CMU-CS-82-121

The Power of Triangulation: Applications to Problems of Visibility and Internal Distance.

Bernard Chazelle

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

March 1982

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-81-K-1539.

Abstract

It is well-known that the complexity of performing operations on a set depends heavily on the structure which we are allowed to put into its representation. For example, searching through a sequence of numbers can be performed more efficiently if the numbers appear in sorted order. In this paper, we take, as a case-study, the class of problems involving a simple N-gon P and, making the assumption that in addition to the usual description of the boundary of P, an arbitrary triangulation is also available, we investigate the computational power gained from having this additional information. Among other results, we give a very simple, optimal algorithm for computing the area visible from an arbitrary point in P. We also present several optimal algorithms for computing the *internal distance* between two points in P. Recall that the internal distance between A and B is defined as the length of the shortest path *inside* P between A and B.

1. Introduction

The complexity of problems that operate on fixed objects is highly dependent on the amount of preprocessing allowed in the objects' representation. As illustrated in the well known paradigm *searching .vs. sorting*, the mere availability of an order among keys cuts down the complexity of searching from linear to logarithmic. In numerical analysis, preconditioning a sparse matrix is standard procedure in order to facilitate the computation of its inverse. In general, the crucial issue is to balance costs and gains of preprocessing so as to optimize the overall performance. Few areas of computer science are free of this type of trade-offs and, in particular, this concern is recurrent in computational geometry, operations research, and in the study of data structures or data bases.

The first area mentioned, computational geometry, provides a good example of a structure, i.e., the *Voronoi diagram*, easy to construct efficiently, while one of the most powerful tools at our disposal for solving closest-point problems [SH77]. Unrelated, .yet equally effective results have shown that convex figures lend themselves to speedier algorithms than arbitrarily-shaped objects [CH80,CD80,DK81]. Consequently, an attractive approach to handle non-convex figures is to decompose them into their convex parts, then apply to these the efficient methods known for convex objects [CH80,FS81,GJ78,SC78,SV80,TO80]. We pursue this endeavor in this paper, and investigate the existence of efficient algorithms for various problems, assuming that in addition to the usual boundary description of a polygon, an arbitrary triangulation is also available. It is standard to define a triangulation of a polygon as a convex decomposition which does not introduce new vertices[1] [GJ78]. For our purposes, however, we can relax this definition and allow the vertices of the triangulation to lie anywhere on the boundary of the polygon. The only provision to make is that the total number of points used in the triangulation is linear in the number of vertices of the polygon. Note that this is always true with the standard, more restrictive definition. We also observe that it is easy, given a convex decomposition of a polygon, to derive a triangulation in linear time. It is then apparent that it is only for simplicity that we choose to be supplied with a triangulation rather than a more general convex decomposition of the polygon.

With this additional information in hand, we are able to describe a very simple, yet optimal algorithm for computing the area visible from any point inside a polygon. We also present several optimal algorithms for computing the *internal distance* between two points inside a polygon. Recall that the internal distance is defined as the shortest distance a person might travel from one point to the other, while remaining within the boundary of the polygon.

[1] i.e., where all the vertices in the decomposition are vertices of the polygon.

Next we introduce our notation, before proceeding with the description of the algorithms. Let P be a *simple*[2] polygon with vertices $v_1,...,v_N$ in clockwise order. We assume the existence of a triangulation T of P, defined as a set of non-overlapping triangles whose union is exactly P, and whose summits are taken in the set $\{v_1,...,v_N\}$. The edges of the triangulation which are not edges of P are called *interior* edges. As mentioned above, we may choose to allow the summits to lie anywhere on the boundary of P, provided that the total number of vertices in the triangulation does not exceed the number of vertices of P, up to within a constant factor. In this case, we may, for simplicity, rename the vertices of P so that the list $v_1,...,v_N$ gives a clockwise description of *all* the vertices appearing on the boundary of the triangulation.

Observing that a triangulation forms the embedding of a planar graph, we choose a *DCEL* representation of this graph as our basic working structure [MP78]. Recall that a *DCEL* is simply a handy data structure, obtainable in linear time from any standard adjacency representation, which in particular, allows one to traverse the boundary of each face in clockwise order and list the faces encountered on the left-hand side during the traversal. Roughly, to each edge e of the graph is assigned a 6-field node containing the names of the two endpoints in some specified order, as well as the two adjacent faces and the names of each of the edges first encountered in traversing these faces in clockwise order, starting at the endpoints of e.

Note that several algorithms are available in the literature for computing an arbitrary triangulation of an N-gon. The best performance achieved so far is $O(N\log N)$ time [GJ78,CH82], but is yet unknown to be optimal or not.

2. Visibility problems

A problem which arises frequently in graphics concerns the elimination of hidden lines from a two- or three-dimensional scene [NS79]. In two dimensions, the problem reduces to computing the sets of points that are visible from a given point inside a polygon P. Linear algorithms for this problem already exist [CH80,EA81], but they involve complicated stack manipulations which become unnecessary, once a triangulation is made available. The problem can be formulated as follows:

> *Given a simple polygon P and a point M inside P, the locus of points V such that the segment MV lies entirely in P is a simple polygon $V(M)$. Compute a clockwise description of the boundary of $V(M)$ (fig.5).*

We can regard the triangles of T as forming the nodes of a graph G, whose edges join the pairs of triangles with a common edge (i.e., an interior edge) (fig.1). As shown in Lemma 1, the absence of interior faces ensures that the graph G is actually a tree.

[2] A polygon is said to be *simple* iff only adjacent edges intersect.

[FIGURE 1]

Figure 1: *The triangulation T and the dual graph G.*

Lemma 1: G is a tree.

Proof: It suffices to show that for any pair of triangles t_1,t_2 in the triangulation, there exists a unique path between t_1 and t_2 in G. The triangle t_1 partitions P into 4 parts. One is the triangle t_1 itself, the others being polygons adjacent to the edges of t_1 (note that some of these polygons may be reduced to a single edge). At any rate, exactly one of the three polygons contains the triangle t_2. Call U this polygon, letting u denote its edge adjacent to t_1 and t be the triangle of T adjacent to u and lying in U (fig.2). Since the triangulation of P also provides a triangulation of U, and its associated graph G_u is a subgraph of G, we can see that if there is a unique path in G_u from t to t_2, there is also a unique path in G from t_1 to t_2. Therefore we can prove the lemma by induction on the number of vertices. □

[FIGURE 2]

Figure 2: *Proving that G is a tree.*

Let e be any interior edge of the triangulation, and let M be any point inside P. Letting t denote the triangle of T which contains M, we can define G(M,e) as the unique subtree of G emanating from e, which does not contain t (fig.3).

[FIGURE 3]

Figure 3: *The subtree G(M,e)*

We are now in a position to give an algorithm for computing the visibility polygon V(M). To facilitate our task, we introduce the function **VISIB**, defined as follows: let e^* be any segment lying entirely on the edge e. Remove from T all the triangles which do not belong to G(M,e), and call Q the resulting polygon. We define **VISIB** (M,e^*) as the part of Q which is visible from M *through* the window e^*. More precisely, **VISIB** (M,e^*) is the set of points u in Q such that the only intersection of Mu with the boundary of Q takes place at e^* (fig.4). Let a,b,c be the vertices of the triangle in Q adjacent to e with $e=ab$ and $e^*=a^*b^*$. We define A (resp. B) as the intersection of the polygonal line $\{bc,ca\}$ with the infinite line passing through Ma^* (resp. Mb^*). It is now straightforward to compute the function **VISIB** recursively.

VISIB (M,e^*)

```
if $e^*$ lies on the boundary of $P$
    then
        return ({$e^*$})
else
    Determine the points $c,A,B$.
    if $c$ lies between $A$ and $B$
        then
            $V \leftarrow$ VISIB ($M,Bc$)
            $V \leftarrow V \cup$ VISIB ($M,cA$)
        else
            $V \leftarrow$ VISIB ($M,AB$)
    return ($V$) - (fig.4)
```

[FIGURE 4]

Figure 4: *Computing the area VISIB (M,e^*).*

To complete the computation of V(M), it suffices to determine the triangle of T where M lies - which can be done in O(N) time - then apply the previous procedure with respect to its three edges.

VISIBILITY (P,M)

```
Let $e_1,e_2,e_3$ be the edges in clockwise order of the
triangle of $T$ which contains $M$. Initially V($M$) = $\emptyset$.
for i = 1,2,3
    begin
    V($M$) $\leftarrow$ V($M$) $\cup$ VISIB ($M,e_i$)
    end
```

See an illustration in fig.5. Note that, as described, the procedure reports the boundary of V(M) in clockwise order, except for the *ray-edges* of V(M), i.e., the segments collinear with M, which are omitted. A single pass through the list V(M), however, will be sufficient to add the missing segments, and we need not elaborate. Using a DCEL representation of the triangulation ensures that each recursive step can be executed in constant time, from which we can conclude:

Theorem 2: Given a simple N-gon P along with an arbitrary triangulation of P, it is possible to compute the visibility polygon from any point M inside P, in O(N) time.

The main advantage of this algorithm is that it avoids the complicated stack manipulations of [CH80] and [EA81]. The reader may convince himself/herself/itself that the algorithm could be rewritten without greater difficulty in order to deal directly with a more general convex decomposition (i.e., without first converting it into a triangulation). This may be an interesting alternative if one is willing to exploit the fact that searching among the edges of a convex polygon can be done in logarithmic time, using a Fibonacci search-based strategy [CH80,CD80]. We would not recommend this approach in practice, however, unless the size of the problem was particularly gigantic. Once again, we leave substantiating these digressions to the attention of the reader.

[FIGURE 5]

Figure 5: *The visibility polygon V(M).*

3. Applications to internal distance problems

3.1. The car-racing problem

What is the shortest trajectory of a racing car on a given circuit? More precisely, the problem which we address in this section can be expressed as follows:

Given a simple polygon P and two arbitrary points A and B in P, find the shortest path inside P between A and B (fig.6).

[FIGURE 6]

Figure 6: *The internal path between A and B, IP(A,B).*

This shortest path is called the *internal path* between A and B, denoted IP(A,B), and its length, |IP(A,B)|, is called the *internal distance* between A and B (fig.6). To have a visual representation of IP(A,B), one can imagine a rubber band inside P tightly stretched between A and B. In [SM77], Shamos suggests an O(N^2) algorithm for computing IP(A,B). The method consists essentially of computing all pairs of vertices visible from each other, in O(N^2) time, so as to form the so-called *viewability graph* of P. We next add weights to the graph by associating to each edge the Euclidean distance between its endpoints. Computing an internal path is now equivalent to finding the shortest path between two vertices of a graph with N vertices, which can be done in O(N^2) time. Of course, we assume in this case that both A and B are vertices of P. We will next show how the use of a triangulation permits us to compute the internal path in O(N) time, without even having to restrict the points to be on the boundary of P. Note that since we know how to compute a triangulation of an N-gon in O($N\log N$) time, this result constitutes a significant improvement.

For the time being, we will assume that both A and B are vertices of P. We will see later on how we can easily dispense with this requirement. If A and B are vertices of the same triangle of T, it is clear that IP(A,B) = AB, so we may assume that this is not the case. In the following, we will say that an interior edge of T is *AB-crossing* if its endpoints u,v are such that A,u,B,v appear this order around the boundary of P. Let P^* be the polygon resulting of the removal from T of all the edges that are not AB-crossing (fig.7). We first prove a few technical lemmas.

[FIGURE 7]

Figure 7: *The transformation of P into P^**

Lemma 3: The internal path between A and B in P is identical to the internal path between A and B in P^*.

Proof: It suffices to show that IP(A,B) can only intersect AB-crossing edges. To see that, suppose that it intersects an interior edge e which is not AB-crossing. Since e partitions P into two polygons, one of them does not contain B, therefore IP(A,B) crosses e at least twice (once in each direction). If A^* (resp. B^*) is the first (resp. second) intersection, going from A to B, replacing the part of IP(A,B) from A^* to B^* by the segment A^*B^* will shorten the length of IP(A,B), which leads to a contradiction. □

Lemma 4: The internal path between A and B intersects every interior edge of P^* exactly once, and intersects no other edge in T.

Proof: The proof of Lemma 3 shows that IP(A,B) cannot intersect any interior edge more than once. On the other hand, we can easily prove by induction that since every interior edge of P^* partitions this polygon into two parts, neither of which contains both A and B, it must intersect IP(A,B) at least once. Putting this result together with Lemma 3 completes the proof. □

It is easy to compute P^* in O(N) time. To do so, consider every interior edge of T in turn, and if it is not AB-crossing, remove it from T along with the dangling sub-polygon, just created, that does not contain A or B. Let $L = \{a_1b_1,...,a_pb_p\}$ be the interior edges of P^*, as they appear from A to B (fig.7), i.e., in the order in which they intersect IP(A,B) (Lemma 4). Note that it is straightforward to obtain L in O(N) time, once P^* has been computed. From now on, the term IP(x,y), with x,y, vertices of P^*, refers to the internal path between x and y with respect to either P or P^*. This is legitimate since the two paths are identical, as a simple generalization of Lemma 3 readily shows.

[FIGURE 8]

Figure 8: *Computing IP(A,B* iteratively.)

Lemma 5: For any i; $1 \leq i \leq p$, there exists a vertex v of P^* such that IP(A,a_i)= IP(A,v) ∪ U and IP(A,b_i)= IP(A,v) ∪ W, where U and W are two convex, non-intersecting polygonal lines turning their convexity against each other, and running from v to a_i and b_i, respectively (fig.8).

Proof: Let C_1 and C_2 be two oriented curves originating at the same point. To carry the analogy with internal paths, we may further assume that neither is self-intersecting; we say that C_1 and C_2 have a *proper crossing* if, as we follow C_1 from its starting point, we encounter a point where C_2 intersects C_1, and actually switches from one side to the other. Fig.9.1 (but not fig.9.2) shows an example of a proper crossing.

[FIGURE 9]

Figure 9: *The notion of proper crossing.*

We next prove that for any three points A,B,C in P, the two paths IP(A,B) and IP(A,C) never have any proper crossings. Suppose that they did; let a be the first point (starting at A) where IP(A,B) and IP(A,C) cease to coincide, and let b denote the next intersecting point. Since IP(A,B) and IP(A,C) take distinct paths from a to b, we may re-route either one to the other, since they must have exactly the same length. Iterating on this process will eventually cause all proper crossings to disappear, which proves the above fact. We can now establish Lemma 5 by induction on i. The initial case being trivial, we may directly assume that the lemma is true for all indices from 1 to i. Since the $a_m b_m$'s are triangulation-edges, we necessarily have $a_i = a_{i+1}$ or $b_i = b_{i+1}$, say, $a_i = a_{i+1}$, wlog. Thus, considering the path IP(A,b_{i+1}), we observe that since it does not have any proper crossings with either IP(A,a_i) or IP(A,b_i),

1. It must pass through their common point v.

2. Its vertices between v and b_{i+1} are vertices of U and W.

From 1., it results that we may concentrate on the path IP(v,b_{i+1}) instead of IP(A,b_{i+1}), since we obviously have IP(A,b_{i+1}) = IP(A,v) $\cup$ IP(v,b_{i+1}). Next, we strengthen proposition 2. by proving that the vertices of IP(v,b_{i+1}) are vertices of U or W, but never of both at the same time. Indeed, suppose wlog that starting at v, the vertices of IP(v,b_{i+1}) are $t_1, t_2, \ldots$, with t_1 through t_m lying on U and t_{m+1} on W. It follows that the angle ($t_m t_{m+1}, t_m t_{m-1}$) is under 180 degrees, therefore there is an obvious shortcut for IP(v,b_{i+1}), avoiding t_m (fig.10), which leads to a contradiction. Thus there are now two basic cases to consider, depending on whether IP(v,b_{i+1}) takes its vertices in U or W. In the former case, v will be relocated further ahead on U, whereas it will stay unchanged in the latter. The details are straightforward, so we may consider the proof of the lemma as complete. □

[FIGURE 10]

Figure 10: *Minimality properties of IP(v,b_{i+1}).*

We are now ready to proceed with the algorithm for computing IP(A,B). The method involves computing IP(A,a_i) and IP(A,b_i), for $i = 1,\ldots,p$, which we can do iteratively by using the results of Lemma 5. The procedure being trivial for $i = 1$, we turn to the general step directly. As already mentioned, we have either $a_i = a_{i+1}$ or $b_i = b_{i+1}$, and we can assume wlog that $a_i = a_{i+1}$. Let $u_1,\ldots,u_\alpha$ (resp. $w_1,\ldots,w_\beta$) be the vertices of U (resp. W) from v to a_i (resp. b_i).

The half-plane delimited by $a_i b_i$ on the side where b_{i+1} lies is partitioned into $\alpha+\beta+1$ regions, themselves delimited by the lines passing through

$$w_{\beta-1}u_\beta, \ldots, w_1 w_2, vw_1, vu_1, u_1 u_2, \ldots, u_{\alpha-1}u_\alpha$$

With this order, the regions appear sorted along the segment $a_i b_i$ from b_i to a_i, so that we can find the region which contains b_{i+1} by testing each of them in turn in this order, until we are successful (fig.11). This corresponds to unfolding W and possibly folding over U. If b_{i+1} lies in a pencil of the kind ($w_{k-1}w_k, w_k w_{k+1}$), we must simply remove $w_{k+1},\ldots,w_\beta$ from W and reset β to $k+1$ and w_β to b_{i+1} (fig.11.1). If b_{i+1} lies in the pencil ($u_{j-1}u_j, u_j u_{j+1}$), however, we must set W to $u_j b_{i+1}$, remove ($v, u_1,\ldots,u_{k-1}$) from U and finally set v to u_j (fig.11.2). All the other cases are similar and call for no further explanation. Since none of the vertices removed in these operations will ever be examined again (Lemma 5), both IP(A,a_p) and IP(A,b_p), hence IP(A,B), will be computed in O(N) time.

[FIGURE 11]

Figure 11: *Updating U and W.*

We generalize this result by allowing both A and B to lie anywhere inside P, and not only on the boundary. Let R (resp. S) be the triangle where A (resp. B) lies. If $R=S$, the problem is solved since IP(A,B)$=AB$. Otherwise, we can compute the chain of triangles P^* in exactly the same way as described above.

Next, let v_iv_j be the interior edge of R which IP(A,B) crosses. We can replace R by the triangle v_iv_jA without altering the path IP(A,B). Applying the same treatment to S will make A and B become vertices of P^*, which allows us to call on the procedure described earlier to compute IP(A,B). In conclusion, we can state our main result:

Theorem 6: Let P be a simple N-gon, and assume that any triangulation of P is available. For any pair of points A,B in P, it is possible to compute IP(A,B), the internal path between A and B, in O(N) time, which is optimal in the worst case.

3.2. The all-internal-paths problem

The problem is to preprocess the polygon P so that a batch of queries of the kind:

What is the internal path between A and B?

can be answered optimally. The method described in the previous section grants an attractive balance between execution and preprocessing time, when only a few queries have to be handled at any given time. It is worst-case optimal, but not optimal in the strictest sense of the term, since all the vertices of P must always be examined for every query. As a result, the precomputation of all possible internal paths between vertices entails a prohibitive O(N^3) cost. The goal which we set forth here is to preprocess P so that the computation of IP(A,B) for any pair of vertices (A,B) requires only time proportional to the size of the output, i.e., the number of vertices in IP(A,B).

To achieve this goal, we use the concept of visibility introduced earlier. Let V(A) be the visibility polygon of A. If IP(A,B)$=AB$, B is a vertex of V(A), otherwise V(A) has a ray-edge (i.e., an edge vw such that v lies on Aw), with the property that vw separates A from B by intersecting IP(A,B). More precisely, vw is the unique edge of V(A) such that either A,v,B,w or A,w,B,v occur in clockwise order (fig.12). Since V(A) is star-shaped, and vw is a ray-edge which is traversed by IP(A,B), v must be the first vertex of IP(A,B) after A. Indeed, there would be a shortcut if IP(A,B) cut vw at any other point. Consequently, we have the relation:

$$\mathrm{IP}(A,B) = Av \cup \mathrm{IP}(v,B)$$

This motivates the introduction of the function $F(A,B)=B$, if $IP(A,B)=AB$, and $F(A,B)=v$ otherwise. Theorem 2 shows that if a triangulation of P is available, the visibility polygon $V(A)$ of each vertex A of P can be obtained in $O(N)$ time. The knowledge of $V(A)$ permits us to set up the array

$$D(A) = \{ F(A,v_i);\ i=1,\ldots,N \}$$

in $O(N)$ time, with $O(N)$ storage, from which we conclude:

Theorem 7: Let P be a simple polygon with N vertices. It is possible to preprocess P in $O(N^2)$ time, using $O(N^2)$ space, so that for any pair of vertices A,B, the path $IP(A,B)$ can be computed optimally, i.e., in time proportional to the size of the output.

Proof: Compute the N arrays $D(v_1),\ldots,D(v_N)$, forming an $N\times N$ matrix $\{F(v_i,v_j\}$, so that $IP(A,B)$ can be computed by retrieving $F(A,B)$ in constant time, and computing $IP(F(A,B),B)$ recursively. □

[FIGURE 12]

Figure 12: *The all-internal-paths problem.*

3.3. The internal-length problem

Imagine that an island with only inland communications is to be serviced by some utility (water tank, power station, fire house, police station, hospital, etc...). An interesting piece of information which may be needed is an upper bound on the internal path length between any pair of points.

Let A^*,B^* be the two vertices of P which form the longest path $IP(A^*,B^*)$. We call $|IP(A^*,B^*)|$ the *internal length* of P. It is easy to determine A^* and B^* by trying out all possible pairs of vertices and using the matrix F of the previous section, given that the longest path can always be assumed to be found between two vertices of the polygon. This leads to an $O(N^3)$ running time, which we can cut down to $O(N^2)$ by proceeding as follows:

Let $D(A,B) = |IP(A,B)|$. We will compute $D(A,B)$ iteratively by summing up partial distances obtained from F. In order to avoid duplicating computations, as soon as $D(A,B)$ is available, we backtrack along the path just followed in F to record the partial results. This ensures that, on average, one value $D(A,B)$ will be computed at every other step, which leads to an $O(N^2)$ algorithm.

INTDIST

- Initially, each $D(A,B)$ is set to -1 for $A \neq B$, and to 0 for $A = B$.

```
for all i ($1 \le i \le N$)
    for all j ($1 \le j \le N$)
        begin
            $Q \leftarrow \{v_i\}$
            $x \leftarrow v_i$
            while $D(x, v_j) = -1$
                begin
                $x \leftarrow F(x, v_j)$
                $Q \leftarrow Q \cup \{x\}$
                end
            if $Q$ has more than one element
                then
                Let $Q = \{x_1, \ldots, x_p\}$
                $L \leftarrow D(x_p, v_j)$
                for k = p-1,...,1
                    begin
                        $L \leftarrow L + |x_k x_{k+1}|$
                        $D(x_k, v_j) \leftarrow L$
                    end
        end
```

$D(A^*, B^*) = \text{Max}\ (\ D(v_i, v_j)\ |$ all pairs of vertices $v_i, v_j\ )$

Since we can compute a triangulation of P in $O(N \log N)$ time, we may conclude:

Theorem 8: It is possible to determine the internal length of a simple N-gon as well as the corresponding internal path in $O(N^2)$ time.

4. Conclusions

This paper has shown on the following examples how to use an arbitrary triangulation advantageously:

1. Computing the visibility polygon at any point inside an N-gon in $O(N)$ time.

2. Computing the internal path between any pair of points in an N-gon in $O(N)$ time.

3. Allowing $O(N^2)$ preprocessing, being able to compute any internal path with optimal performance.

4. Computing the internal distance of an N-gon and the associated internal path in $O(N^2)$ time.

All of these algorithms achieve significant improvements over previously known methods, since a triangulation of an N-gon can be computed in O(Nlog N) time. The improvements are to be measured either in terms of better performance (Problems 2,3,4) or in terms of added simplicity (Problem 1). We should also observe that it is yet unknown whether the triangulation algorithms available in the literature are optimal. Since, on the other hand, half of the algorithms which we have described in this paper are linear after triangulation, overall speed-ups would automatically result from the discovery of faster triangulating procedures.

This work was meant as a case-study and, of course, the list of possible improvements brought about by the use of a triangulation is not closed. Further research should attempt to enlarge the list given here, and carry the same approach with other preprocessing structures, whether geometrical or not.

REFERENCES

[CH80] Chazelle, B.M.

Computational geometry and convexity, PhD thesis, Yale University, 1980. Also available as CMU Tech. Rept. CMU-CS-80-150, July 1980.

[CH82] Chazelle, B.M.

How to divide a polygon fairly, CMU- Tech. Report, Carnegie-Mellon Univ., April 1982.

[CD79] Chazelle, B.M., Dobkin, D.P.

Decomposing a polygon into its convex parts, Proc. 11th SIGACT Symp., Atlanta, 1979, pp. 38-48.

[CD80] Chazelle, B.M., Dobkin, D.P.

Detection is easier than computation, Proc. 12th SIGACT Symp., Los Angeles, 1980.

[DK81] Dobkin, D.P., Kirkpatrick, D.G.

Fast detection of polyhedral intersections, Unpublished manuscript.

[EA81] El Gindy, H., Avis, D.

A linear algorithm for computing the visibility polygon from a point, Journal of Algorithms, 2, 186-197 (1981).

[FS81] Ferrari, L., Sankar, P.V., and Sklansky, J.

Minimal rectangular partitions of digitized blobs, Proc. 5th International Conference on Pattern Recognition, Miami Beach, Dec. 1981, pp. 1040-1043.

[GJ78] Garey, M.R., Johnson, D.S., Preparata, F.P., and Tarjan, R.E.

Triangulating a simple polygon, Info. Proc. Lett., Vol. 7(4), June 1978, pp. 175-179.

[MP78] Muller, D.E., Preparata, F.P.

Finding the intersection of two convex polyhedra, Theoret. Comput. Sci., 7 (1978), pp. 217-236.

[NS79] Newman, W.M., Sproull, R.F.

Principles of interactive computer graphics, McGraw-Hill, 2nd ed., 1979.

[SC78] Schachter, B.

Decomposition of polygons into convex sets, IEEE Trans. on Computers, Vol. C-27, 1978, pp. 1078-1082.

[SV80] Schoone, A.A., van Leeuwen, J.

Triangulating a star-shaped polygon, Tech. Rept. RUV-CS-80-3, University of Utrecht, April, 1980.

[SH77] Shamos, M.I.

Computational geometry, PhD thesis, Yale University, 1977.

[SH75] Shamos, M.I., Hoey, D.J.

Closest-point problems, 16th IEEE FOCS Symp. (1975), pp. 151-162.

[SM77] Shamos, M.I.

Problems in computational geometry, Carnegie-Mellon University, 1977.

[TO80] Toussaint, G.T.

Decomposing a simple polygon with the relative neighborhood graph, Proceedings of the Allerton Conference, Urbana, Illinois, October, 1980.

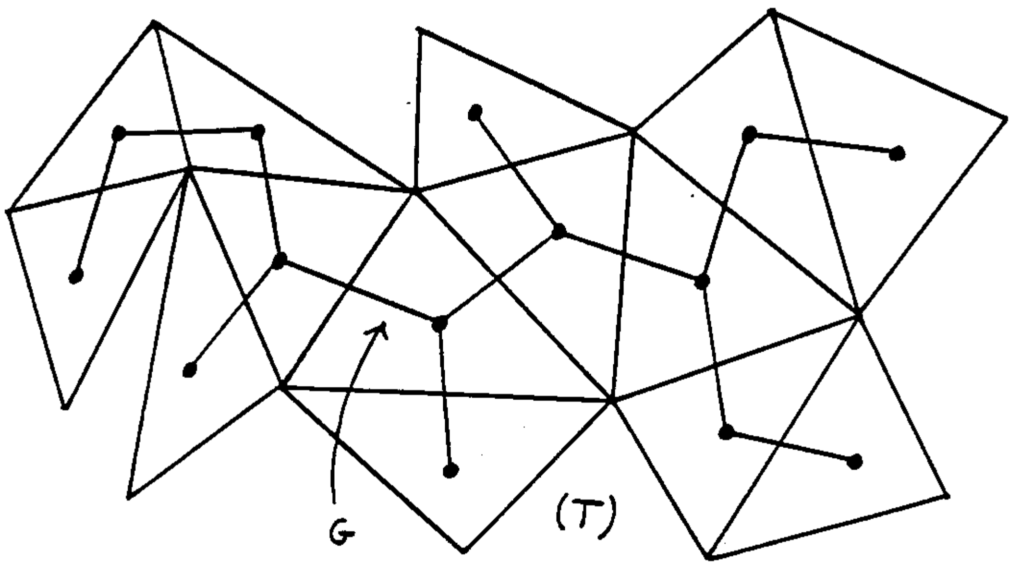


Figure 1

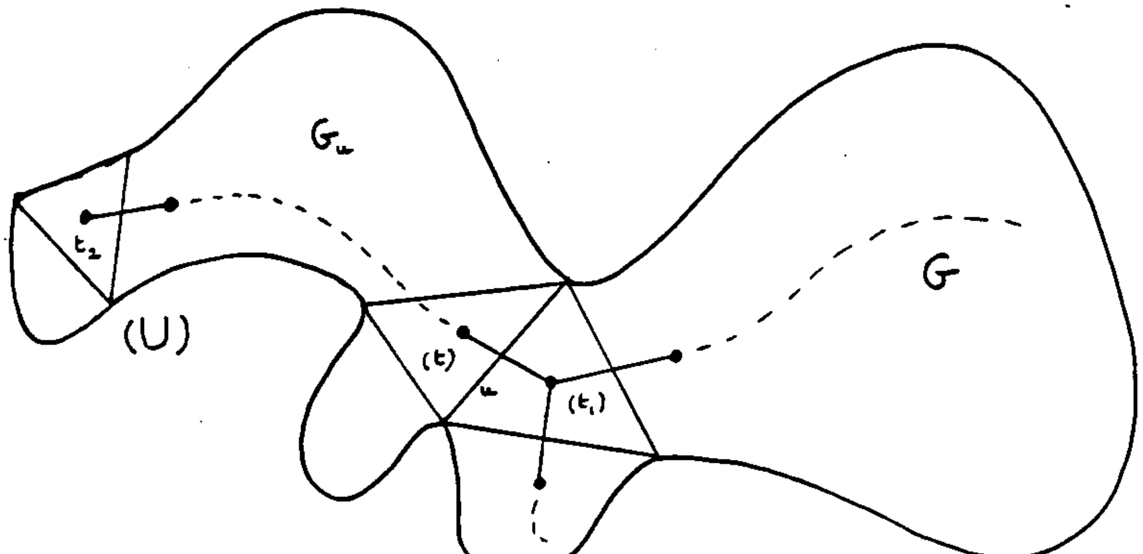
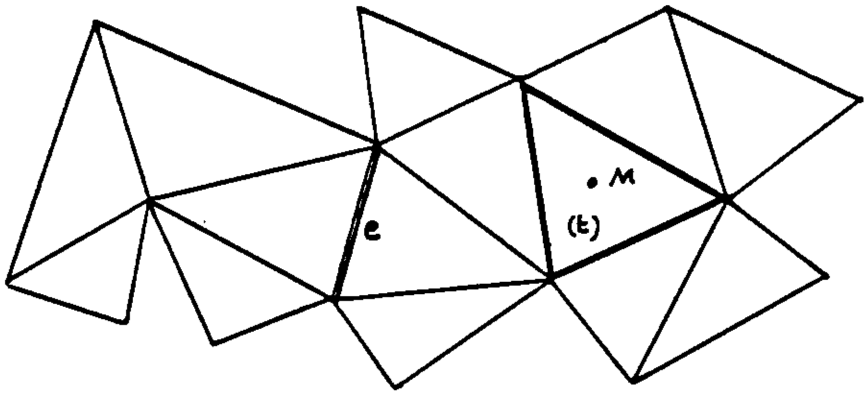


Figure 2



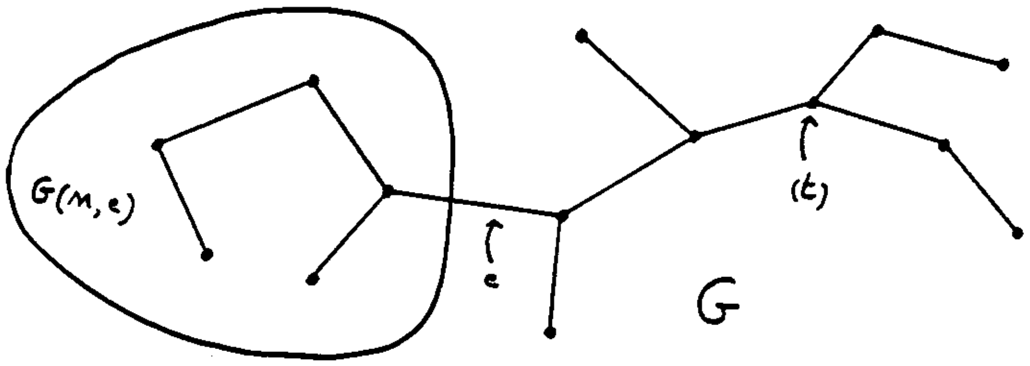


Figure 3

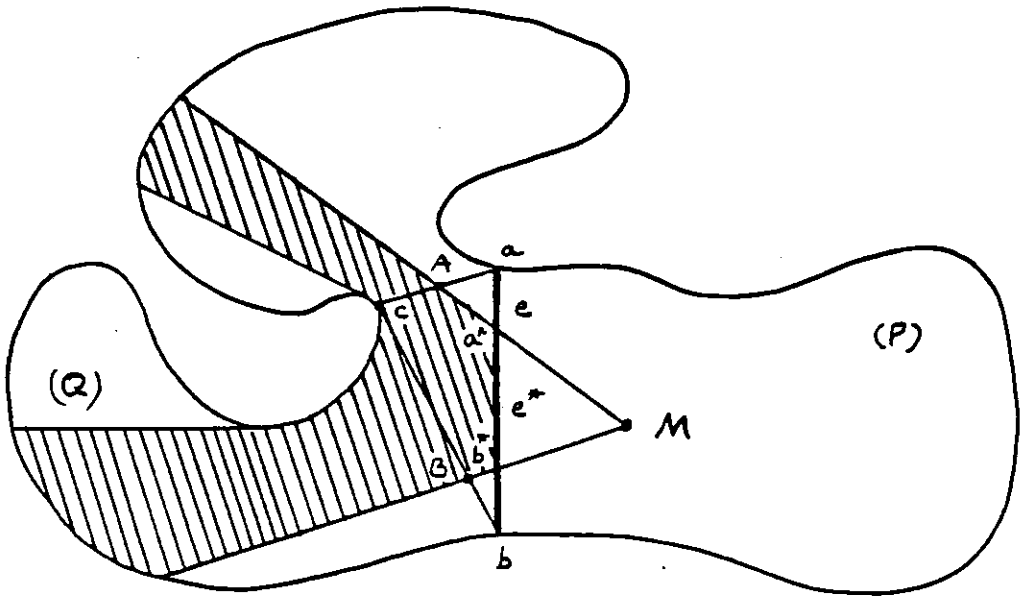


Figure 4

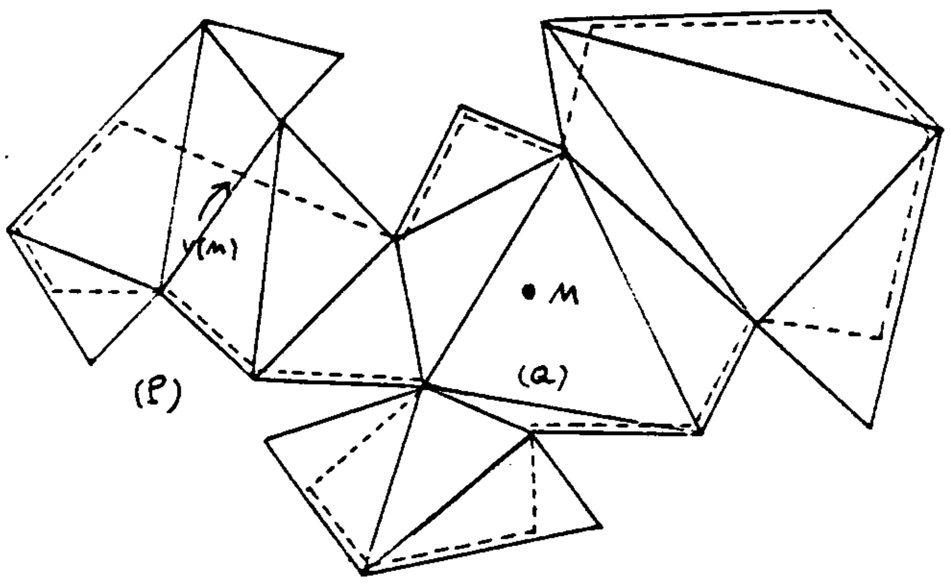


Figure 5

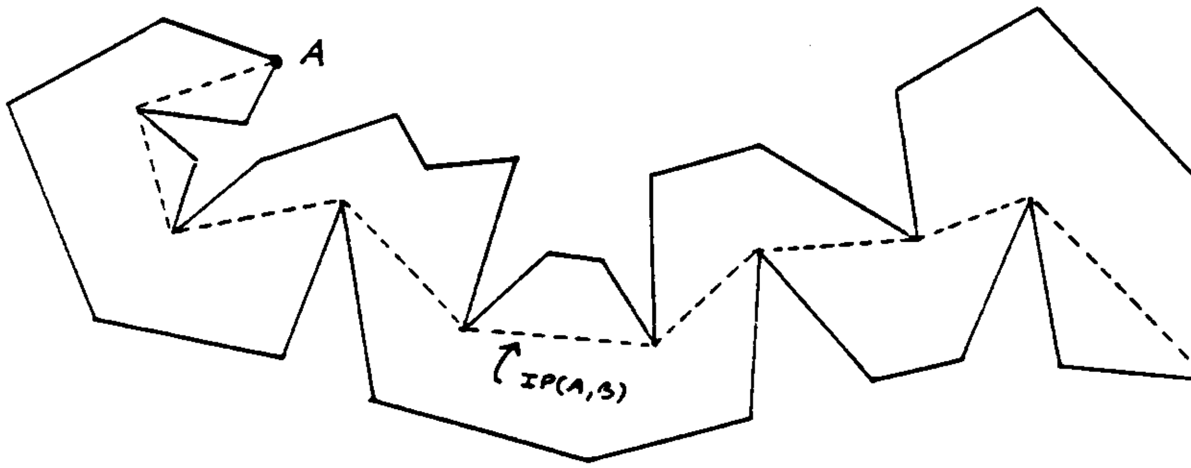


Figure 6

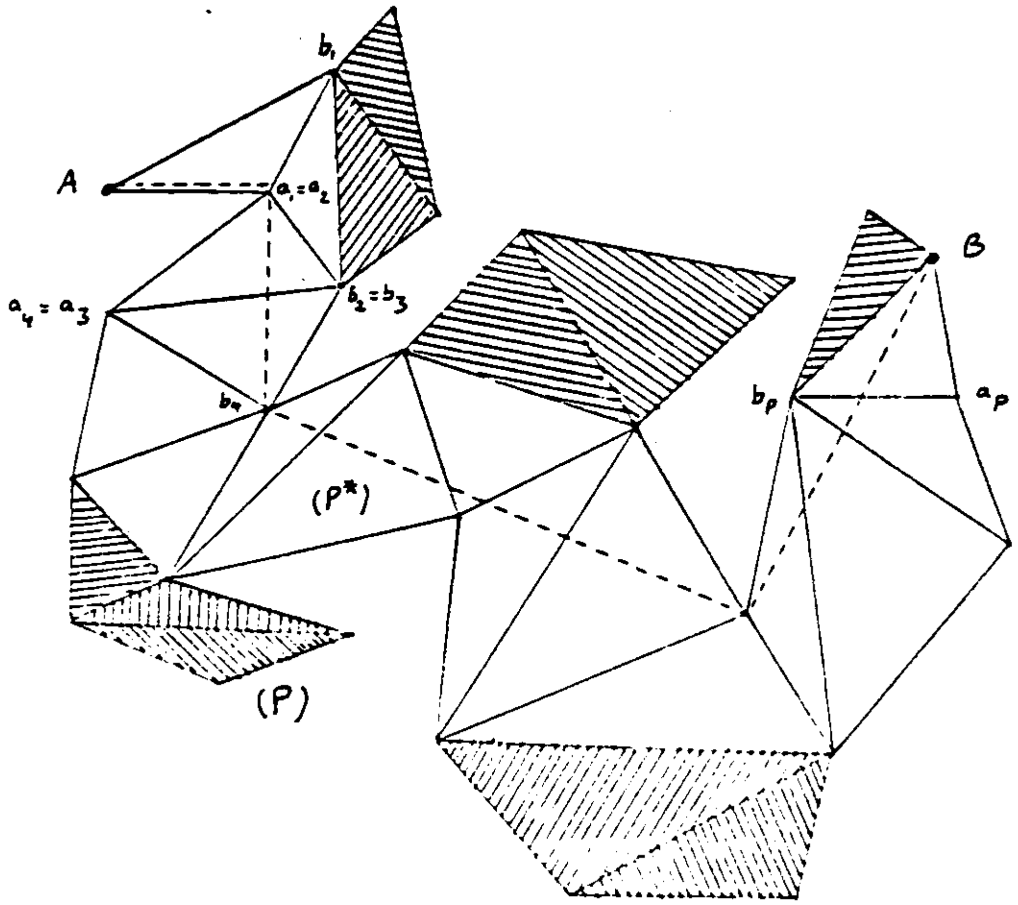


Figure 7

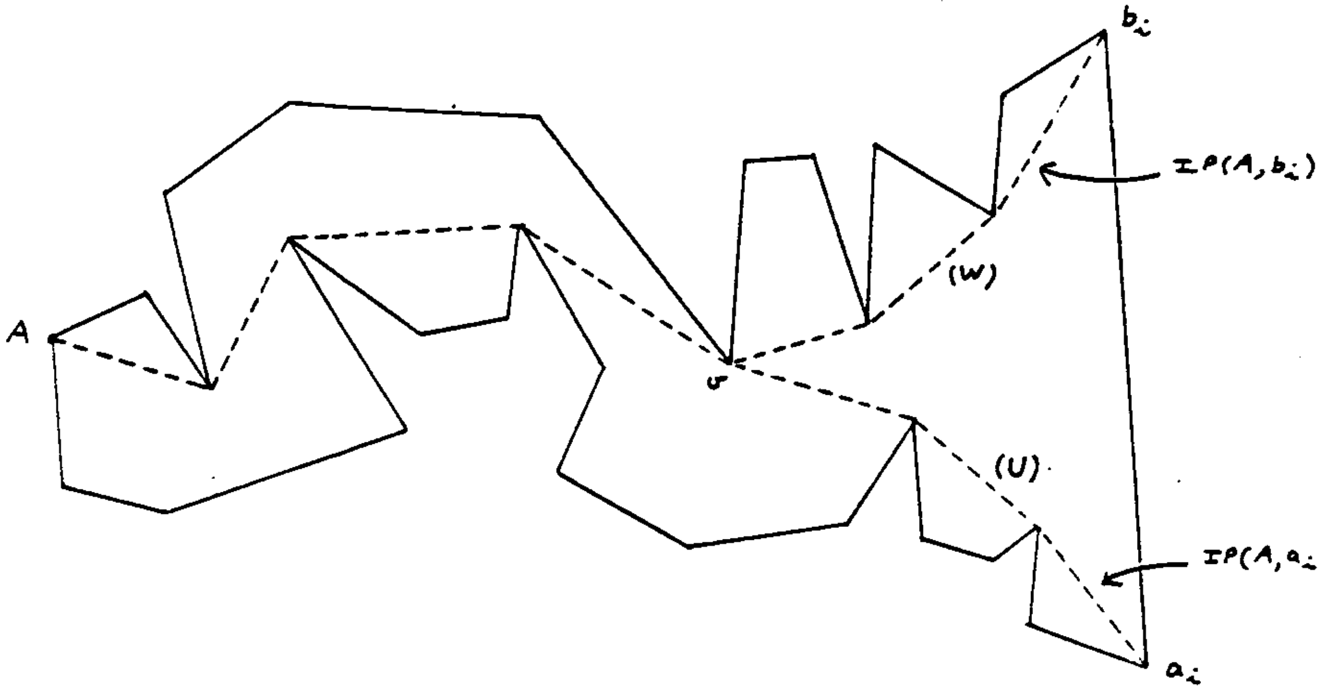


Figure 8

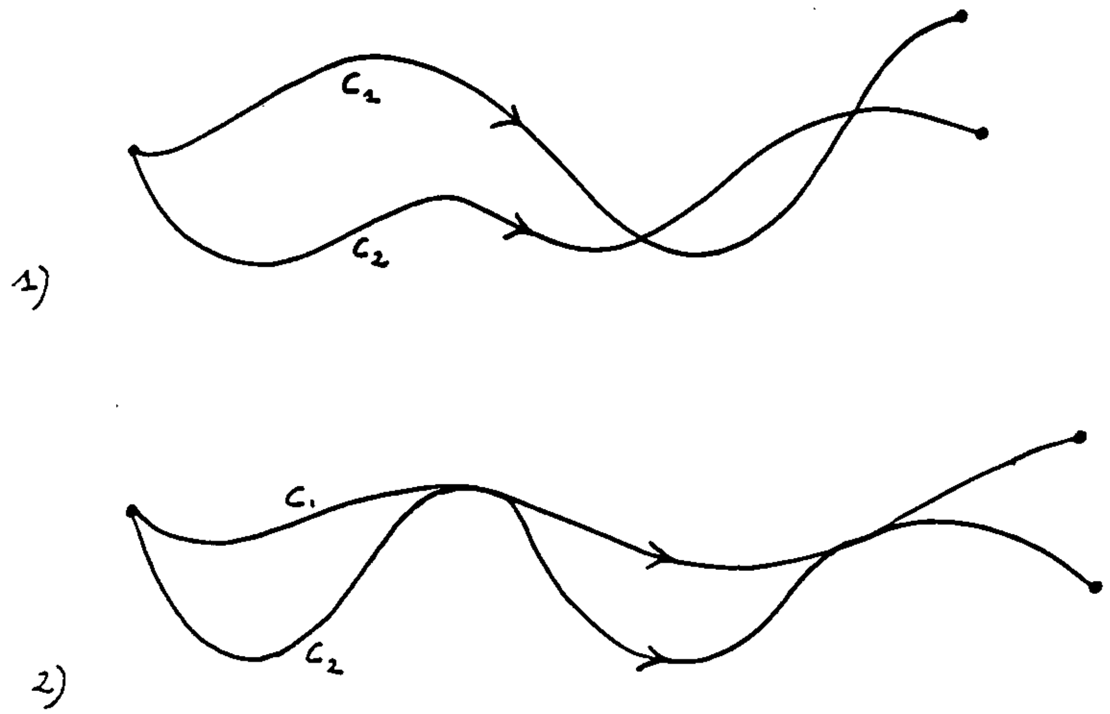


Figure 9

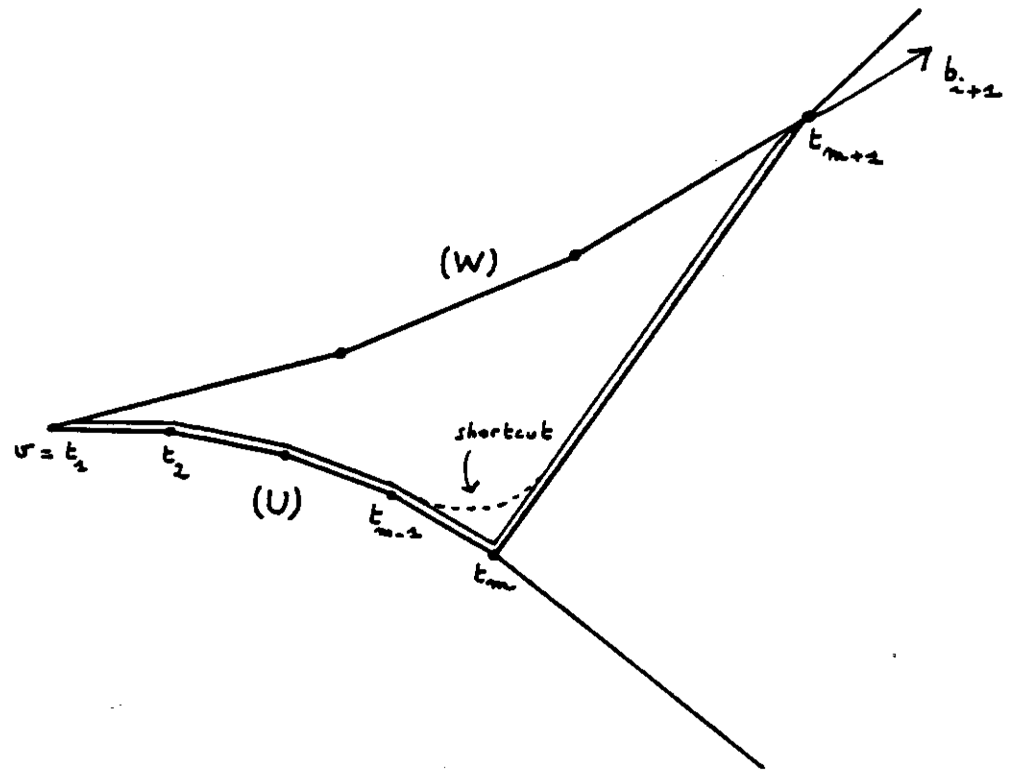
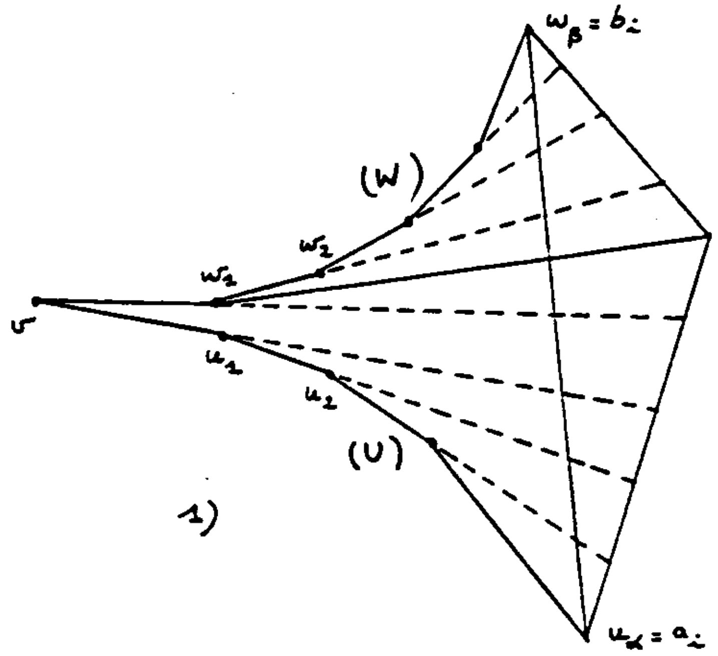


Figure 10


$w_\beta = b_i$
(W)
b_{i+1}
w_2
w_1
v
u_1
u_2
(U)
1)
$u_\alpha = a_i$

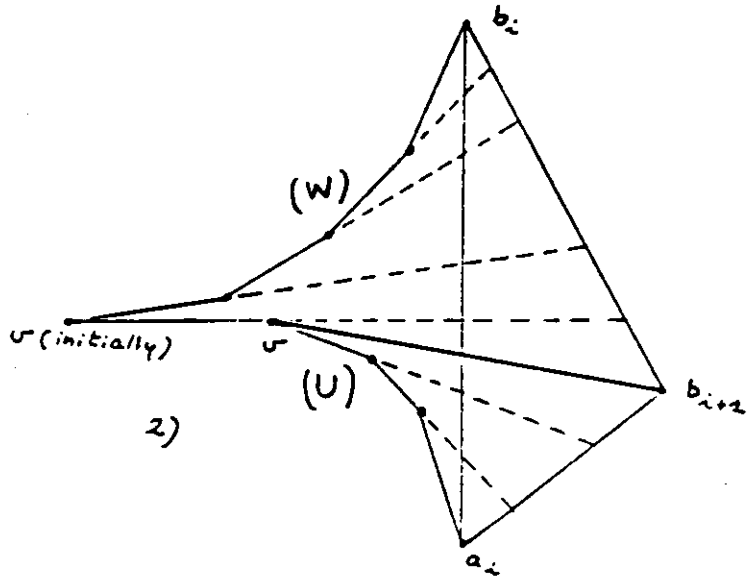

b_i
(W)
v (initially)
v
(U)
b_{i+1}
2)
a_i

Figure 11

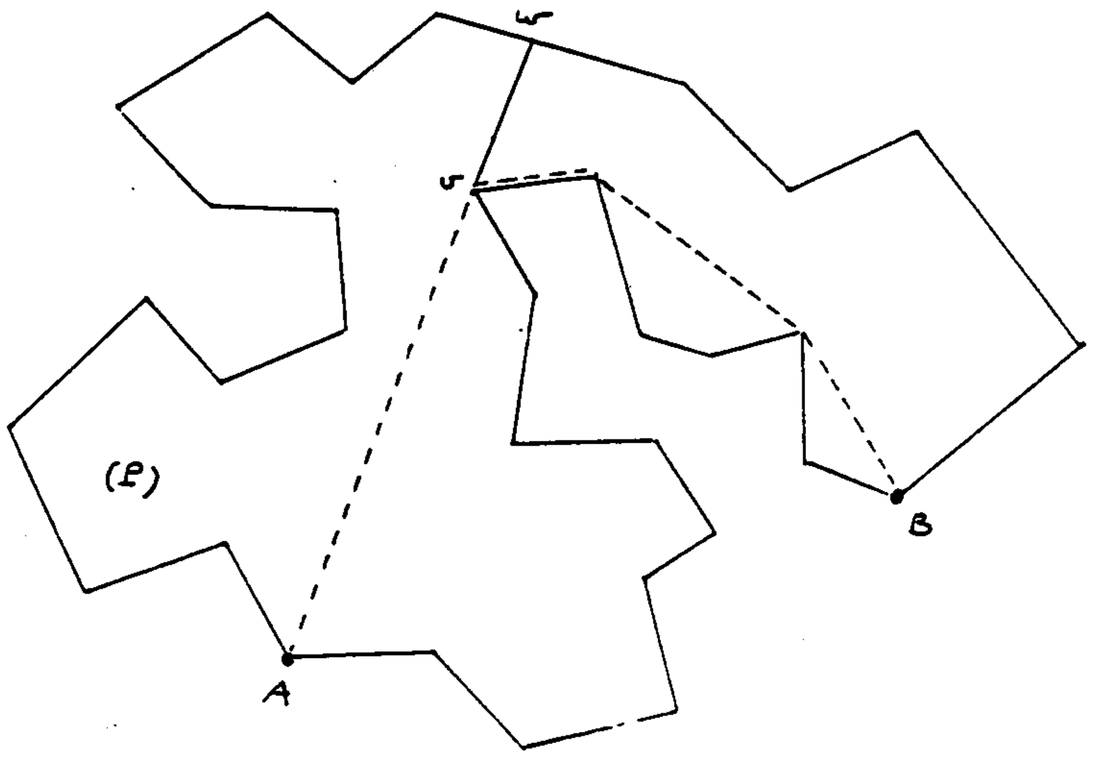


Figure 12