometry column 29, International Journal of Computational Geometry and Applications 6 (1996) 507–511.
[20] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer, New York, 1985 (Corrected and expanded second printing, 1988).
[21] B. Schachter, Decomposition of polygons into convex sets, IEEE Transactions on Computers 27 (1978) 1078–1082.