TR-88-029

A GENERALIZATION OF THE LOCUS APPROACH IN COMPUTATIONAL GEOMETRY

A Generalization of the Locus Approach in Computational Geometry

H. Y. Tu and R. C. Chang

TECHNICAL REPORT

Institute of Information Science

Academia Sinica

Taipei, Taiwan, Republic of China

July 1989

Contents

1. Introduction

In this paper we discuss techniques for the design of geometric algorithms. Of course, computational geometry makes wide use of standard algorithm design methods, such as divide–and–conquer, dynamic programming, and so forth [10]. We have tried to concentrate here on *a generalization of the locus approach.*

The locus approach is widely used in computational geometry algorithms. Some classic examples can be found in the problems related to construct *Voronoi diagram* [3,4]. Recent interest in the fields of robotics and industrial automation has prompted the study of *motion planning.* One of the basic problems in this field is to determine a continuous path for motion of a given body (the robot) in an environment that imposes geometric constraints on the body's motion. Sharir and Schorr [26,29] used the locus approach successfully to find the shortest paths in polyhedral spaces. Following their concepts, Schevon and O'Rourke [22,27,28] found an $O(n^{14}\log n)$ algorithm to compute the geodesic diameter of a 3–polytope, and also gave an $O(Kn^3\log n)$ algorithm to compute edge sequences on a convex polyhedron, where a loose bound of K is $O(n^6)$. A subsequence result of Hwang, Chang and Tu has further reduced the complexity to $O(n^6\log n)$ [11]. Instead of using the traditional locus approach: "subdividing space, which is the surface of a polyhedron, into regions," Hwang, Chang and Tu partitioned the Cartesian product of two edges into equivalent classes in [11]. This new technique is a generalization of the original locus approach.

Another field of directly using the locus approach is *dynamic computational geometry* [2,12]. In [2], Atallah proposed some problems in computational geometry when every one of the input points is moving in a prescribed manner. He also presented and analyze efficient algorithms for a number of problems and prove lower bounds for some of them. Using the generalized locus approach once again, Hwang, Chang and Tu [12]

reduced the time complexity of *separability problems* in dynamic computational geometry to almost $O(n^2 \log n)$ in 2–D but almost $O(n \log n)$ in 1–D. Recently Tu and Kao [31] are trying to apply this technique to the *cruising guard problem.* The results show that by the generalized locus approach the 1– and 2–cruising guard problems can be solved respectively in $O(n)$ and $O(kn)$ where k is the number of pockets in the simple polygon.

In this paper we will discuss and illustrate the generalized locus approach by examples. The main part of this paper is divided into three sections. After a general introduction, in Section 2 we deal mostly with the design method of the locus approach and give an example: finding all the shortest path edge sequences on the surface of a convex polyhedron. In Section 3 we apply the generalized locus approach to separability problem in dynamic computational geometry. Since the *Davenport–Schinzel sequences* plays an important role in domain partition, we also discuss them in the end of this section independently. Finally in Section 4 we end this paper with conclusions and remarks.

2. The Locus Approach

Locus approach is a common technique in solving geometric problems. A classic example is the *post office problem*: given n sites in the plane and a query point x, report the site closest to x. If the sites remain fixed over several queries, then it pays to subdivide the plane into regions, each of which consists of all points closest to a particular site. This partitioning of the plane is the well–known *Voronoi diagram* [3,4]. Once we have the Voronoi diagram, a nearest neighbor query can be answered simply by doing a point location in the diagram. This is the essence of the locus approach: subdivide space into regions such that all points in the same region yield the same answer to the type of query we are interested in. Point–location in this subdivision can then be used to answer any specific query.

In this section we introduce the generalization of the locus approach. Instead of subdividing the space on which the input points are located, we partition any domain which we concern. Furthermore, the partition functions can be various type. For instance, when finding all the shortest paths on the surface of a convex polyhedron, we use hyperbolic curves to partition the domain generated by Cartesian product of two edges. These two conceptual changes are different from the original locus approach.

To illustrate how we generalize the locus approach from the original one, we give a worked out example: finding all the shortest path edge sequences on the surface of a convex polyhedron. The following three subsections are all close related to this example. In the first subsection we present the general information of finding all shortest paths on the surface of a convex polyhedron. In the second subsection, applying the locus approach to the continuous Dijkstra technique, we develop an $O(n^6 \log n)$ algorithm to solve this problem. Finally in the third parts, by the same concepts and the same data structures, we solve 1– and 2– cruising guard problems respectively in $O(n)$ and $O(kn)$

time where k is the number of pockets in a simple polygon.

2.1 Shortest Path Edge Sequences on the Surface of a Convex Polyhedron

2.1.1 Introduction

Recent interest in the fields of robotics and industrial automation has prompted the study of *Motion Planning*. One of the basic problems is to determine a continuous path for the motion of a given body in an environment that imposes geometric constraints on the body's motion. In this section and the next one we consider the problem of computing the Euclidean shortest path between two points on the surface of a convex polyhedron P [26]. This problem is also of considerable interest in terrain navigation, where a moving vehicle is bound to move along a surface what could be modeled by a polyhedron (here we treat the vehicle as a single moving point) [17]. The shortest path problem on a convex polyhedron can be formally defined as follows [17]:

Let S be the surface of a given convex polyhedron P, defined by a set of faces, edges, and vertices, with each edge occurring in two faces and two faces intersecting either at a common edge, a vertex, or not at all. A *shortest path* between two points A and B on S is the Euclidean shortest path between points A, B along the surface of P. A *shortest path edge sequence* can be defined as an ordered list of edges of P such that any two adjacent edges share a common face, and such that there exists a shortest path traversing the edges in the list. A shortest path edge sequence is said to be *maximal* iff it is not the subset of any other shortest path edge sequence [27]. If the question is to find the shortest path between two fixed points on S, we call it *Discrete Geodesic Problem*. If only one source point (say A) is fixed and we are asked to build a structure which allows one to find out a shortest path from A to any other query point (say B), it is called *Single–Source Discrete Geodesic Problem*. For the general case, if two query points are

allowed to be chosen arbitrarily (both are not fixed) on S, we name it *General Geodesic Problem*. We can also make a restriction on the query domain such that the query points can only be chosen on edges. In this way, it is called *Edge–Point' General Geodesic Problem*. The enumeration of all shortest path edge sequences on a convex polyhedron is named *All Shortest Path Edge Sequence Problem*.

The *Discrete Geodesic Problem* and *Single–Source Discrete Geodesic Problem* were first posed in [29], where an $O(n^3 \log n)$ algorithm was given for the case of a convex polyhedron. A subsequent result of Mount [19] has reduced the running time to $O(n^2 \log n)$. Both methods are to find the subdivision on the surface of a given convex polyhedron according to one fixed source point, such that any point in the same region has the same shortest path edge sequence to this source point. After building the subdivision, the shortest path problem can be transformed into a standard point location problem and the shortest path from the fixed source point to a given query point can be computed in time $O(k+\log n)$ where k is the number of edges in the corresponding shortest path edge sequence. For the nonconvex case, O'Rourke, Suri, and Booth gave an $O(n^5)$ algorithm [21]. Subsequently, Mitchell [17] improved this result to $O(n^2 \log n)$ by using the "Continuous Dijkstra" technique. He combined the concepts of the original Dijkstra algorithm for finding shortest paths in a graph [6], and the subdivision method in [29]. In [17], edges of the given polyhedron behave like nodes of a graph, but here the distance from the source to an edge is not the unique value. Instead, Continuous Dijkstra Algorithm uses a function that serves as a label for an interval of the edge. Keeping track of the discrete description of these functions, one can subdivide the edge into regions for which the shortest path to points in the region have the same shortest path edge sequence. This method is a generalization of the algorithm proposed in [29].

Since all of the previous algorithms are inefficient to solve *General Geodesic Problem* or even *Edge–Point General Geodesic Problem*, few papers discuss them [17],

[29]. The problem of finding all shortest path edge sequences on a convex polyhedron originated from Sharir [26]. He proposed a method to compute shortest paths in 3–D amidst convex obstacles, whose solutions depend on all shortest path edge sequences of these convex obstacles. Sharir [26] gave an $O(n^8 \log n)$ algorithm to compute these edge sequences for each obstacle. He also provided a bound of $O(n^7)$ on the number of edge sequences. Subsequently, Mount [20] had further reduced this bound to $O(n^4)$ and gave an example to show that it is tight. Recently, Schevon and O'Rourke [27] used a graph–theoretic argument to show that the number of maximal sequences of edges traversed by shortest paths is $\theta(n^3)$. This result also provided an alternate proof that the total number of shortest path edge sequences is $O(n^4)$. In the same paper he also proposed an $O(n^7 \log n \cdot 2^{\alpha(n^2)})$ algorithm to compute all shortest path edge sequences of a convex polyhedron, which improved slightly on Sharir's algorithm.

In this section and the next one we shall propose an $O(n^6 \log n)$ algorithm to compute all shortest path edge sequences of a convex polyhedron, by using a data structure with a size of $O(n^4)$. According to this data structure, not only can we enumerate all shortest path edge sequences and draw out all maximal ones, but we can also find the shortest path between any two points lying on edges in $O(k+\log n)$ time where k is the number of edges crossed by the shortest path. Our approach consists of two major parts. In this section we shall first consider all $O(n^4)$ shortest path edge sequences as n edge sequence trees, and use the property of visibility between points on edges to construct these trees. In the next section, instead of creating the subdivision on the surface of a convex polyhedron [26], [17], for each edge pair (e_s, e_e) we construct the subdivision on domain $Z = e_s \times e_e$ so that any point (A,B) in the same region has the same shortest path edge sequence from point A to point B on S. This approach is the generalization of Continuous Dijkstra Algorithm in [17] and Slice Algorithm in [26].

2.1.2 Tree Representation for All Shortest Path Edge Sequences

Let P be a 3–D convex polyhedron with n edges. For each pair of points (A,B) on the surface of P, we denote the shortest path from A to B as $\pi(A,B)$, and the sequence of edges of P crossed by $\pi(A,B)$ as $\xi(\pi(A,B))$. To solve the *Edge–Point General Geodesic Problem* and generate all shortest path edge sequences, we shall first consider the restricted case in which the starting point A lies on an edge e_s, and the ending point B lies on another edge e_e. These two edges, e_s and e_e are called the starting edge and the ending edge respectively. Since the shortest paths on a convex polyhedron cannot cross any edge more than once [29], we can use the brute force approach to form all of the edge sequences as a permutation tree, and then determine which of these sequences are shortest path edge sequences.

For a convex polyhedron P, an *edge sequence tree* T with starting edge e_s is a tree specifying e_s as the root. Each node N_i in T is an edge of P, denoted as $E(N_i)$. Node N_j is a son of node N_i, if $E(N_j)$ shares a common face with $E(N_i)$ on P and $E(N_j)$ is not an ancestor of $E(N_i)$ in T. The path from root e_s to node N_i, denoted as $ES(N_i)$, is an edge sequence of P. If T' is the tree obtained from deleting some nodes of T, such that for every node N_i of T', $ES(N_i)$ is a shortest path edge sequence, we call T' a *shortest path edge sequence tree.* T' is considered *maximal* iff it can not be extended to form another shortest path edge sequence tree.

For example, the edge sequence tree of a tetrahedorn (see Fig. 2.1a) with starting edge e_1 is shown in Fig. 2.1b. Hence, the problem to find all shortest path edge sequences with a fixed starting edge is now reduced to the problem to build a maximal shortest path edge sequence tree with this edge.

Lemma 2.1 If edge sequence $\xi=(e_1e_2...e_{n-1}e_n)$ is a shortest path edge sequence, then its

subsequence $\xi_1=(e_1e_2...e_{n-1})$ is also a shortest path edge sequence.

Proof: Since ξ is a shortest path edge sequence, there exist two points, say X and Y, on e_1 and e_n respectively, such that the shortest path $\pi(X,Y)$ crosses ξ. Let Z be the intersection of $\pi(X,Y)$ and e_{n-1}. The subpath of $\pi(X,Y)$ from X to Z then, must be the shortest path between X and Z. Otherwise, the concatenation of $\pi(X,Z)$ and the subpath of $\pi(X,Y)$ between Z and Y would be shorter than $\pi(X,Y)$. Therefore, ξ_1 must be the edge sequence crossed by the shortest path from X to Z. ■

Lemma 2.1 implies that once we have found a shortest path edge sequence ξ_1, it is very likely that ξ would be another shortest path edge sequence. Thus, the process to find new shortest path edge sequences can be considered as the "expansion" on edge sequence trees. First, we specify the starting edge e_s as the root of T, and add the edges which share a common face with e_s as the children of the root. Then iteratively select a leaf F_i, whose $ES(F_i)$ is a shortest path edge sequence on P, and add edges sharing a common face with $E(F_i)$ as the children of F_i, until all the shortest path edge sequences are found.

In the process of expansion, we are immediately confronted with two problems: to determine which leaf F_i will lead $ES(F_i)$ to be the shortest path edge sequence; and to decide when to stop expanding the edge sequence tree. To decide whether $ES(F_i)$ is a shortest path edge sequence or not, we use the concept of visibility between points on edges [26]. Some definitions are specified as follows. Let $f_1f_2...f_n$ be a sequence of faces on a convex polyhedron P such that edge e_s (resp. e_e) is on the boundary of f_1 (resp. f_n) and f_i, f_{i+1} be adjacent on edge e_i for i=1,2,..,n–1. The planar unfolding of P relative to edge sequence $\xi=(e_se_1e_2...e_{n-1}e_e)$ is obtained by unfolding these faces, one at a time, about the edges that separate them, until they all lie in the plane containing f_1 (with no two adjacent faces overlapping one another, see Fig. 2.2). Two points A and B, on starting

edge e_s and ending edge e_e respectively, are *visible* to each other in edge sequence ξ if, after the planar unfolding relative to ξ, the straight line from A to B crosses ξ (if ξ is a set of edge sequences, it means that A and B are visible to each other in at least one of these edge sequences). Let $\pi_\xi(A,B)$ be the straight line segment connecting points A and B in this unfolding. $|\pi_\xi(A,B)|$ denotes its length. During expanding edge sequence tree T, the weight of leaf F_i is defined as follows:

$$W(F_i) = \min(\{\ |\pi_\xi(A,B)| :\ (A,B)\in e_s\times e_e,\ ES(F_i)=\xi;$$
$$\text{and}\ \forall\, N_j\in T\setminus\{F_i\},\ E(N_j)=E(F_i),$$
$$\text{such that}\ |\pi_\xi(A,B)|\le|\pi_{\xi'}(A,B)|\ \text{ where } ES(N_j)=\xi'\ \});$$

If the set in function min is empty, $W(F_i)$ is set to be infinite.

A leaf F_i is called *with minimal weight* if no other weights of leaves in T are smaller than $W(F_i)$. Roughly speaking, the weight of leaf F_i can reflect the existence of shortest paths between the points on $E(F_i)$ and the points on e_s in the planar unfolding relative to $ES(F_i)$. When $W(F_i)$ goes infinite, it implies that, for every $(A,B)\in e_s\times E(F_i)$ either the points A and B are invisible to each other, or we have already had a node N_j in the expanding edge sequence tree T such that the shortest path from A to B crossing edge sequence $ES(N_j)$ is shorter than the one crossing $ES(F_i)$. In other words, there are no shortest paths crossing edge sequence $ES(F_i)$.

Lemma 2.2 [29] If points A and B, on edges e_s and e_e respectively, are not visible in edge sequence ξ, then the shortest path edge sequence between A and B can not be ξ.

Lemma 2.3 In building edge sequence tree T, if F_i is the leaf with minimal weight, then

edge sequence $ES(F_i)$ is a shortest path edge sequence.

Proof: Assume that F_i is the leaf with minimal weight and $ES(F_i)=\xi_i$. There must be a pair of points (A,B) such that $W(F_i)=|\pi_{\xi_i}(A,B)|$. By the definition of weight, $|\pi_{\xi_i}(A,B)|$ is the smallest one for all possible (A,B) in the planar unfolding relative to ξ_i. If $\pi_{\xi_i}(A,B)$ is not the shortest path between A and B along the surface of P, then there exists another leaf F_j (let $ES(F_j)=\xi_j$) such that either ξ_j is the shortest path edge sequence between A and B, or ξ_j is just the subsequence of this shortest path edge sequence. In the latter case, we have $W(F_j)<W(F_i)$. This implies that $W(F_i)$ is not the minimal one. In the former case, we have $|\pi_{\xi_j}(A,B)|<|\pi_{\xi_i}(A,B)|$. $W(F_i)$ should not be $|\pi_{\xi_i}(A,B)|$ (by the definition of weight). Thus, $\pi_{\xi_i}(A,B)$ is the shortest path from A to B and ξ_i is its shortest path edge sequence. ■

From **Lemma 2.2** and **Lemma 2.3** we know when to stop expanding edge sequence tree T. If all the leaves in T are with infinite weight, there are no leaves to be expanded. **Lemma 2.3** also tells us which leaf should be included into the shortest path edge sequence tree, and this chosen leaf is also the next one to be expanded. In order to compute the weight for each leaf and find the minimal one quickly, a data structure called *visibility relation diagram* is used to maintain the visible relations between the points on edges.

2.2 A Generalized Locus Approach with Continuous Dijkstra Algorithm

In this section we shall first describe the structure of *visibility relation diagrams* in details. Then formally state the algorithm of finding all shortest path edge sequences on a convex polyhedron, and analyze the time complexity of the algorithm. Finally, conclusions with remarks are in the end.

2.2.1 Visibility Relation Diagrams

Let T be the currently expanding edge sequence tree with root e_s. Assume that S is a set of edge sequences in which all edge sequences have the same ending edge e_e. In order to determine whether there are shortest paths crossing the edge sequences in S, by Lemma 2.2, we must show the visible relationships between the points on e_s and the points on e_e in the planar unfoldings relative to the edge sequences of S. Our approach is to consider the 2–D space $Z=e_s \times e_e$ of all possible pairs of starting and ending points, and partition it into regions, such that for each such region R_ξ there exists an edge sequence ξ of S such that, for all $(A,B)\in R_\xi$, $\pi_s(A,B)=\pi_\xi(A,B)$. In other words, not only are the points A, B visible to each other in the planar unfolding relative to ξ, but the straight line segment connecting A and B in ξ is also smaller than the others in edge sequences of $S\backslash\{\xi\}$.

Definition. Assume that $S=\{\xi_1,\xi_2,\cdots,\xi_{n-1},\xi_n\}$ is a set of edge sequences with starting edge e_s and ending edge e_e. Let the function $f: e_s \times e_e \longrightarrow S\cup\{\phi\}$ defined by

$f(A,B)=\xi_i$ iff A and B are visible to each other in ξ_i ,

and $\pi_{\xi_i}(A,B) \le \pi_{\xi_j}(A,B)$ for all $\xi_j \in S\backslash\{\xi_i\}$;

$f(A,B)=\phi$ iff A and B can not be seen from each other in S.

For a pair of edges, e_s and e_e, a *visibility relation diagram* (short for VRD) restricted to S is a partition on domain $Z=e_s \times e_e$ defined by f. We denote the equivalent class corresponding to ξ_i as R_{ξ_i}.

In the following, we first consider the special case in which S contains only one edge sequence, and then we show how to modify the VRD restricted to S to a new VRD when adding an edge sequence into S. In the remained paragraphs, the method to compute weights of leaves from VRD will be proposed.

Initially, let S contain only one edge sequence $\xi=(e_1e_2..e_{n-1}e_n)$. After performing the planar unfolding relative to ξ, we have a polygon, denoted by $G_1(\xi)$, whose boundary is composed of e_1, e_n, and the edges connecting the end points of e_i, e_{i+1} for i=1,2,..,n−1 (see Fig. 2.3a). By using the algorithm in [9], it is easy to find two shortest paths connecting the end points of e_1 and e_n in $G_1(\xi)$ such that these two paths are not crossed with each other (see Fig. 2.3b). Hence, these two paths together with e_1 and e_n define another simple polygon $G_2(\xi)$. For the points $A \epsilon e_1$ and $B \epsilon e_n$, if they are visible to each other in ξ, their connecting straight line segment should be contained in $G_2(\xi)$. According to visibility between points on e_1 and on e_n, domain $Z=e_1 \times e_n$ can be partitioned into two equivalent classes R_ξ and R_ϕ : for the point (A,B) in R_ξ, A and B are visible to each other in ξ; if (A,B) is in R_ϕ, they can not be seen from each other in ξ. In order to find the boundary between R_ξ and R_ϕ on domain Z, we should formularize the boundary between these two equivalent classes.

Definition. The *boundary–points* of R_ξ are the points $(A,B) \epsilon Z$ where the straight line segment $\overline{AB}$ in $G_2(\xi)$ contains a vertex of $G_2(\xi)$.

Lemma 2.4 For a fixed vertex of $G_2(\xi)$ the locus of boundary points is a hyperbolic curve on domain $Z=e_1\times e_n$.

Proof: Let A and B be points respectively on e_1 and e_n, and c be the fixed vertex. Parameterize A and B as a_1+b_1u and a_2+b_2v respectively, for appropriate vectors a_1, a_2, b_1, b_2, and real parameters u, v. Then the condition that A, B, c are collinear can be written as

$$(A-c)\times(B-c)=0,$$

i.e.,
$$\begin{aligned} 0 &= (a_1+b_1u-c)\times(a_2+b_2v-c) \\ &= (a_1-c)\times(a_2-c) + [b_1\times(a_2-c)]u + [(a_1-c)\times b_2]v + (b_1\times b_2)uv, \end{aligned}$$

which is an equation of a hyperbola in u–v space [29]. ■

By Lemma 2.4, each vertex of $G_2(\xi)$ has a corresponding hyperbolic curve. The boundary of R_ξ is composed of these cure segments. For example the boundary defined by the polygon in Fig. 2.4b is shown in Fig. 2.4c.

Since we have found the equivalent class corresponding to just one edge sequence, our next goal is to show how to modify an existent VRD to a new VRD when adding a new edge sequence to edge sequence trees. Let $S=\{\xi_1,\xi_2,..,\xi_n\}$ be the set of all edge sequences with the same ending edge e_e in currently expanding edge sequence tree T. Assume that we have had a visibility relation diagram VRD_s restricted to S. Whenever a new node N is generated on T, if E(N) is e_e, we should modify VRD_s to show the existence of ES(N), since it is possible that some paths crossing ES(N) are shorter than the ones crossing the other edge sequences already existing in S. To simplify the notation, let ES(N) be ξ. The modification consists of two steps :

Step (1) partition domain Z into R_ξ and R_ϕ;

Step (2) for all points $(A,B)\in R_\xi\cap R_{\xi_i}$, decide whether the shortest path in the planar unfolding relative to ξ is shorter than the one in the planar

unfolding relative to ξ_i (determine whether (A,B) should be classified into R_ξ or R_{ξ_i}).

Step (1) can be accomplished by the previous method. For step (2), we perform two planar unfoldings relative to ξ and ξ_i on a common plane such that they share the common e_s. However, point B on e_e will be duplicated to two points in these two unfoldings, say B_ξ and B_{ξ_i} (see Fig. 2.5). Let the perpendicular bisector of $\overline{B_\xi B_{\xi_i}}$ intersect e_s at point C. This bisector partitions the plane into two halfplanes. One contains B_ξ while the other contains B_{ξ_i}. If A is in the same halfplane with B_ξ, then $|\overline{AB_\xi}|<|\overline{AB_{\xi_i}}|$. In other words, the path from A to B crossing ξ is shorter than the one crossing ξ_i. Hence (A,B) should be classified to R_ξ. On the contrary, if A and B_{ξ_i} are in the same halfplane, (A,B) belongs to R_{ξ_i}. When A is just located on C, we have $|\overline{AB_\xi}|=|\overline{AB_{\xi_i}}|$. It means that if we move point B on the edge e_e (the position of point C is well defined) the locus of (C,B) can partition $R_\xi \cap R_{\xi_i}$ into two regions, where one should be combined into equivalent class R_ξ, while the other should be included into R_{ξ_i}. We name these points (C,B) the *partition–points*. Hence, this new partition on domain Z, obtained by modifying the original VRD_s, is the visibility relation diagram restricted to $SU\{\xi\}$. In the same way mentioned in **Lemma 2.4**, it is easy to show that the locus of these partition–points is also a hyperbolic curve on domain Z.

Lemma 2.5 The locus of the partition–points of $R_\xi \cap R_{\xi_i}$ is a hyperbolic curve on domain Z.

Proof: To make the proof simple. We follow the previous notations. Let $B_\xi = a+bu$, $B_{\xi_i}=a_i+b_iu$, and $C=c+dv$, for appropriate parameters. Since $\angle B_\xi B_{\xi_i} C = \angle B_{\xi_i} B_\xi C$, we

have the following equation,

$$(B_{\xi_i}-B_{\xi})\times(C-B_{\xi}) + (B_{\xi}-B_{\xi_i})\times(C-B_{\xi_i}) = 0,$$

which can be simplified as

$$c_0+c_1u+c_2v+c_3uv+c_4u^2=0,$$

where

$$c_0=[(a_i-a)\times(c-a)]+[(a-a_i)\times(c-a_i)]$$

$$c_1=[(a_i-a)\times d]+[(a-a_i)\times d]$$

$$c_2=[(a_i-a)\times(-b)+(b_i-b)\times(c-a)]+[(a-a_i)\times(-b_i)+(b-b_i)\times(c-a_i)]$$

$$c_3=[(b_i-b)\times d]+[(b-b_i)\times d]$$

$$c_4=[(b_i-b)\times(-b)]+[(b-b_i)\times(-b_i)]$$

This equation is also a hyperbola in u–v space. ■

As mentioned in Section 2.1.2, VRD is built to show the visible relation between points on edges, and to compute the *weights* of leaves in expanding T. In order to get the weights of leaves from VRD, we should point out which path makes the edge sequence to be the shortest path edge sequence. Let F be a leaf in T and ES(F)=ξ. A *weight–point* (A,B) of F is a point in R_{ξ} such that $W(F)=|\pi_{\xi}(A,B)|$. If R_{ξ} is empty, F has no weight–points. We define the *boundary* of R_{ξ} as the union of its boundary–points and partition–points.

Lemma 2.6 If F is a leaf with non–empty R_{ξ}, there exits a *weight–point* (A,B) of F, which is located on the *boundary* of R_{ξ}.

Proof: We prove this lemma by contradiction. Assume that all the weight–points of F are neither boundary–points nor partition points. Let (A,B) be one of these weight–points. By definition of weight–point we can find two points on the boundary of R_{ξ}, say (A_1,B_1) and (A_2,B_2), such that $\pi_{\xi}(A_1,B_1)$ and $\pi_{\xi}(A_2,B_2)$ are both longer then

π_ξ(A,B), run parallel with π_ξ(A,B), and are on the different sides of π_ξ(A,B). But this is contrary to the fact that both A_1, A, A_2 are collinear on starting edge, and B_1, B, B_2 are collinear on ending edge (see Fig. 2.6). Thus, there must be a weight–point on the boundary of R_ξ. ■

With the same geometric analyses used in **Lemma 2.4** and **Lemma 2.5**, the lengths of $\overline{AB}$ and $\overline{CB}$ can be formulized as hyperbolic functions of parameters u and v, too. Since there exists a weight–point on the boundary, we can compute the weight by differentiating these functions. Hence the visibility relation diagrams not only can show the visibility between edges but also can maintain the weights of nodes during expanding the edge sequence trees.

2.2.2 The Algorithm and its Time Complexity

In this section we first formally state the algorithm of finding all shortest path edge sequences on a convex polyhedron, and then analyze its time complexity.

We can describe our algorithm formally as follows:

Algorithm: Finding_All_Shortest-Path–Edge–Sequences (FAS)

Input: The data structure representing the convex polyhedron P

Output: Visibility Relation Diagrams and Edge Sequence Trees for All Shortest Path Edge Sequences of P

(1) FOR each edge e_i on P, use e_i as the starting edge DO :

(2) Let e_i be the root of edge sequence tree T_i;

(3) FOR each edge e_j sharing a common face with e_i DO :

(4) Construct the VRD on domain $Z=e_i \times e_j$;

(5) Let e_j be the son of e_i and compute the *weight* of e_j;

(6) END of FOR;

(7) WHILE there exists a leaf whose *weight* $\neq \infty$ DO :

(8) Find the leaf F with *minimal weight*;

(9) FOR each edge e_k sharing a common face with E(F) DO :

(10) Construct/Modify the VRD on domain $Z=e_i \times e_k$;

(11) FOR each leaf F' with $E(F')=e_k$ DO :

(12) Compute/Recompute the *weight* of F' END of FOR;

(13) Let e_k be the son of F;

(14) END of FOR;

(15) END of WHILE;

(16) END of FOR.

The correctness of Algorithm FAS can be shown in the following theorem.

Theorem 2.7 By Algorithm FAS, we can construct a one to one correspondence between the shortest path edge sequences on P and the paths from root to internal nodes in edge sequence trees.

Proof: We prove this theorem by induction. Let i be the length of the edge sequence. For i=1 or 2, the statements are obviously true. Assume that the statements are true for $i \leq n-1$. Let $\xi=(e_1e_2..e_n)$ be a shortest path edge sequence on P. By Lemma 2.1, the edge sequence $\xi_1=(e_1e_2..e_{n-1})$ is also a shortest path edge sequence. By inductive hypothesis, there must exist a node N in edge sequence trees such that $ES(N)=\xi_1$. Since ξ is a shortest path edge sequence, N has at least one son, say F, where $E(F)=e_n$, such that W(F) is NOT infinite (ref. Lemma 2.2, Lemma 2.3, and the definition of *weight* in Section 2.1.2). This implies that ξ is a path from root to node N in edge sequence trees.

The argument is clearly reversible; hence the theorem is proved. ∎

The running time of Algorithm FAS depends on

(a) the number of nodes in edge sequence trees,

(b) the number of regions in visibility relation diagrams, and

(c) the time to modify visibility relation diagrams during expanding edge sequence trees.

To see this, we examine each separately.

For (a), Mount [20] and O'Rourke [27] have proved that there are $O(n^3)$ shortest path edge sequences from a fixed starting edge to the other edges on P. This implies that the number of internal nodes in each edge sequence tree can be bound to $O(n^3)$. To simplify the analysis, assume that P is triangulated. By the fact that the shortest path can cross a face only once, each internal node has no more than two children. Hence there are overall $O(n^3)$ nodes (including leaves) in an edge sequence tree. Since we construct n edge sequence trees in Algorithm FAS, there are totally $O(n^4)$ nodes in n edge sequence trees.

For (b), to count the number of regions in visibility relation diagrams, we first examine the correspondence between the regions and the shortest path edge sequences.

Lemma 2.8 There are $O(n^2)$ regions in each visibility relation diagram after performing Algorithm FAS.

Proof: To prove this lemma, we show that for any two points $A=(A_s,A_e)$, $B=(B_s,B_e)$ on domain $Z=e_s\times e_e$, if A and B are in the same equivalent class, say R_ξ, then there exists a path $P\subset R_\xi$ connecting A and B on domain Z. In other words, R_ξ is path connected. Without loss of generality, we discuss the following cases separately.

CASE 1: If $A_e=B_e$, $\overline{AB}$ is parallel to e_s on domain Z (see Fig. 2.7a). We shall claim

that $\overline{AB} \subset R_\xi$.

Assume that there exists some point $C=(C_s,A_e)$ on $\overline{AB}$ but belonging to $R_{\xi'}$, where $\xi' \neq \xi$. We first perform planar unfoldings relative to ξ and ξ' on a common plan such that they share the common edge e_s. However, point A_e will be duplicated to two points, say A_ξ and $A_{\xi'}$, (see.Fig. 7b). Let the perpendicular bisector to $\overline{A_\xi A_{\xi'}}$ intersect e_s at point E. This bisector partitions the plane into two halfplanes. Since $(C_s,A_e) \in R_{\xi'}$, and $(A_s,A_e) \in R_{\xi'}$, we have $|\pi_\xi(C_s,A_e)| > |\pi_{\xi'}(C_s,A_e)|$ and $|\pi_\xi(A_s,A_e)| < |\pi_{\xi'}(A_s,A_e)|$. This implies that respectively $|\overline{C_s A_\xi}| > |\overline{C_s A_{\xi'}}|$ and $|\overline{A_s A_\xi}| < |\overline{A_s A_{\xi'}}|$ in these planar unfoldings. Hence E must be on $\overline{A_s C_s}$,and B_s is in the same hyperplane with C_s, which means $|\pi_\xi(B_s,A_e)| > |\pi_{\xi'}(B_s,A_e)|$. The shortest path edge sequence from B_s and A_e should not be ξ ,but be ξ'. This contradicts to our assumption, $(B_s,A_e) \in R_\xi$. The case of $A_s=B_s$ can also be derived from, instead of create two e_e, duplicating e_s when performing planar unfoldings.

CASE 2: With same notations, if neither $A_e=B_e$ nor $A_s=B_s$, we have two kinds of planar unfoldings relative to ξ (see Fig. 2.8). In one case $\pi_\xi(A_s,A_e)$ crosses $\pi_\xi(B_s,B_e)$, while in the other case these two paths are not crossed by each other.

For the former case, if $\pi_\xi(A_s,A_e)$ crosses $\pi_\xi(B_s,B_e)$ at point D (see Fig. 2.8a), we shall claim that the following curve P is a path connecting point A and B in domain Z and $P \subset R_\xi$.

P is a hyperbolic curve in domain $[A_s,B_s] \times [A_e,B_e]$ such that

for every point $(P_s,P_e) \in P$, P_s, D, and P_e are collinear in the planar unfolding relative to ξ.

Since A and B belong to R_ξ, $\overline{A_s A_\xi}$ and $\overline{B_s B_\xi}$ should be included in polygon $G_2(\xi)$. Hence, for all points $(P_s,P_e) \in P$, P_s and P_e can be seen by each other in the planar unfolding relative to ξ. Assume that some point $C \in P$ belongs to $R_{\xi'}$, where $\xi' \neq \xi$ and $C=(C_s,C_e)$. This implies that $|\pi_\xi(C_s,C_e)| > |\pi_{\xi'}(C_s,C_e)|$. On the other hand, since A

and B belong to R_ξ, we have $|\pi_\xi(A_s,A_e)|<|\pi_{\xi'}(A_s,A_e)|$ and $|\pi_\xi(B_s,B_e)|<|\pi_{\xi'}(B_s,B_e)|$ respectively. A simple geometric analysis can derive that $|\pi_\xi(C_s,C_e)|<|\pi_{\xi'}(C_s,C_e)|$, which contradicts to our assumption that $C\in R_{\xi'}$. Thus every point $C\notin P$ must belong to R_ξ.

For letter one, since both A and B belong to R_ξ, $\overline{A_sA_\xi}$ and $\overline{B_sB_\xi}$ should be included in polygon $G_2(\xi)$. Hence, all points on $\overline{A_sA_\xi}$ and $\overline{B_sB_\xi}$ can be seen by each other in the planar unfolding relative to ξ. Let $A'=(A_s,B_e)$ and $B'=(B_s,A_e)$. In the following, we shall claim that either $\overline{AA'}\cup\overline{A'B}$ or $\overline{AB'}\cup\overline{B'B}$ (but not both) belongs to R_ξ.

Assume that A' belongs to $R_{\xi'}$, but $\xi'\neq\xi$. This implies that the shortest path edge sequence from A_s to B_e is ξ'. Thus, we have $|\pi_\xi(A_s,B_e)|>|\pi_{\xi'}(A_s,B_e)|$. On the other hand, since both A and B belong to R_ξ, we have $|\pi_\xi(A_s,A_e)|<|\pi_{\xi'}(A_s,A_e)|$ and $|\pi_\xi(B_s,B_e)|<|\pi_{\xi'}(B_s,B_e)|$ respectively. A simple geometric analysis can derive that either $|\pi_\xi(A_s,B_e)|<|\pi_{\xi'}(A_s,B_e)|$ (see Fig. 2.8b) or $B'\in R_\xi$, but not both. Here, the former one contradicts to our assumption that $A'\in R_\xi$, while the latter one meets $\overline{AB'}\cup\overline{B'B}\subset R_\xi$ (by *CASE 1*). The relative statements are also true for assumption $B'\notin R_\xi$.

With the analytical results in *CASE 1* and *CASE 2*, it is not difficult to see R_ξ is path connected. Since the number of equivalent classes on domain Z is the same as the number of shortest path edge sequence, the number of regions on domain Z is bound to $O(n^2)$. Hence Lemma 2.8 is true. ■

Our next goal is to show that when processing Algorithm FAS the number of regions in each visibility relation diagram is also no more than $O(n^2)$. By Lemma 2.8 we have known that each internal node has only one corresponding region, but it is possible that, in building edge sequence trees, we have a leaf (or leaves) whose edge sequence has two (or more) corresponding regions in the visibility relation diagram. For this leaf, it

will eventually be either an internal node constituting an edge of some shortest path edge sequence, or a leaf with infinite weight. In the former case, the final internal node will contain only one corresponding region, while the other regions will be overlaid by the regions of other internal nodes. The latter one implies that the path from root to this node is not a shortest path edge sequence. Its equivalent class should be empty and all its corresponding regions will be covered by regions of other edge sequences. Thus, during the entire process of **Algorithm FAS**, the number of regions in each visibility relation diagram will be no more than the number of all shortest path edge sequences from a fixed starting edge to another fixed ending edges. The above discussion can be summarized as the following.

Corollary 2.9 The number of regions in each visibility relation diagram can be bound to $O(n^2)$ during the whole process of **Algorithm FAS**.

For (c), we shall claim that for each time we expand a node in edge sequence trees it takes $O(n^2\log n)$ time to modify its corresponding visibility relation diagram. Using the notations in Section 2.1.2, the planar unfolding relative to some edge sequence [29] and the construction of polygon $G_2(\xi)$ [9] can both be performed in $O(n\log n)$ time. The construction of all intersection regions $R_{\xi} \cap R_{\xi_i}$ can be accomplished by calculating the intersections between the hyperbolic curves, by sorting these points along each of these curves, and then by tracing the boundary of each intersection region. Since by Corollary 2.9 we know there are at most $O(n^2)$ regions in each visibility relation diagram, it needs overall time $O(n^2\log n)$ [26] to draw out all intersection regions. For each of the resulting $O(n^2)$ intersection regions (at most), we must draw a hyperbolic curve to partition it. Since the planar unfoldings relative to ξ and ξ_i have been put on a common plane, this step takes constant time. To compute the weight for a new node, we must differentiate

the boundaries of its corresponding regions. It takes $O(n)$ time. With this information, the next time we modify its weight, we need only constant time. The above discussions can be summarized as follows. To expand a new node in an edge sequence tree, we spend $O(n\log n)$ time to construct $G_2(\xi)$ and region R_ξ, $O(n^2\log n)$ time to find the intersection regions, and $O(n^2)$ time to modify the visibility relation diagram and compute the weights of leaves. Hence, it takes overall $O(n^2\log n)$ time to expand a new node in the edge sequence tree.

By the analytical results to (a), (b), and (c), we can conclude that Algorithm FAS totally takes $O(n^6\log n)$ time to construct n edge sequence trees and $n(n-1)/2$ visibility relation diagrams. Since the visibility relation diagrams show us the visibility between points on edges of P, the problem of finding shortest path edge sequences on P can be reduced to a location problem on VRD's. For a pair of given points (A,B) lying on edges e_s and e_e respectively, we need only $O(\log n)$ time to identify its located region in domain $Z=e_s\times e_e$. Thus, its corresponding shortest path edge sequence can be draw out from edge sequence trees immediately.

By concluding above discussions, we give the following theorem.

Theorem 2.10 Given a convex polyhedron P with n vertices, one can preprocess P by a procedure which runs in $O(n^6\log n)$ time. This procedure produces n edge sequence trees and $n(n-1)/2$ visibility relation diagrams, in each of which has $O(n^2)$ regions. With the aid of these trees and diagrams one can find the shortest path edge sequence between any two specified points lying on edges in $O(k+\log n)$ time where k is the number of edges in the shortest path edge sequence.

2.2.3 Conclusions and Remarks

As mentioned in Schevon and O'Rourke's paper [27], the gap between the number of shortest path edge sequences and the time to compute them can be narrowed. This research work has shown it. We transfer the visible relationship between edges into *Visibility Relation Diagrams*, and organize all shortest path edge sequences into *n Edge Sequence Trees* in overall $O(n^6 \log n)$ time. This is a new approach in finding all shortest path edge sequences. It is different from Sharir's [26] or Mount's [19] methods, in which they partitioned the surface of a polyhedron into slices. Hence the running time can be reduced. It seems quite likely that the algorithm developed in this paper is much closer to the optimal one, as there are $O(n^4)$ shortest path edge sequences on the polyhedron, and for given two points, without preprocessing, one needs $O(n^2 \log n)$ time to find their shortest path edge sequence (the best method up to now). We expect that the time complexity could be reduced to $O(n^6)$ by using some better data structures to maintain visibility relation diagrams. Keeping the ordering of the boundary of each region during constructing visibility relation diagrams could be another approach to reduce the time bound. The data structure of the visibility relation diagram may be of interest in its own right.

The method we used in this paper is a generalization of the continuous Dijkstra technique in [17]. In [17], the Continuous Dijkstra technique was limited to discuss the relationship between a fix point p and the points on each edge e or face f. In our term, these relationships can be characterized as the visibility relation diagrams on domain Z=p×e or Z=p×f respectively. For example, It is easy to understand that *Single–Source Discrete Geodesic Problem* can be looked at as a special case of *Edge–Point General Geodesic Problem*. In Algorithm FAS, we simply initialize original starting edge to be a single point, and then proceed exactly as before to construct the visibility relation diagram for each edge. Obviously, each of these visibility relation diagrams is a partition on the corresponding edge. This is actually what Mitchell has done in [17].

We believe that the generalized *Continuous Dijkstra algorithm* can also be applied to *General Geodesic Problem*, which is important in the study of robotics and terrain navigation. But in the generalization from $Z=e_i \times e_j$ to $Z=f_m \times f_n$, the process to partition Z into equivalent regions will be more complex. It obviously includes a subproblem which is the dynamic point location problem in 4–D. Hence, whether we can develop a good algorithm for this generalized case crucially depends on the method to solve the dynamic point location problem in 4–D.

Acknowledgements

We thank Joseph O'Rourke for his helpful papers, and Micha Sharir for his suggestions and comments.

2.3 Cruising Guard Problem

The *cruising guard problem* is another problem to which we successfully apply the locus approach. By the same concepts and the same data structures described in Section 2.2, in this subsection we develop algorithms to solve 1– and 2– cruising guard problems respectively in $O(n)$ and $O(kn)$ where k is the number of pockets in a simple polygon.

2.3.1 Definitions and Notations

Let P denote a simple polygon which is represented by a sequence of vertices v_0, v_1,.., v_{n-1}, and $e_0 = \overline{v_{n-1}v_1}$ with $e_i = \overline{v_{i-1}v_i}$ for i= 1,2,..,n–1 are the edges of the polygon connecting the corresponding vertices. We also assume that the boundary of P is directed clockwise, that is, the interior of P lies to the right of each edge.

P is said to be weakly visible from an edge e if for each p on P, there is a p_e on e such that the line segment $\overline{pp_e}$ lies inside P. Here we say that a line segment lies inside P if the interior of the line segment lies in the interior of P [33].

Definition. Let P be a simple polygon which is weakly visible from some specified edge e. The cruising guard problem is to

Minimize MAX({| s_i | i=1,2,..,k})

s.t. P is weakly visible from E

where $E = \cup s_j$ and $\forall s_j \subset e$ j=1,2,..,k

As motivation for the definition, consider the placement of k guards on edge e, whose job is to observe the entire polygon P. If P is weakly visible from e, it is necessary for the guards to cruise along some sections of e. Of course, to monitor the whole area in

this polygon efficiently, we would like to minimize the longest section.

To simplify the algorithm and yield an easier proof of correctness, we put the polygon in a "standard form." Consider the polygon in Fig. 2.1. Let v_{n-1}' and v_0' be the intersections, if any, of the extension of $\overline{v_{n-1}v_0}$. It is clear that the vertices in region A can only be seen by v_{n-1}. Hence, E must include v_{n-1}. The same applies to region B, with vertex v_0 replacing vertex v_{n-1}. To be more general, let P' be a new simple polygon obtained by replacing v_0' and v_{n-1}' with v_0 and v_{n-1} respectively, and deleting regions A and B. P' has the property that all of its vertices lie on the same side of the line $\overleftrightarrow{v_{n-1}v_0}$. A polygon with such a property is said to be in standard form. In the following, all algorithms will designed to work on such polygons.

We will assume for convenience that the origin of our coordinate system is at v_{n-1} and that the edge $\overline{v_{n-1}v_0}$ lies along the positive x–axis.

2.3.2 One Cruising Guard Problem

Let P be a simple polygon, which is in standard form and is weakly visible from edge $e_0 = \overline{v_{n-1}v_0}$. For the cruising guard problem, if k=1, we need to find a shortest segment on e_0, from which P is weakly visible.

Definition. For vertex v_i, define

r_i = MAX({ r | $r \in e_0$ and v_i is visible from r }) and

l_i = MIN({ l | $l \in e_0$ and v_i is visible from l }).

For edge e_i, if $r_{i-1} < l_i$, $\overline{r_{i-1}l_i}$ is said to be of *type 2 segment* and denoted as t_{2i}; if $r_{i-1} > l_i$, $\overline{l_i r_{i-1}}$ is said to be of *type 1 segment* and short for t_{1i} (see Fig. 2.10).

The geometric meanings of type 1 segment and type 2 segment are the following

facts;

Fact 1: (type 2 segment) If $\overline{r_{i-1}l_i}$ is a type 2 segment, t_{2i} is the shortest segment from which e_i is visible.

Fact 2: (type 1 segment) If $\overline{l_i r_{i-1}}$ is a type 1 segment, e_i is visible from any point of t_{1i}.

Thus, the necessary and sufficient conditions of weakly visibility from a segment are

Lemma 2.11 If E is a segment of e_0 such that

$$\forall\ t_{1i}\ ,\ t_{1i} \cap E \neq \phi \quad \text{and} \quad \forall\ t_{2i}\ ,\ t_{2i} \cap E = t_{2i}\ ,$$

then P is weakly visible from E.

Lemma 2.12 The shortest E, which has the property of Lemma 2.11, is the optimal solution for 1 cruising guard problem.

Hence we design an algorithm for 1–cruising guard problem directly from Lemma 2.11 and Lemma 2.12.

Algorithm 2.1 (1–cruising guard problem):

(1) For each v_i , compute r_i and l_i ;

(2) For each e_i , compute $\overline{r_{i-1}l_i}$ or $\overline{l_i r_{i-1}}$ and label them as t_{2i} or t_{1i} ;

(3) If there are no type 2 segments

then $R_2 = \phi$ and $L_2 = \phi$;

else $R_2 = MAX(\{\ l_i \mid \overline{r_{i-1}l_i}$ is a type 2 segment $\})$ and

$L_2 = \text{MIN}(\{\ r_{i-1} \mid \overline{r_{i-1}l_i}$ is a type 2 segment $\})$;

(4) If there are no type 1 segments

then $R_1 = \phi$ and $L_1 = \phi$;

else $R_1 = \text{MAX}(\{\ l_i \mid \overline{l_i r_{i-1}}$ is a type 1 segment $\})$ and

$L_1 = \text{MIN}(\{\ r_{i-1} \mid \overline{l_i r_{i-1}}$ is a type 1 segment $\})$;

(5) $R = \text{MAX}(\ R_1\ ,\ R_2\ )$ and $L = \text{MIN}(\ L_1\ ,\ L_2\ )$;

(6) If $L < R$

then segment $\overline{LR}$ is the answer ;

else any point on $\overline{RL}$ is the answer.

The running time of this algorithm is clearly polynomial in n. Step (1) and step (2) do the same work as described in [1] in which paper Avis and Toussaint had a linear algorithm for determining the visibility of a polygon from an edge. For steps from (3) to (6) all the work to do is to find MAX and MIN. It an be also accomplished in $O(n)$. Thus we have a linear time algorithm for solving 1–cruising guard problem.

2.3.3 Two Cruising Guard Problem

Assume that for convenience that the optimal solution of 2–cruising guard problem is a set of two segments, s_1 and s_2, of e_0. Let E=$\overline{LR}$ be the segment obtained by Algorithm 2.1. Point p_m is the midpoint of E. The following lemma for the upper bound of s_1 and s_2 is always true.

Lemma 2.13 $|E|/2$ is the upper bound of $\text{MAX}(\{|s_1|,|s_2|\})$. Hence, neither s_1 nor s_2 crosses p_m.

In the following, we assume that s_1 is contained in segment $\overline{Lp_m}$ while s_2 is contained in segment $\overline{p_mR}$. The main idea of our algorithm consists of two steps.

(1) For each edge e_i ,

Maximize MIN($\{ \overline{l_{pi}p_m}$, $\overline{p_m r_{pi}} \}$)

s.t. $p_i \in e_i$

r_{pi} = MAX($\{ r \mid r \in e_0$ and p_i is visible from $r \}$) and

l_{pi} = MIN($\{ l \mid l \in e_0$ and p_i is visible from $l \}$).

(2) Find R' = MIN($\{ r_{pi} \mid i=1,2,..,n-1 \}$) and

L' = MAX($\{ l_{pi} \mid i=1,2,..,n-1 \}$).

We have $\{ s_1 , s_2 \} = \{ \overline{LL'} , \overline{R'R} \}$. (see Fig. 2.11)

With a little difference from Section 2.2, we define the visibility relation diagram as follows.

Definition. According to the visibility of points on e_0 and e_i, domain $e_0 \times e_i$ can be partitioned into two equivalent classes, say R_{ei} and R_{ϕ}. For any pair of points (p_0,p_i) in R_{ei} , p_0 is visible from p_i; if (p_0,p_i) is in R_{ϕ}, they can not be seen from each other in the simple polygon P. The visibility relation diagram of e_i, short for VRD_{ei} , is the partition of domain $e_0 \times e_i$ defined by visibility between the points on edge e_0 and edge e_i (see Fig. 2.12).

The relationship between the geometric meanings and visibility relation diagram are

Fact 1: (boundary)

The boundary of R_{ei} and R_{ϕ} on domain $e_0 \times e_i$ is typically made up of pieces each

of which is a section of hyperbolic curve. It has two non–intersect parts both of which are monotone decreasing. For convenience, we define the upper boundary $F_i(x)$ as the concatenation of hyperbolic functions $f_{ij}(x)$ and the lower one $G_i(x)$ as the concatenation of hyperbolic functions $g_{ij}(x)$. Furthermore, each $f_{ij}(x)$ or $g_{ij}(x)$ can be defined by the following condition (see Fig. 2.13):

$$[\ a+by-v_j\ ] \times [\ c+dx-v_j] = 0$$

(collinear of the points on edge e_0, points on e_i and vertex v_j)

where v_j is the location vector of vertex v_j , and

$a+by$ and $c+dx$ are points respectively located on edge e_0 and edge e_i .

Fact 2: (type 1 and type 2 segments on domain $e_0 \times e_i$)

In Fig. 2.12, $F_i(1)$ is r_{i-1} while $G_i(0)$ is l_i. Hence, if $F_i(1) \geq G_i(0)$, the type 1 segment of e_i is $\overline{G_i(0)F_i(1)}$; if $F_i(1) \leq G_i(0)$, $\overline{F_i(1)G_i(0)}$ is the type 2 segment of e_i. For a point p_i on e_i , $F_i(p_i)$ is r_{pi} of p_i while $G_i(p_i)$ is l_i of p_i.

Fact 3: (local optimal solution)

Let a and b be two points on edge e_i , such that in domain $e_0 \times [a,b]$, $F_i(x)$ and $G_i(x)$ are respectively defined by a single $f(x)$ and a single $g(x)$ (see Fig. 2.14). Assume that s_{i1} should contain L while s_{i2} contains R. For segment $[a,b]$, the local optimal solution is to

Minimize $\quad MAX\{\ s_{i1}\ ,\ s_{i2}\ \}$

s.t. $\quad$ segment $\overline{ab}$ is visible from $s_{i1} \cup s_{i2}$.

In other words,

(1) $\quad$ if $p_m \in [\ \frac{f(b)+g(b)}{2}\ ,\ \frac{f(a)+g(a)}{2}\ ]$,

then $p_i \in [a,b]$ such that $|\overline{f(p_i)p_m}| = |\overline{g(p_i)p_m}|$,

i. e. $s_1 = \overline{Lg(p_i)}$ and $s_2 = \overline{f(p_i)R}$

(2) if $p_m \notin [\frac{f(b)+g(b)}{2}, \frac{f(a)+g(a)}{2}]$,

CASE 1. when $f(b) > g(a)$ (type 1 segment),

if $p_m < \frac{f(b)+g(b)}{2}$, then $s_2 = \overline{f(b)R}$ and $s_1 = L$;

if $p_m > \frac{f(a)+g(a)}{2}$, then $s_1 = \overline{Lg(a)}$ and $s_2 = R$.

CASE 2. when $f(b) < g(a)$ (type 2 segment),

If $p_m < \frac{f(b)+g(b)}{2}$, then $s_2 = \overline{f(b)R}$ and $s_1 = L$;

If $p_m > \frac{f(a)+g(a)}{2}$, then $s_1 = \overline{Lg(a)}$ and $s_2 = R$.

Actually speaking, Fact 3 is obtained by solving the following equations:

$[\, a+by_1 - v_j \,] \times [\, c+dx - v_j] = 0$ (vertex v_j and points on e_0 and e_i are collinear)

$[\, a+by_2 - v_k \,] \times [\, c+dx - v_k] = 0$ (vertex v_k and points on e_0 and e_i are collinear)

$[\, a+by_1 - p_m \,] = [\, a+by_2 - p_m \,]$ (optimal solution)

Here we parameterize the points on e_0 as $a+by_1$, $a+by_2$ and points on e_i as $c+dx$.

p_m is the position vector of midpoint p_m.

v_j and v_k are the position vectors of vertices v_j and v_k respectively.

According to above observations, the optimal solution should satisfy the following lemma.

Lemma 2.14 The optimal solution of 2–cruising guard problem is a pair of segments (s_1,s_2),

such that $\forall\, s_{i1}, s_{i1} \subset s_1$ and $\forall\, s_{i2}, s_{i2} \subset s_2$.

Now we formally outline the algorithm.

Algorithm 2.2 (2–cruising guard problem):

(1) Compute L and R by Algorithm 1;

(2) If L>R then we need only one guard and can put him on any point of $\overline{RL}$;

(3) For each e_i , compute its Visibility Relation Diagram VRD_{ei} ;

(4) For each VRD_{ei} , partition it to $O(n)$ subdomains as described in Fact 3, and compute their (s_{i1},s_{i2});

(5) Let s_1 = the longest s_{i1} obtained from step (4);

s_2 = the longest s_{i2} obtained from step (4);

(6) Output (s_1,s_2).

As mentioned in 1–cruising guard problem, step (1) needs only $O(n)$. Steps (3) and (4) can be accomplished in $k \times O(n) + (n-k) \times C$ time, where k is the number of pockets in polygon P and C is a suitable constant. For step (5), the pair of longest segments, s_1 and s_2, can be found in linear time. Hence, the time complexity of Algorithm 2.2 is bounded by $O(kn)$.

2.3.4 Remarks on Future Research

In this subsection we presented two algorithms for respectively solving 1– and 2–cruising guard problems. Recently researching results [31] show that k–cruising guard problem can also be solved by a polynomial time algorithm which is bounded by $O(2^k n^3 \log n)$. In the same technical report Tu and Kao proved the boundary curves in visibility relation diagrams form a Mobius group of functions with degree 2. By induction the critical segment is the one cruised by the [(k+1)/2]th guard if k is odd, or

by [k/2]th guard and [(k/2)+1]th guard if k is even. Hence we can find the optimal solution either on the locations which minimize the length of critical segments or on the intersections of the curves on visibility relation diagrams. A final, more general problem, which is a natural extension of our work, is: "Instead of edge e_0 we allow that k guard can cruise on a set of edges. Dose there exist a polynomial algorithm to minimize the longest segment ?"

Acknowledgment

We would like to thank Doctor Y. T Ching for careful reading of the manuscript of this subsection, and giving us many helpful suggestions, including the idea of Lemma 2.13 and the definition of type 1 segment and type 2 segment.

3. Dynamic Computational Geometry

Another direct use of the locus approach is *dynamic computational geometry*. The term "dynamic computational geometry" was original coined by Atallah's [2] to denote the study of some geometric problems which were reconsidered under the assumption that the coordinate of each input point is a function of a time variable t. The word *dynamic* was referred to the situation when the input geometric objects are moving in a prescribed manner and the word *static* for the case when they are fixed. Thus, some of the techniques for solving static computational geometry problems do not seem to help in the dynamic case, especially when we are trying to continuously update over time the information we have about the moving points. Since the paths of the moving objects are pre–defined, to answer the queries at a given time interval, we can draw a diagram by tracing their loci on domain $t \times x$ which is defined by time space t and a specific considered space x. Of cause, Locus approach does this work well. For example, in this section, we focus our attention on the *separability problem*. For the static case, it has been shown that linear separability is a linear programming problem, and can be solved in time $O(n)$ where n is the cardinality of the set of input points [15,16,24]. In dynamic case, the problem becomes much harder. Since the given points are moving, the separability crucially depends on the function of each moving point. Thus, we draw the locus of each point on domain $t \times \theta$, where θ is the angle that the path of the moving point makes with the x–axis. Then, according to the diagram on domain $t \times \theta$, we can subdivide the time space into several time intervals in which the given two point sets are linearly separable.

This section is consist of two parts. In the first part we introduce the direct use of the locus approach in dynamic computational geometry by an example: *separability problem*. Given n red points and m blue points having 1–motion in plane, the obvious

brute force approach gives an $O(mn(m+n)\log(m+n))$ time solution [2]. In Section 3.1, we propose an algorithm to find the time intervals in which the red and blue points are separable. For the points having k–motion in the plane (2–D), the algorithm can be accomplished in time $O(mN\log mN+nM\log nM+mN+nM)$, where N is $O(\lambda(n,4k))$, M is $O(\lambda(m,4k)$, and $\lambda(i,j)$ is the length of the maximal (i,j) Davenport–Schinzel sequence. If the points are moving on a line (1–D) with k–motion, the complexity can be reduced to $O(N\log N+M\log M+N+M)$ where N is $O(\lambda(n,k))$ and M is $O(\lambda(m,k)$. For a more special case, in which points are restricted to 1–motion on 1–D, we have a linear time algorithm for deciding whether they are separable. Our algorithm consists of two major parts. First, we use the method proposed by Atallah [2] (which solved the dynamic convex hull problem) to obtain a set of time intervals in which the red points are not in the interior or boundary of the convex hull of blue points, and in the same way we can enumerate the time interval set of blue points. Then, in the second part of our algorithm, these two sets of time intervals will be merged to be one final set of separable time intervals.

In the second part, since the Davenport–Schinzel sequences play an important role in the analysis and calculation of the lower envelope of collection of functions in the separability problem, we introduce its lower bound and upper bound with some applications in Section 3.2.

3.1 Separability Problem

This subsection is organized as follows. After introducing our notation and definitions, we present an algorithm for separability problems with k–motion in 2–D. Then we discusses the special case, separability problem in 1–D. Concluding remarks are given in the final part.

3.1.1 Notations and Definitions

For convenience, in the following subsections we assume the input points are moving in Euclidean space. A set of points are said to be with k–motion, if every coordinate of every moving point is a polynomial function with no more than k degree in the time variable t. Thus, for general d–dimension, each input point p_i can be described as follows:

$$p_i(t) = (X_{i1}(t), X_{i2}(t), \ldots, X_{id}(t))$$

where each

$$X_{ij}(t) = C_{ijk}t^k + C_{ijk-1}t^{k-1} + \cdots + C_{ij0}t^0 \quad , 1 \le j \le d.$$

and C_{ijh} are constant coefficients of polynomial $X_{ij}(t)$ for $0 \le h \le k$. At time instance t_0, the configuration of input point set P is denoted as $P(t_0)$.

For separability, we define the follows.

Definition. Two set of points R and B in E^d are said to be *linearly separable* at time instance t_0 if there exists a (d–1)–dimension hyperplane L such that $R(t_0)$ and $B(t_0)$ lie on opposite sides of L.

The separability problem in dynamic computational geometry now can be stated

as: given two set of points having k–motion in d–dimension, try to find the time intervals during which R and B are linearly separable. A *separable time interval* is a time interval $[t_1,t_2]$ during which R and B can be linearly separated .

In the following we would like to introduce some notations related to convex hull which will play an important role in this section. Fig. 3.1 illustrates the convex hull of point set B. Let CH(B) denote this convex hull. For the vertices on the convex hull, such as v_1, v_2,v_3, v_4, and v_5 in Fig. 3.1, we name them *convex points*, and classify them into four types: *top*, *bottom*, *left* and *right* according to the following specifications.

(1) A convex point is a *top convex point* if no other points lie above it, i.e. the top convex point has the largest y–value among all points. We choose the rightmost one to be the top convex point if there are more than one point with this y–value. In Fig. 3.1, v_1 is a top convex point.

(2) A *bottom convex point* is a convex point of the smallest y–value. If there are more than one convex point with this y–value, we choose the leftmost one to be the bottom convex point. In Fig. 2.1, v_3 is a bottom convex point.

(3) A *left convex point* is a convex point on the counterclockwise path from top convex point to bottom convex point. v_2 is a left convex point in Fig. 3.1.

(4) A *right convex point* is a convex point on the clockwise path from top convex point to bottom convex point. v_4 and v_5 are right convex points in Fig. 3.1.

Let $|U|$ be the length of an (n,s) Davenport–Schinzel sequence (also see section 3.2 for details) [13,25]. Then, the function

$\lambda(n,s)$ = MAX { $|U|$: U is an (n,s) Davenport–Schinzel sequence }.

Suppose that F=$\{f_1,f_2,...,f_n\}$ is a set of n real–valued continuous functions defined on a common interval I, where for every two distinct functions f_i and f_j, they intersect in at most s points. We define $l(t)$ = MIN $\{f_i(t)$: i=1..n and $t\in I\}$ to be the *lower envelope* of

F. Note that $\ell(t)$ is typically made up of "pieces" each of which is a section of $f_i(t)$. The length of a lower envelope, $|\ell(t)|$, is the number of pieces in $\ell(t)$. Attalah [2] showed that $\lambda(n,s)$ is the upper bound of $|\ell(t)|$. Attalah [2], Sharir [13,25] and Hart [13] have proved the following bounds.

$\lambda(n,1) = n$, and $\lambda(n,2) = 2n-1$;

$\lambda(n,3) = \theta(n \cdot \alpha(n))$, and $\lambda(n,s) = O(n \cdot \alpha(n)^{O(\alpha(n)^{s-3})})$.

These results will be used in the rest of this section.

3.1.2 Separability of k–motion in 2–dimension

In this subsection, we first state some properties of separability, and then iteratively use Attalah's method (which solved the dynamic convex hull problem) to obtain the set of separable time intervals for two given point set.

Let R and B be two given sets of points with k–motion in 2–D plane. Assume that $|R|=n$ and $|B|=m$. A crucial criterion for linear separability is provided by the following theorem.

Theorem 3.1 ([30] Store and Witzgal (1970), Theorem 3.3.9) Two sets of points R and B are *linearly separable* if and only if their convex hulls do not intersect and the interior of these two convex hulls are mutually disjoined.

We immediately have Corollary 3.2.

Corollary 3.2 If R and B are linearly separable at time instance t_0, then each point b_i of B is a convex vertex of the convex hull of set $R \cup b_i$ at t_0, and each point r_i of R is also a convex vertex of the convex hull of set $B \cup r_i$ at t_0, i.e.

$$b_i(t_0) \in CH(\, R(t_0) \cup b_i(t_0)\,) \quad \text{for } \forall\, b_i \in B,$$

and $$r_j(t_0) \in CH(\, B(t_0) \cup r_j(t_0)\,) \quad \text{for } \forall\, r_j \in R.$$

Since Corollary 3.2 is only a necessary condition, it is not strong enough to detect the intersection between the interiors of two convex hulls. Fig. 3.2 illustrates a counterexample in which R and B meet the requirement of Corollary 3.2 but are not linearly separable. Hence, we propose the following theorem.

Theorem 3.3 R and B are linearly separable at time instance t_0 *iff* one of the following conditions is true

(i) each point $b_i(t_0)$ of $B(t_0)$ is a *top*, *bottom*, or *right* convex point of $CH(\, R(t_0) \cup b_i(t_0)\,)$, and each point $r_j(t_0)$ of $R(t_0)$ is a *top*, *bottom*, or *left* convex point of $CH(\, B(t_0) \cup r_i(t_0)\,)$;

(ii) each point $b_i(t_0)$ of $B(t_0)$ is a *top*, *bottom*, or *left* convex point of $CH(\, R(t_0) \cup b_i(t_0)\,)$, and each point $r_j(t_0)$ of $R(t_0)$ is a *top*, *bottom*, or *right* convex point of $CH(B(t_0) \cup r_j(t_0))$.

Proof: Here we prove condition (i) only.

If sets of points, R and B, are linearly separable at time instance t_0, by definition, there is a line L, say $y=ax+b$, such that $R(t_0)$ and $B(t_0)$ lie on its opposite sides. Without loss of generality, assume that the points of $B(t_0)$ lie on the half–plane $y<ax+b$ while the points of $R(t_0)$ lie on the half–plane $y>ax+b$. Let y_t and y_b be respectively the largest and the smallest y–values of points in $R(t_0)$. It is easy to see that set $B(t_0)$ can be classified into three subsets, B_t, B_r, and B_b, where

$$B_t \subset \{(x,y) \mid y<ax+b \text{ and } y>y_t\}\,,$$

$$B_r \subset \{(x,y) \mid y<ax+b\,,\ y<y_t\,, \text{ and } y>y_b\} \text{ and}$$

$$B_b \subset \{(x,y) \mid y<ax+b \text{ and } y<y_b\}.$$

Condition (i) is a immediate consequence of above statements and Corollary 3.2 (see Fig. 3.3).

Conversely, if condition (i) holds, we will claim that the convex hull of set $R(t_0)$ and the convex hull of $B(t_0)$ do not intersect, and the interiors of these two convex hulls are mutually disjoined. Let y_t and y_b be respectively the largest and the smallest y–values of points in $R(t_0)$. Since condition (i) holds, once again we can subdivide set B into three subsets, say B_t, B_b, and B_r, where

$B_t \subset \{(x,y) |\ y>y_t\}$,

$B_r \subset \{(x,y) |\ y<y_t \text{ and } y>y_b\}$ and

$B_b \subset \{(x,y) |\ y<y_b\}$.

Suppose that $\overline{b_i(t_0)b_j(t_0)}$ is an edge of $CH(B(t_0))$, which intersects $CH(R(t_0))$. By the fact that $b_i(t_0)$ and $b_j(t_0)$ are the *top, bottom,* or *right* convex points, $\overline{b_i(t_0)b_j(t_0)}$ should intersects $CH(R(t_0))$ on two and exactly two edges (see Fig. 3.4). This implies that there exists a point $r_k(t_0)$ of $R(t_0)$ where

(a) its y–value is between the y–values of $b_i(t_0)$ and $b_i(t_0)$, and

(b) it is located on the right side of $\overline{b_i(t_0)b_j(t_0)}$.

(a) and (b) obviously contradict to condition (i), "point $r_k(t_0)$ is a *top, bottom,* or *left* convex point of $CH(\ B(t_0) \cup r_k(t_0)\ )$." Thus, $CH(B(t_0))$ should not intersect $CH(R(t_0))$. For the interiors, if these two convex hulls do not intersect with each other, but parts of their interior are in common, there is only one case meeting this requirement : one convex hull with its interior is properly contained in the other one. This also contradicts to the assumption of convex points. Thus, R and B are linearly separable at time instance t_0. ■

Our next goal is to compute the time intervals during which a given point belongs to the convex hull.

Let $\theta_{ij}(t)$ be the angle determined by $<\overline{b_i(t)r_j(t)}>$ and x–axis at time t where $-\pi<\theta_{ij}(t)\leq+\pi$. Define α_i, β_i, γ_i, and δ_i as follows [2]:

$$\alpha_i(t)=\underset{j}{\mathrm{MIN}}\{\ \theta_{ij}(t)\ |\ \theta_{ij}(t)\geq 0\ \}$$
$$\beta_i(t)=\underset{j}{\mathrm{MAX}}\{\ \theta_{ij}(t)\ |\ \theta_{ij}(t)\geq 0\ \}$$
$$\gamma_i(t)=\underset{j}{\mathrm{MIN}}\{\ \theta_{ij}(t)\ |\ \theta_{ij}(t)<0\ \}$$
$$\delta_i(t)=\underset{j}{\mathrm{MAX}}\{\ \theta_{ij}(t)\ |\ \theta_{ij}(t)<0\ \}$$

If all $\theta_{ij}(t)$ are negative, $\alpha_i(t)$ and $\beta_i(t)$ are both undefined. Similarly, if all $\theta_{ij}(t)$ are positive, $\gamma_i(t)$ and $\delta_i(t)$ are both undefined.

Lemma 3.4 For $CH(\ R(t) \cup b_i(t)\ )$, point $b_i(t)$ is

a right convex point *iff* $\alpha_i(t)-\delta_i(t)\geq\pi$,

a left convex point *iff* $\beta_i(t)-\gamma_i(t)\leq\pi$,

a top convex point *iff* $\alpha_i(t)$ and $\beta_i(t)$ are undefined, and

a bottom convex point *iff* $\gamma_i(t)$ and $\delta_i(t)$ are undefined.

Proof: Directly derived from Lemma 4.7 in [2].

Note that each of the functions α_i, β_i, γ_i, and δ_i contains $O(n)$ transitions and jump discontinuities, hence, each of them has no more than $\lambda(n,4k)$ pieces. If we count the time needed to find the roots of polynomial with $O(k)$ degree as $O(1)$, a divide–and–conquer algorithm, in time $O(\lambda(n,4k)\log\lambda(n,4k))$, can compute the set of time intervals during which $b_i(t)$ is a top, bottom, left, or right convex point of $CH(\ R(t) \cup b_i(t)\ )$ [2]. Denote the set of these time intervals as I_{bi}. After executing the computations for all $b_i(t)$, we have n sets of time intervals, say I_{b1}, I_{b2},..,I_{bn} (see Fig. 3.5). Now intersect these n sets to be a new set of time intervals I_b in the following way.

(1) A time interval $[t_1,t_2]$ belongs to I_b if $[t_1,t_2]\subset[a,b]$, where $[a,b]\in I_{bi}$ for i=1,2,..,n.

(2) A time interval $[t_1,t_2]$ in I_b is assigned to "*R*–type" if every $b_i(t)$ is either a top, bottom, or right convex point of $CH(\ R(t) \cup b_i(t)\ )$ where $t\in[t_1,t_2]$.

(3) A time interval $[t_1,t_2]$ in I_b is assigned to "*L*–type" if every $b_i(t)$ is either a top,

bottom, or left convex point of $CH(\ \mathrm{R}(t) \cup b_i(t)\ )$ where $t \in [t_1, t_2]$.

Fig. 3.6 illustrates the set I_b obtained by intersecting all I_{bi}'s according to above steps. In the same way, we can compute I_{rj} for each $r_j(t)$, and get their intersection I_r.

By Theorem 3.3, it is easy to understand that the final result I, the set of separable time intervals for sets R and B, can be obtained by intersecting I_b and I_r in the following way (see Fig. 3.7):

a time interval $[t_1,t_2] \in I$ if $[t_1,t_2] \subset [a,b]$ and $[t_1,t_2] \subset [c,d]$ where $[a,b] \in I_b$ and $[c,d] \in I_r$.

In the following, we shall summarize the algorithm, and give an analysis of its time complexity.

Algorithm 3.1 . Given two sets of points $\mathrm{R}=\{r_1, r_2, .., r_n\}$ and $\mathrm{B}=\{b_1, b_2, .., b_m\}$ having k–motion in 2–D. We compute the set of separable time intervals.

Step (1) For each b_i, compute I_{bi}, the set of time intervals during which b_i is a convex point of $CH(\ \mathrm{R}(t) \cup b_i(t)\ )$;

Step (2) Intersect all I_{bi}'s to get I_b, the set of time intervals with either R–type or L–type;

Step (3) For each r_j, compute I_{rj}, the set of time intervals during which r_j is a convex point of $CH(\ \mathrm{B}(t) \cup r_j(t)\ )$;

Step (4) Intersect all I_{rj}'s to get I_r, the set of time intervals with either R–type or L–type;

Step (5) Intersect I_b and I_r to get the final result I, the set of separable time intervals.

The running time of this algorithm is clearly polynomial in n and m. Let M be $O(\lambda(m,4k))$ and N be $O(\lambda(n,4k))$. Specifically, Step (1) can be accomplished, for each b_i

in time $O(M\log N)$, in overall time $O(m\cdot M\log N)$ [2]. In Step (2), we need to merge m sets of time intervals produced in Step (1), and identify the type (R–type or L–type) for each time interval. Since each I_{bi} is a sorted list with $O(N)$ time intervals, this step can be carried out by a linear merging algorithm for m sorted lists in time $O(m\cdot N\cdot\log m)$. In the same way, we know that Step (3) and Step (4) need $O(n\cdot M\log M)$ and $O(n\cdot M\cdot\log n)$ time respectively. Since list I_b has $O(m\cdot N)$ sorted elements while there are $O(n\cdot M)$ sorted elements in list I_r, Step (5) can also be performed by a linear merging procedure in time $O(mN+nM)$. Hence, totally **Algorithm 3.1** can be accomplished in time $O(mN\cdot\log mN + nM\cdot\log nM + mN+nM)$.

In [25], Sharir gave the almost linear upper bounds for N and M:

$M = O(m\cdot\alpha(m)^{O(\alpha(m)^{4k-3})})$ and $N = O(n\cdot\alpha(n)^{O(\alpha(n)^{4k-3})})$, where $\alpha(i)$ is the functional inverse of Ackermann's function. The function $\alpha(i)$ is very slowly growing, but tends to infinity with i. Note that $\alpha(i)\leq 4$ for all $i\leq A$ which is a tower with 65536 2's,

i.e. $A = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ ⟶ with 65536 2's in the exponential tower.

Thus $\alpha(i)\leq 4$ is suitable for all practical purposes. Assume that the sizes of sets R and B are almost equal. The formula of time complexity for **Algorithm 3.1** can be simplified to $O(C\cdot n^2\cdot\log n)$ where C is a function of $\alpha(n)$.

3.1.3 Separability of k–motion in 1–dimension

We now consider a more special case, where input points are specified to be k–motion in 1–D. Suppose that one point can run over another point without collisions. Each point p_i can be described as follows.

$$p_i(t) = C_{ik}t^k + C_{ik-1}t^{k-1} + \cdots + C_{i0}t^0$$

where each C_{ih} is a constant coefficient. At time t, define

$$b_{max}(t) = \mathrm{MAX}_i\{b_i(t)\},$$

$b_{min}(t) = \underset{i}{\text{MIN}}\{b_i(t)\}$,

$r_{max}(t) = \underset{j}{\text{MAX}}\{r_j(t)\}$ and

$r_{min}(t) = \underset{j}{\text{MIN}}\{r_j(t)\}$.

Since $CH(\text{R}(t)) = \{r_{max}(t), r_{min}(t)\}$, $b_i(t)$ is a convex point of $CH(\text{R}(t) \cup b_i(t))$ if and only if $b_i(t) < r_{min}(t)$ or $b_i(t) > r_{max}(t)$, where the first condition specifies $b_i(t)$ to be a left convex point while the second condition classifies $b_i(t)$ into the set of right convex points. Hence, we can immediately deduce a more simple property from Lemma 3.4.

Lemma 3.5 Assume that the points in R and B are k–motion in 1–D. At time t_0, $\text{R}(t_0)$ and $\text{B}(t_0)$ are linearly separable *iff* one of the following conditions is true

(i) $b_{max}(t_0) < r_{min}(t_0)$ or (ii) $b_{min}(t_0) > r_{max}(t_0)$.

By Lemma 3.5, Algorithm 3.1 can be simplified to the following version.

Algorithm 3.2 Given two sets of points $\text{R}=\{r_1, r_2, .., r_n\}$ and $\text{B}=\{b_1, b_2, .., b_m\}$ having k–motion in 1–D. We compute the set of separable time intervals.

Step (1) Compute $r_{min}(t)$ and $r_{max}(t)$ for $t \in [0, \infty)$, the lower envelope and upper envelope of $\text{R}(t)$ respectively;

Step (2) Compute $b_{min}(t)$ and $b_{max}(t)$ for $t \in [0, \infty)$, the lower envelope and upper envelope of $\text{B}(t)$ respectively;

Step (3) Comput I_{rb}, the set of time intervals during which $r_{max}(t) < b_{min}(t)$;

Step (4) Comput I_{br}, the set of time intervals during which $b_{max}(t) < r_{min}(t)$;

Step (5) Merge I_{rb} and I_{br} to be the final result I, the set of separable time intervals.

Since the envelope functions of $\text{R}(t)$ and $\text{B}(t)$ respectively have N and M pieces of

polynomial functions, where N is $O(\lambda(n,k))$ and M is $O(\lambda(m,k))$, Step (1) and Step (2) can be performed by a simple divide–and–conquer technique in time $O(N\log N + M\log M)$ [2]. To accomplish Step (3) and Step (4), we need only $O(M+N)$ time by executing a standard merging procedure. For Step (5), the time to combine I_{rb} and I_{br} can be dominated by previous time order. Hence, the time complexity of Algorithm 3.2 is $O(\ N\log N + M\log M + M+N\ )$. Assume that R and B have the equal size, n. We have a more simple formula, $O(C\cdot n\log n)$, where C is a function of $\alpha(n)$.

For another special case of separability problem, we consider the problem of which points are moving in 1–D with 1–motion. The position of each p_i now is a linear function of time t,

$p_i(t) = C_{i1}t^1 + C_{i0}t^0$ where C_{i1} and C_{i0} are all constant coefficients.

Hence, the upper envelope of R(t), say $r_{max}(t)$, is a concave piecewise linear function while its lower envelope, $r_{min}(t)$, is a convex one. Since there is at most two separable time intervals in this simple case, separability here can be transformed to a linear programming problem as follows. Specifically, given two sets of points having 1–motion in 1–D, R=$\{r_1,r_2,..,r_n\}$ and B=$\{b_1,b_2,..,b_m\}$, we therefore seek the two optimal solutions, one for "Maximize t" and the other for "Minimize t", satisfying the conditions

(1)
$$\begin{aligned} y &\leq r_i(t) && \text{where i=1,2,...,n} \\ y &\geq b_j(t) && \text{where j=1,2,...,m} \\ t &\geq 0 \end{aligned}$$

and we also find another pair of minimum and maximum of t meeting the requirements

(2)
$$\begin{aligned} y &\geq r_i(t) && \text{where i=1,2,...,n} \\ y &\leq b_j(t) && \text{where j=1,2,...,m} \\ t &\geq 0 \end{aligned}$$

Without loss of generality we assume that the optimal solutions of (1) and (2) are

respectively t_{max1}, t_{min1}, t_{max2}, and t_{min2}. It is easy to understand that $[t_{min1}, t_{max1}]$ and $[t_{min2}, t_{max2}]$ are separable time intervals. For the time complexity, since Megiddo [15,16] has proposed a linear time algorithm to solve linear programming problems with two variables, the linearly separable problem with 1–motion in 1–D can be solved in time $O(m+n)$.

3.1.4 Concluding Remarks

In this subsection we proposed an algorithm to solve the *separability problem* in dynamic computational geometry. For input points having k–motion in 2–D, our algorithm can be accomplished in time $O(mN \cdot \log mN + nM \cdot \log nM + mN+nM)$, where N and M is almost linearly proportional to the size of input set. In the special case, the points are k–motion in 1–D, the complexity can be reduced to $O(N \cdot \log N + M \cdot \log M + N+M)$. If the points are restricted to 1–motion in 1–D, our algorithm needs only linear time $O(m+n)$.

In d–dimension, the dynamic separability problems are much harder than static ones. For static version, linear separability can always be transformed to a linear programming problem [24]. Specifically, given two sets of points in d–dimension, say $R=\{r_1,r_2,...,r_n\}$ and $B=\{b_1,b_2,...,b_m\}$, we seek a $(d-1)$–dimension hyperplane

$$\sum_i C_i X_i = 0$$

satisfying the conditions

$$\sum_i C_i X_{ir_j} \leq 0 \qquad \text{for } r_j \in R$$

$$\sum_i C_i X_{ib_j} \geq 0 \qquad \text{for } b_j \in B.$$

This is clearly a linear programming, which can be solved in linear time by Megiddo's algorithm [15,16]. On the contrary, the dynamic linear separability is not a simple "Yes" or "No" problem. We need describe the whole set of separable time intervals. For the

problems moving in 1–D and 2–D, by Lemma 3.4 and Lemma 3.5, an efficient algorithm can be designed successfully. Although Theorem 1 holds for any fixed d–dimension, it is still very difficult to use this *iff* condition to construct an algorithm, even for 3–D case. It could be an interesting future research topic to find good characteristics of linear separability in dynamic computational geometry for high dimension.

3.2 Davenport–Schinzel Sequences and Envelopes of a Set of Functions

Davenport–Schinzel sequences are interesting and powerful combinatorial structures that arise in the analysis and calculation of the lower envelope of collections of functions, and therefore have applications in many geometric problems that can reduced to the calculation of such an envelope. The sequence has originally been posed by Davenport and Schinzel [8]. Their interest in it arose from its connection to the analysis of solutions of linear differential equations. Recently, Atallah [2] has raised it again independently, because of its significance for problems in dynamic computational geometry. For a formal definition [13], consider the following combinatorial problem: Let n, s be positive integers. A sequence $U = (u_1, u_2, .., u_m)$ of integers is an (n,s) *Davenport–Schinzel sequence*, if it satisfies the following conditions:

(i) $1 \le u_i \le n$ for each i.

(ii) For each $i \le m$ we have $u_{i-1} \neq u_i$.

(iii) There do not exist $s+2$ indices $1 \le i_1 < i_2 < ... < i_{s+2} \le m$ such that $u_{i1} = u_{i3} = u_{i5} = ... = a$, $u_{i2} = u_{i4} = u_{i6} = ... = b$, and $a \neq b$.

We will write $|U| = m$ for the length of the sequence U. Define

$\lambda_s(n) = \max \{ |U| : U \text{ is an } (n,s) \text{ Davenport–Schinzel sequence} \}$.

Roughly speaking, an (n,s) Davenport–Schinzel sequence is a sequence composed of n symbols with the properties that no two adjacent elements are equal and that it does not contain as a subsequence any alternation of two distinct symbols of length $s+2$. The main goal in the analysis of these sequences is to estimate their maximal possible length for any given values of the parameters n and s.

The important of (n,s) Davenport–Schinzel sequences lies in their relationship to the combinatorial structure of the envelope of a set of functions. We briefly described as

follow. For more details, please see [2,13,29]. Let $\{ f_1, f_2, \ldots, f_n \}$ be a set of real–valued continuous functions defined on the real line. Each pair of functions intersect in at most s points. From the indices of the functions, we pick a sequence of indices in the order in which these functions attain their lower envelope from left to right. This sequence is actually an (n,s) Davenport–Schinzel sequence. On the other hand, any (n,s) Davenport–Schinzel sequence can be realized in this way for an appropriate collection of n continuous nuivariate functions each pair of which intersect in at most s points.

The crucial property of these sequences is that, for a fixed s, the maximum length $\lambda_s(n)$ of an (n,s) sequence if "practically linear" in n [2,13,25]. More precisely,

$$\lambda_1(n) = n, \text{ and } \lambda_2(n) = 2n-1;$$
$$\lambda_3(n) = \theta(n \cdot \alpha(n)), \text{ and } \lambda_s(n) = O(n \cdot \alpha(n)^{O(\alpha(n)^{s-3})}),$$

where $\alpha(i)$ is the functional inverse of Ackermann's function. The function $\alpha(i)$ is very slowly growing, but tends to infinity with i. Note that $\alpha(i) \leq 4$ for all $i \leq A$ which is a tower with 65536 2's,

i.e. $A = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \longrightarrow$ with 65536 2's in the exponential tower.

Thus $\alpha(i) \leq 4$ is suitable for all practical purposes.

4. Conclusions

In this paper we have attempted to survey a number of different paradigms for the design and analysis of geometric algorithms by the generalized locus approach. In the following paragraphs we shall make a conclusions on this approach. This conclusion is divided into three parts: design techniques, data structures and analysis techniques

For design techniques, the main idea of the locus approach is to construct the relationship between the objects we concern. Generally speaking, this relationship is depicted by a combinatorial structure. Furthermore this relationship also defines a equivalent relation on a specific domain on which we shall perform queries. According to this partition, a query can usually be transformed to a location problem in two dimensional domain. Another important thing in using the locus approach is to find the functions for boundaries of the subdivisions. Since the spirit of the locus approach is based on the assumption that all the points, which are on the domain we concern, are moving, we should choose the appropriate parameters to describe this phenomenon. For example, in section 2.1 and section 2.2, finding all shortest path edge sequences on the surface of a convex polyhedron, we subdivide domain $Z = e_s \times e_e$, all possible pairs of starting and ending points, by shortest path edge sequences and define the boundaries of the equivalent classes by hyperbolic curves. By the same concept, we construct the visibility relation diagram to solve the cruising guard problem in section 2.3. For dynamic computational geometry, especially for the separability problem, the partition is on the time domain and is defined by the intersections of loci of moving objects. Two time instances are in the same time interval if the blue and red point sets are separable in the same configuration at these two time instances. Hence, there are two necessary things when we use the locus approach: define the partition and depict the boundary of each equivalent class.

For the data structures, one drawback of the locus approach is that sometimes the size of the partitioning structure that we need to compute is too large. For example, each visibility relation diagram occupies $O(n^2)$ space. If we want to construct a partition on domain $Z = f_m \times f_n$ (section 2.2.3) to reflect the visible relationship between the points on two faces, its space complexity may be even worse than $O(n^4)$. On the contrary, in dynamic computational geometry we have a better results. Usually we should compute the envelope of collections of functions and the size of the envelope if almost linear, $O(n \cdot \alpha(n))$, to the number of functions we concern. To generalize the partition on d dimensions, we need a tool to join, split, union, intersect equivalent classes and still maintain the topological orders. These interesting topics can be found in Dobkin's paper [7] for three dimensions and Brisson's paper [5] for d dimensions.

The time complexity of the locus approach crucially depends on two factors: the number of equivalent classes on partition domains and the time to depict the boundaries of equivalent classes. For example, the time complexity of finding all shortest path edge sequences on the surface of a convex polyhedron is dominated by the time to construct visibility relation diagrams. Since each equivalent class in this partition is path connected, the size of the partition equals to the number of equivalent classes, $O(n^2)$. Otherwise, the upper bound can not be reduced [28]. To depict the boundaries, the crucial point is the degree of the boundary functions. In the examples of this paper, we only discusses the hyperbolic functions. Hence, the time complexity to compute the roots is assumed to be a constant time. If the degree of function is more than four, the solution has no close forms. We need make another assumption on the time complexity when solving the roots or intersections of these high degree curves.

The locus approach is widely used in computational geometry algorithms, and plays a significant role in several problems. We hope that in time more and more tools from geometry will be applied to this approach.

References

[1] D. Avis and G. T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," *IEEE Transaction on Computers*, (1981) 12: 910 – 914.

[2] M. Atallah, "Dynamic computational geometry," In *Proceedings of the 23th Symposium on Foundations of Computer Science*, 1983, pp. 92 – 99.

[2] M. Blum, R. W. Floyd, V. Pratt, R. Rivest, and R. E. Tarjan, "Time bounds for selection," *Journal of Computer and Systems Science*, (1973) 7: 448 – 461.

[4] J. L. Bentley and Th. Ottman, "Algorithms for reporting and counting geometric intersections," *IEEE Transaction on Computers*, (1979) 28(9): 643 – 647.

[5] E. Brisson, "Representing geometric structures in d dimensions: topology and order," In *Proceedings of the 5th ACM Symposium on Computational Geometry*, June, 1989, pp. 218 – 227.

[6] E. W. Dijkstra, "A note on two Problems in connection with graphs," *Numerische Mathematik*, (1959) 1: 269 – 271.

[7] D. P. Dobkin and M. J. Laszlo, "Primitives for the manipulation of three–dimensional subdivisions," *Algorithmica*, (1989) 4: 3 – 32.

[8] H, Davenport and A. Schinzel, "A combinatorial problem connected with differential equations," *Amer. Journal of Math.*, (1965) 87: 684 – 694.

[9] L. Guibas, J. Herchberger, K. Leven, M. Sharir, R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside simple polygons," *Algorithmica* (1987) 2: 209 – 233.

[10] L. J. Guibas and J. Stolfi, "Ruler, Compass and Computer – The design and Analysis of Geometric Algorithms," in *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, Vol. F40, R. A. Earnshaw, ed.,

Spring–Verlag Berlin Heidelberg, 1988, pp. 111 – 165.

[11] Y. H. Hwang, R. C. Chang, and H. Y. Tu, "Finding all shortest path edge sequences on a convex polyhedron," Technical Report, Academia Sinica, Taipei, Taiwan, Republic of China, 1989.

[12] H. J. Hwang, R. C. Chang, and H. Y. Tu, "The separability problem in dynamic computational geometry," Springer Verlag series *Lecture Notes in Computer Science: Workshop on Algorithm and Data Structures*, Carleton University, Ottawa, Canada, August 1989.

[13] S. Hart and M. Sharir, "Nonlinearity of Davenport – Schinzel sequences and of generalized path compression schemes," *Combinatorica*, (1986) 6(2): 151 – 17.

[14] D. Leven and M Sharir, "On the number of critical free contacts of a convex polygonal object moving in two –dimensional polygonal space," *Discrete & Computational Geometry*, (1987) 2: 255 – 270.

[15] N. Megiddo, "Linear–time algorithms for linear programming in R^3 and related problems," *SIAM J. Comput.*, (1983) 12(4): 759 – 776.

[16] N. Megiddo, "Linear–programming in linear time when the dimension is fixed," *JACM*, (1984) 31(1): 114 – 127.

[17] J. S. B. Mitchell, "Planning shortest paths," Ph. D. Thesis, Stanford University, 1986.

[18] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM Journal* of *Computing*, (1987) 16: 647 – 668.

[19] D. M. Mount, "On finding shortest paths on convex polyhedra," Technical report, University of Maryland, 1985.

[20] D. M. Mount, "The number of shortest paths on the surface of a polyhedron," Technical report, University of Maryland, 1987.

[21] J. O'Rourke, S. Suri, and H. Booth, "Shortest paths on polyhedral surfaces,"

Manuscript, Johns Hopkins University, 1984.

[22] J. O'Rourke and C. Schevon, "Computing the geodesic diameter of a 3 – polytope," In *Proceedings of the 5th ACM Symposium on' Computational Geometry*, June, 1989, pp. 370 – 379.

[23] C. H. Papadimitriou, "An algorithm for shortest-path motion in three dimensions", *Inf. Proc. Lett.*, (1985) 20: 259 – 263.

[24] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer–Verlag, New York, 1985.

[25] M. Sharir, "Almost linear upper bounds on the length of general Davenport–Schinzel sequences," *Combinatorica*, (1987) 7(1): 131 – 143.

[26] M. Sharir, "On shortest paths amidst convex polyhedra," *SIAM J. Comput.*, (1987) 16(3): 561 - 572.

[27] C. Schevon and J. O'Rourke, "The number of maximal edge sequences on a convex polytope," In *Proceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing*, University of Illinois at Urbana–Champaign, October 1988, pp. 49 – 57.

[28] C. Schevon and J. O'Rourke, "An algorithm to compute edge sequences on a convex polytope," Technical Report JHU89–03, The Johns Hopkins University, 1989.

[29] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," *SIAM J. Comput.*, (1986) 15(1): 193 - 215.

[30] J. Stoer and C. Witzgall, *Convexity and optimization in finite dimensions I*, Springer–Verlag, New York, 1970.

[31] H. Y. Tu and M. D. Kao, "Cruising guards problem," Technical Report, Academia Sinica, Taipei, Taiwan, Republic of China, 1989.

[32] R. E. Tarjan and C. J. Van Wyk, "An $O(n\log\log n)$–time algorithm for

triangulating a simple polygon," *SIAM J. Comput.*, (1988) 17(1): 143 – 178.

[33] F. A. Valentine, "Minimal sets of visibility," *Proc. Amer. Math. Soc.*, (1953) 4: 917 – 921.

[34] A. Wiernik and M. Sharir, "Planar realizations of nonlinear Davenport – Schinzel sequences by segments," *Discrete & Computational Geometry*, (1988) 3: 15 – 47.

List of Figures

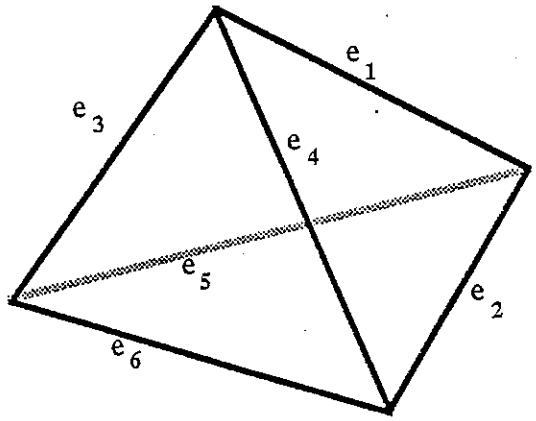

Fig. 2.1a A given convex polyhedron P.

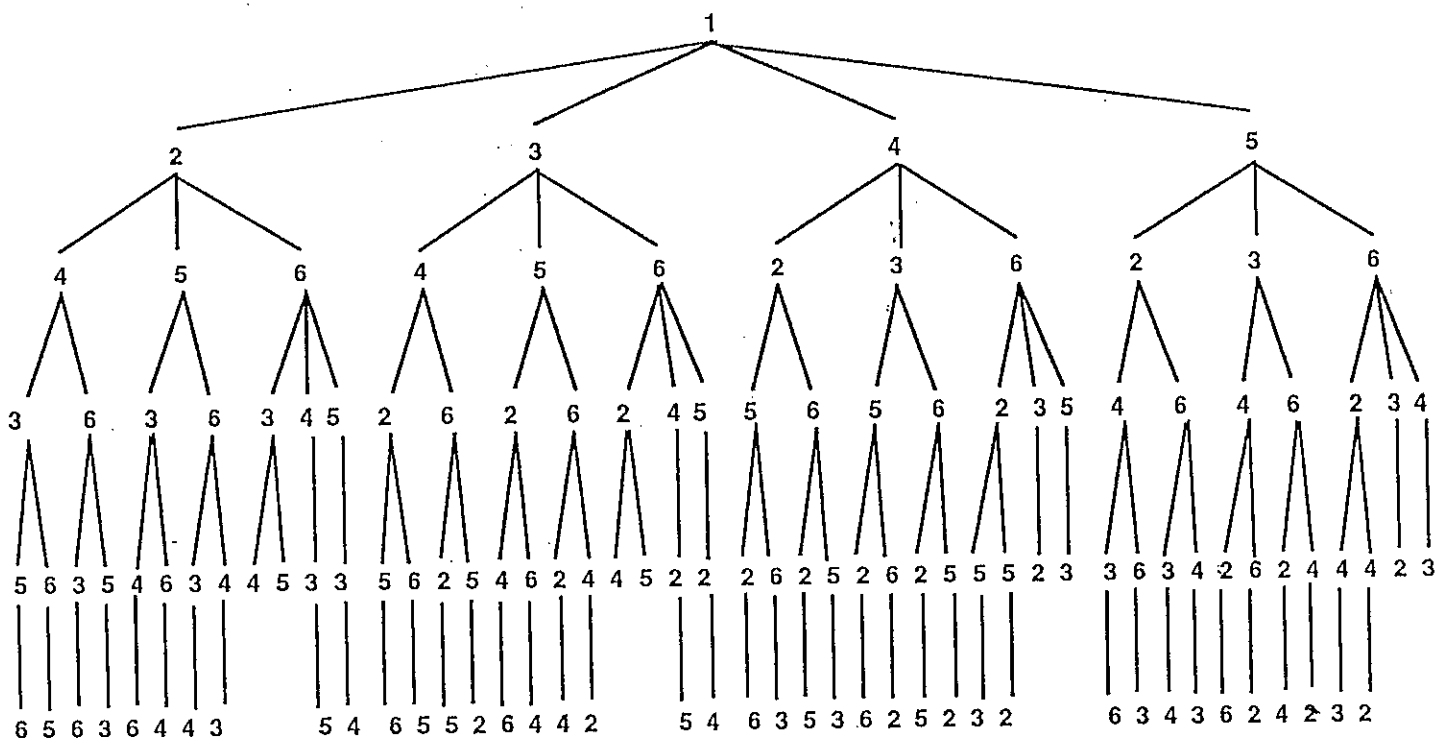

Fig. 2.1b The edge sequence tree of the convex polyhedron with starting edge e_1.

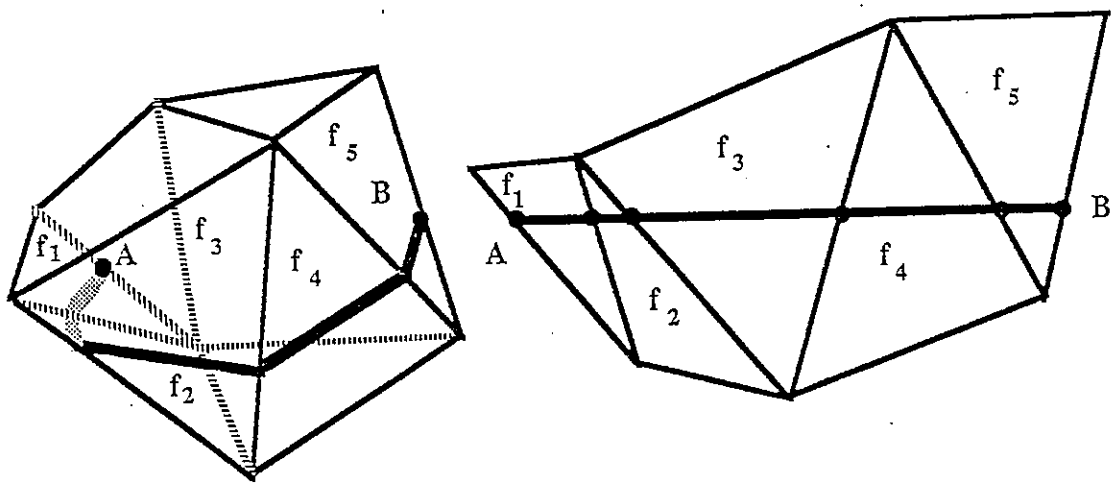

Fig. 2.2 The planar unfolding relative to edge sequence $(e_s\ e_1\ e_2\ e_3\ e_4\ e_e)$.

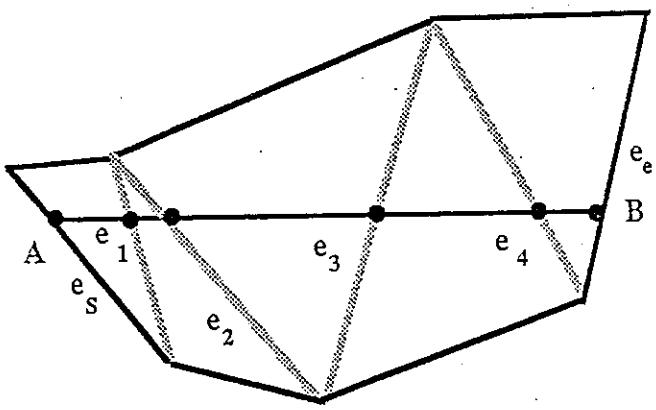

Fig. 2.3a Simple polygon G_1

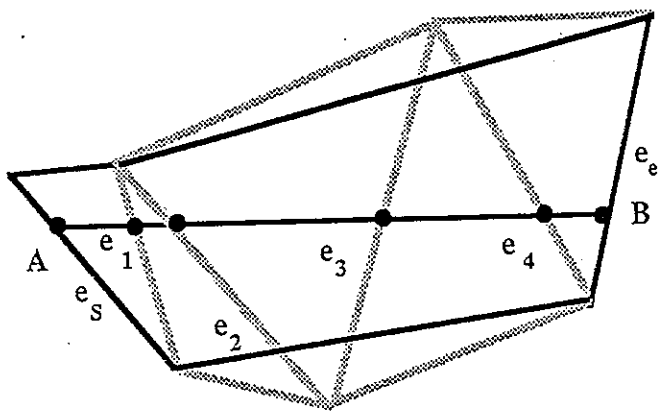

Fig. 2.3b Simple polygon G_2

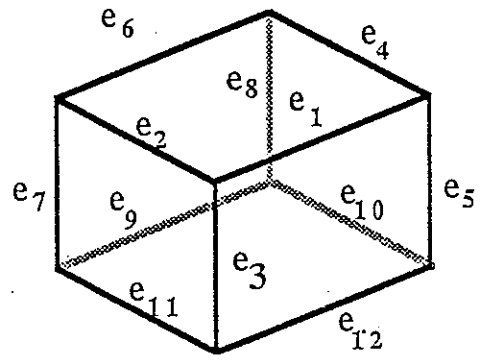

Fig. 2.4a A given rectangular polyhedron.

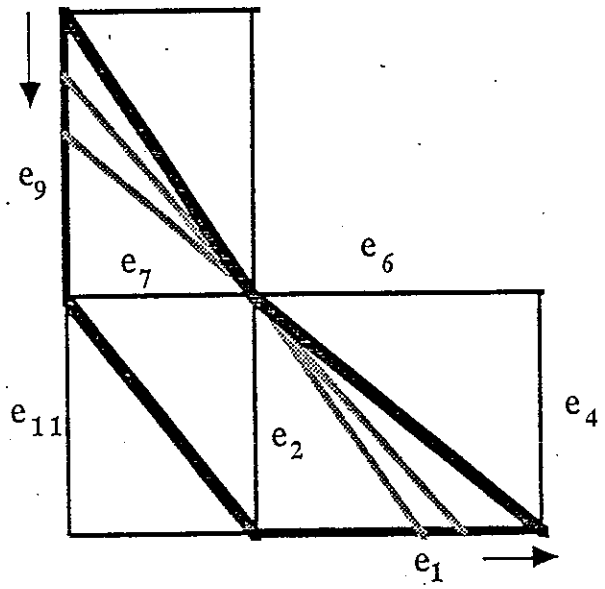

Fig. 2.4b The planar unfolding relative to edge sequence ($e_1\ e_2\ e_7\ e_9$).

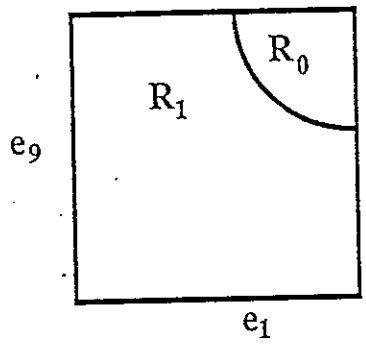

Fig. 2.4c The visibility relation diagram of Fig. 2.4b.

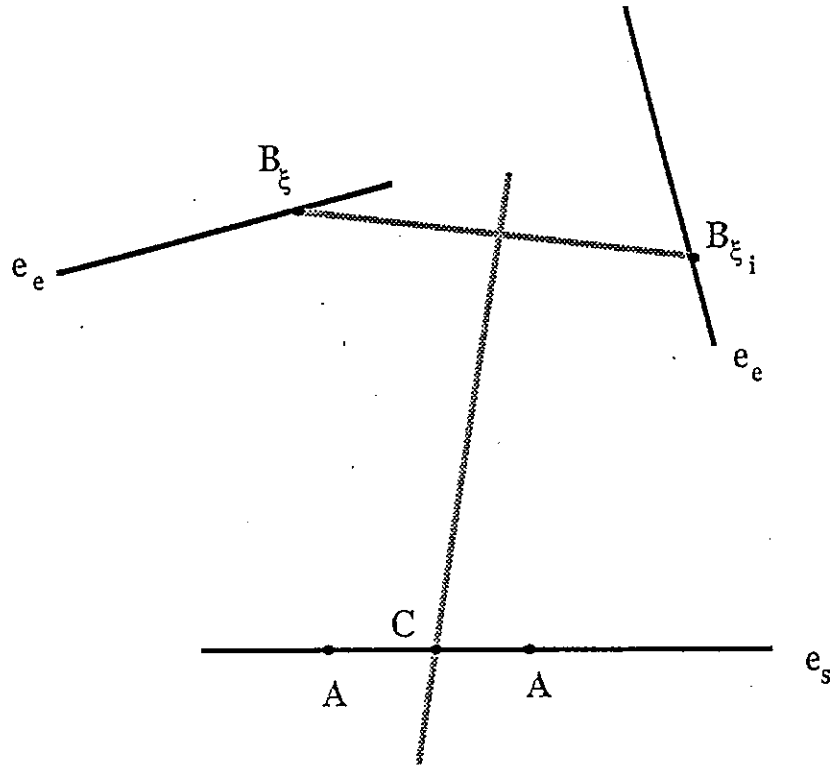

Fig. 2.5 The planar unfoldings relative to ξ and ξ_i

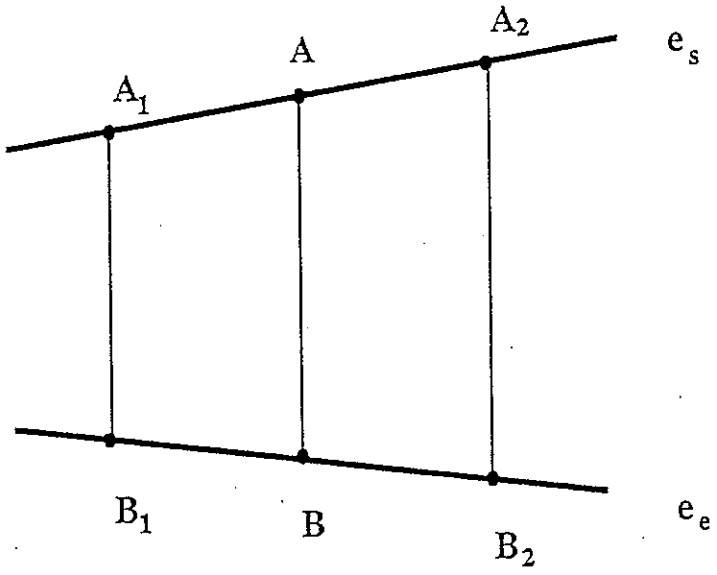

Fig. 2.6 The weight-points located on the boundary of region.

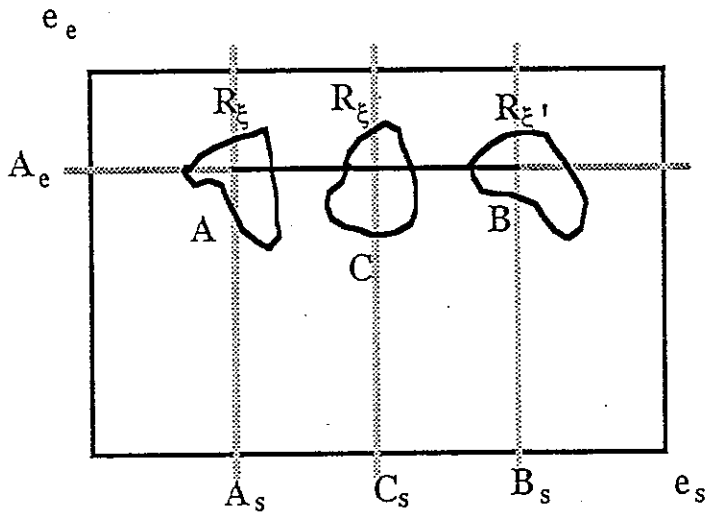

Fig. 2.7a In CASE 1, $A_e = B_e$

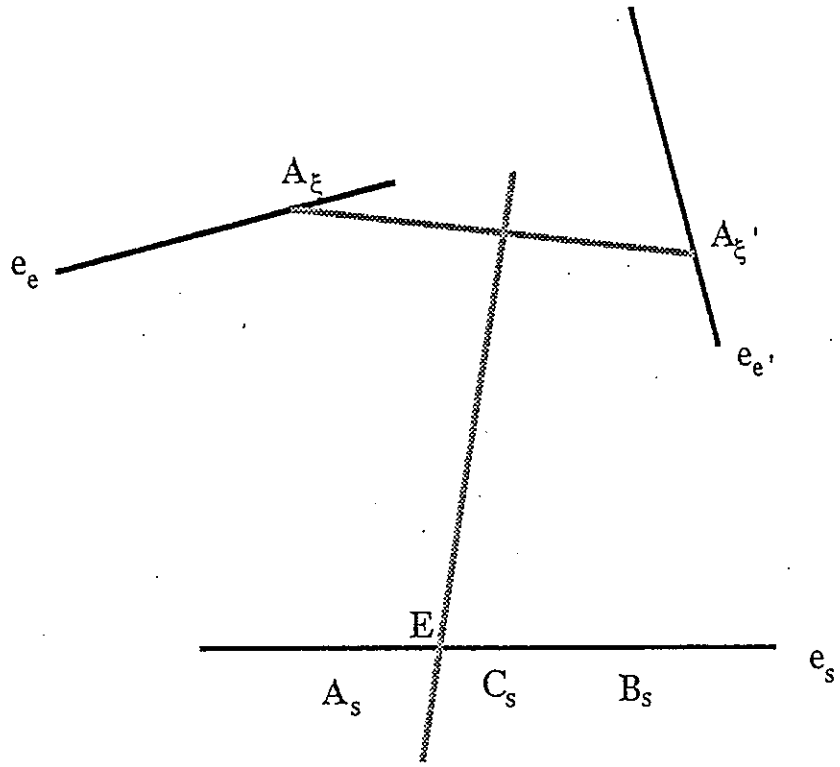

Fig. 2.7b The planar unfoldings of CASE 1.

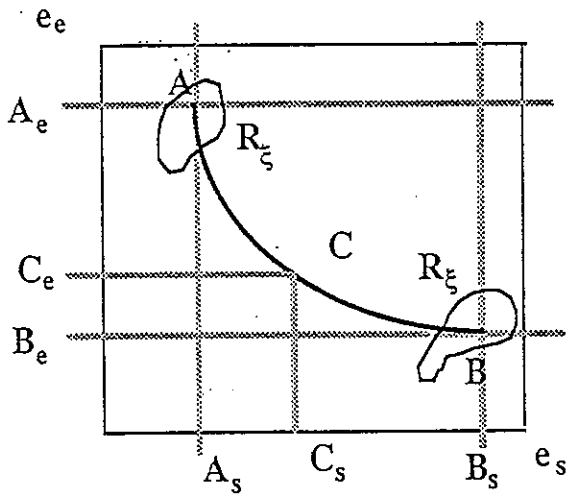

Fig. 2.8a (1) In CASE 2, A_e is not equal to B_e

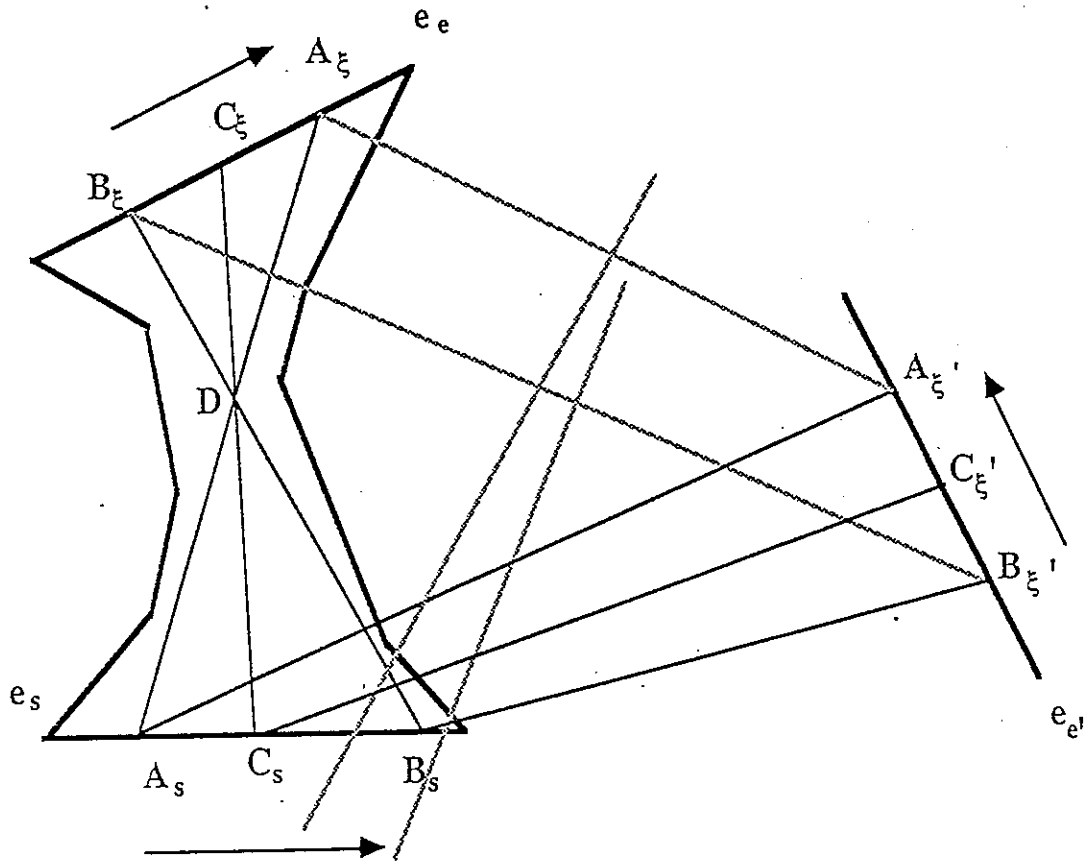

Fig. 2.8a (2) In CASE 2, two shortest paths cross each other.

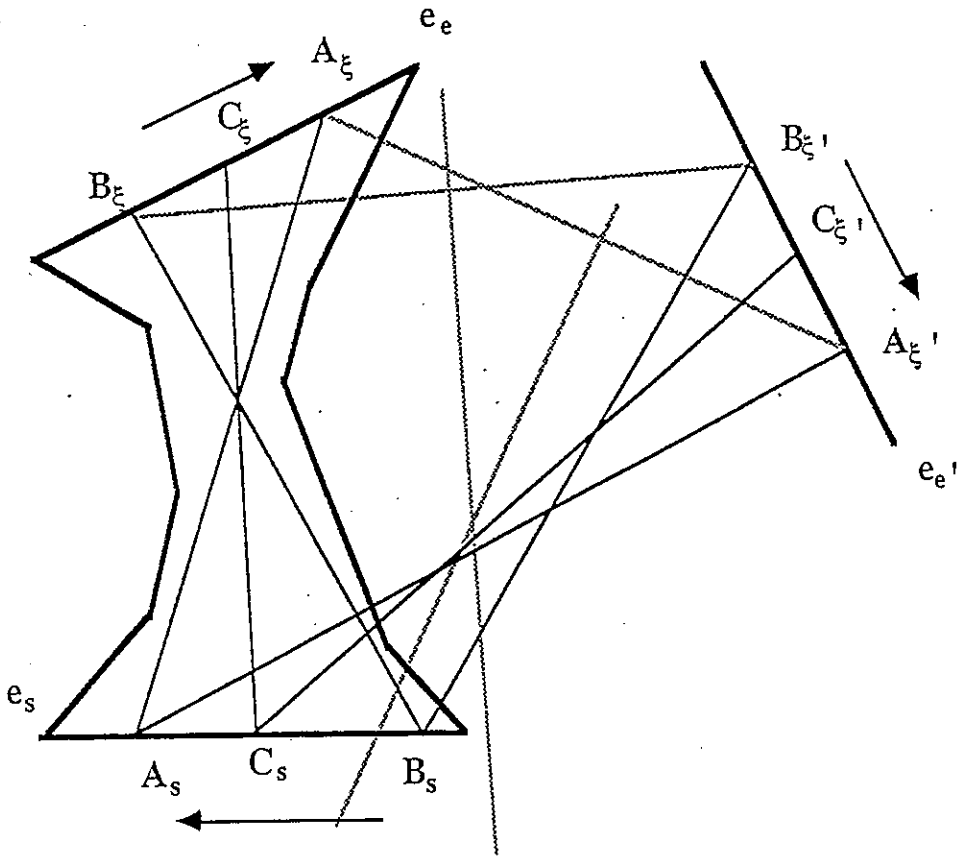

Fig. 2.8a (3) In CASE 2, four shortest paths cross each other in pairs.

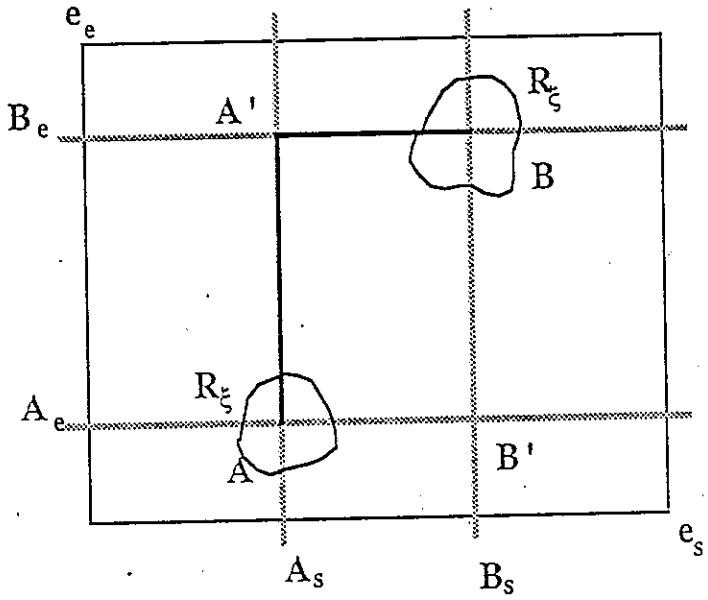

Fig. 2.8b (1) The path connects A and B by crossing A'.

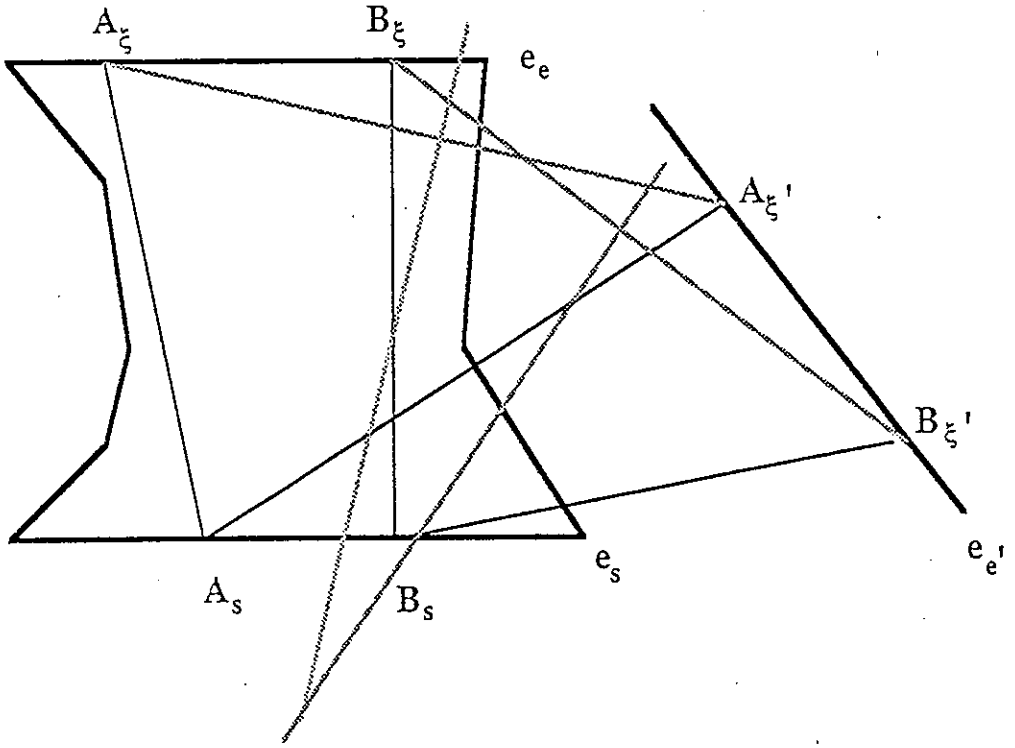

Fig. 2.8b (2) The unfolding relative to Fig. 2.8b (1).

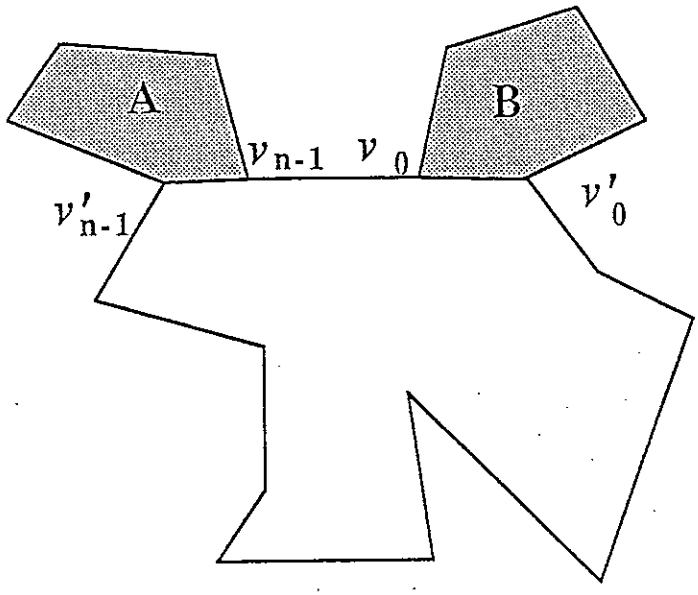

Fig. 2.9a Simple polygon P.

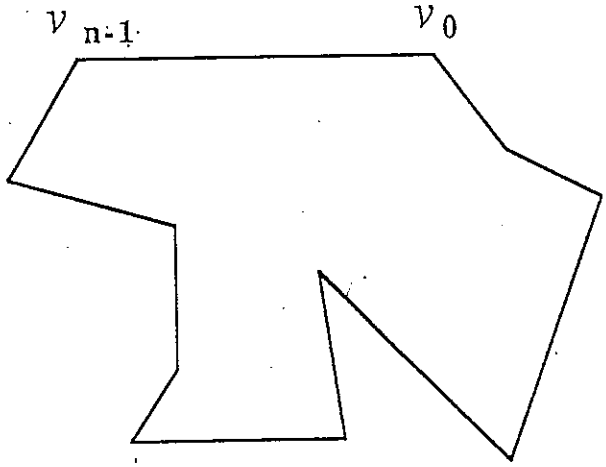

Fig. 2.9b The standard form of P.

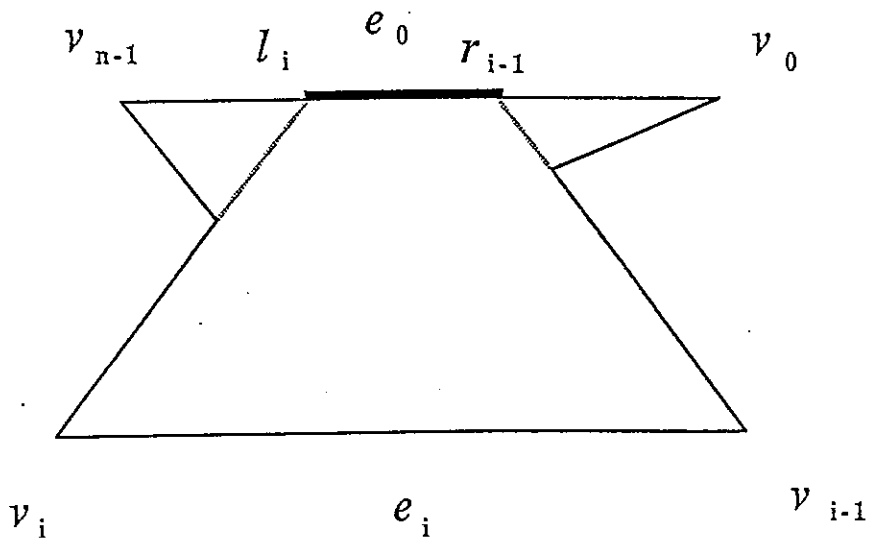

Fig. 2.10a Type 1 segment

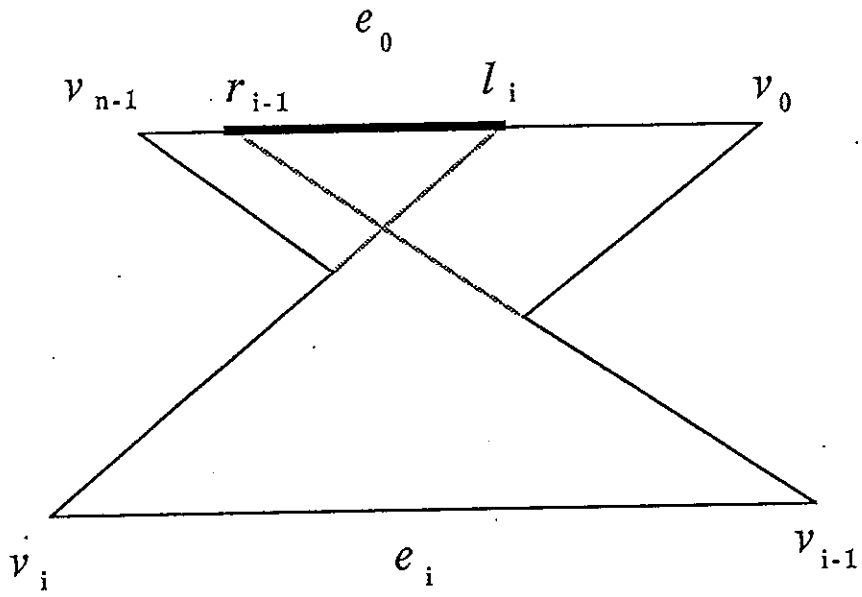

Fig. 2.10b Type 2 segment

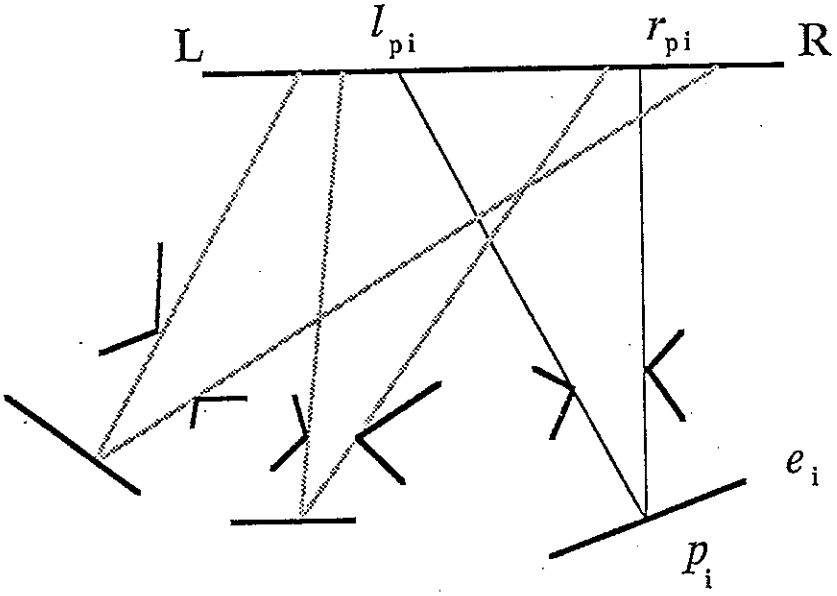

Fig. 2.11 Illustration of l_{pi} and r_{pi}

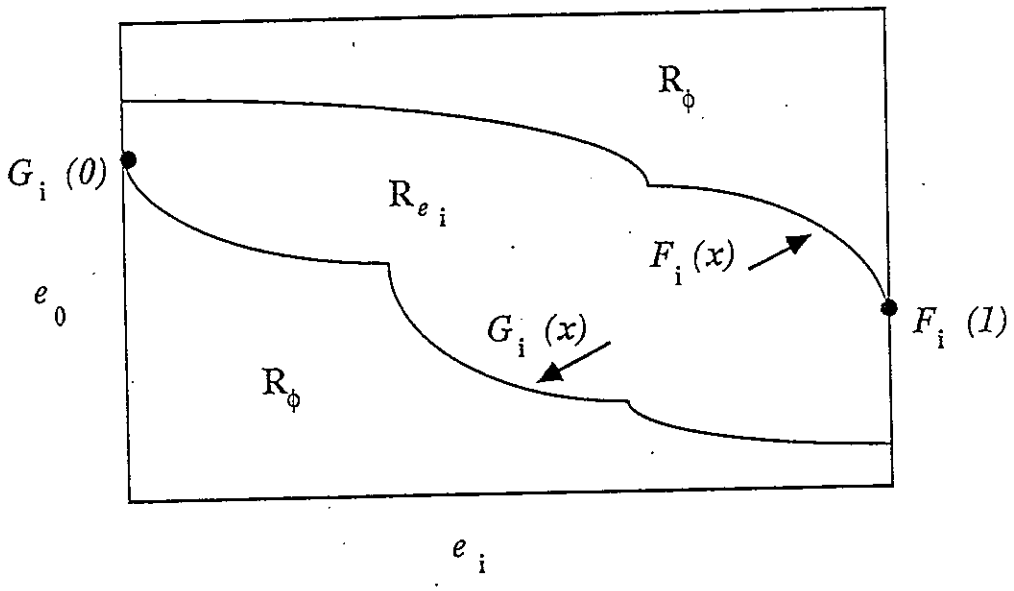

Fig. 2.12 Visbility Relation Diagram of $e_0 X\ e_i$.

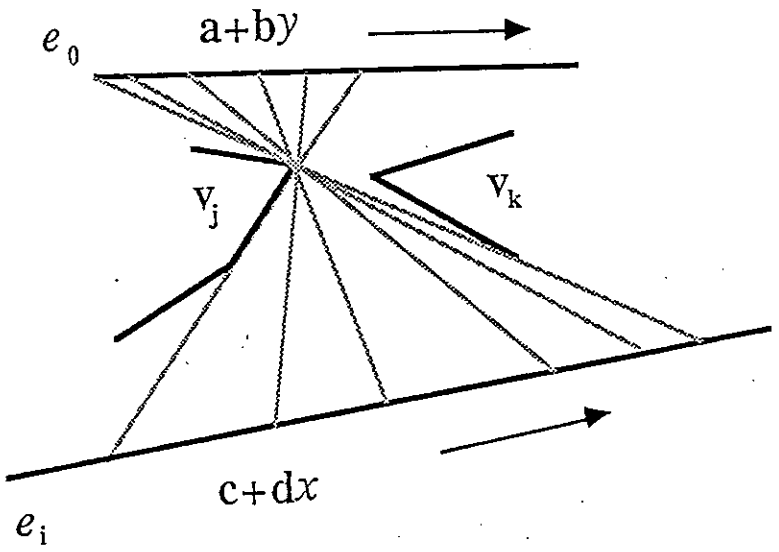

Fig. 2.13a Simple polygon with proper parameters.

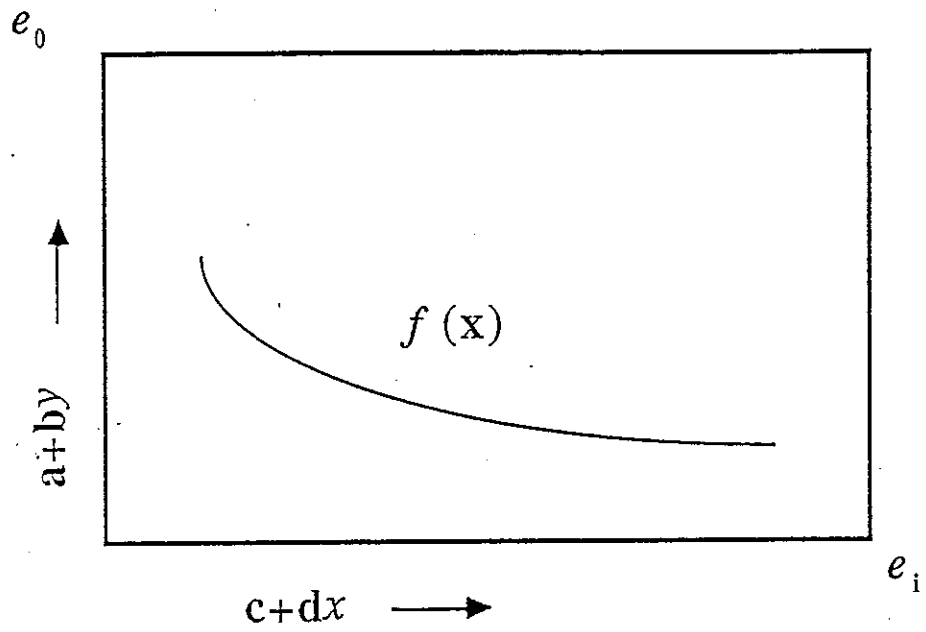

Fig. 2.13b The curve function f(x) of (a) relative to v_j

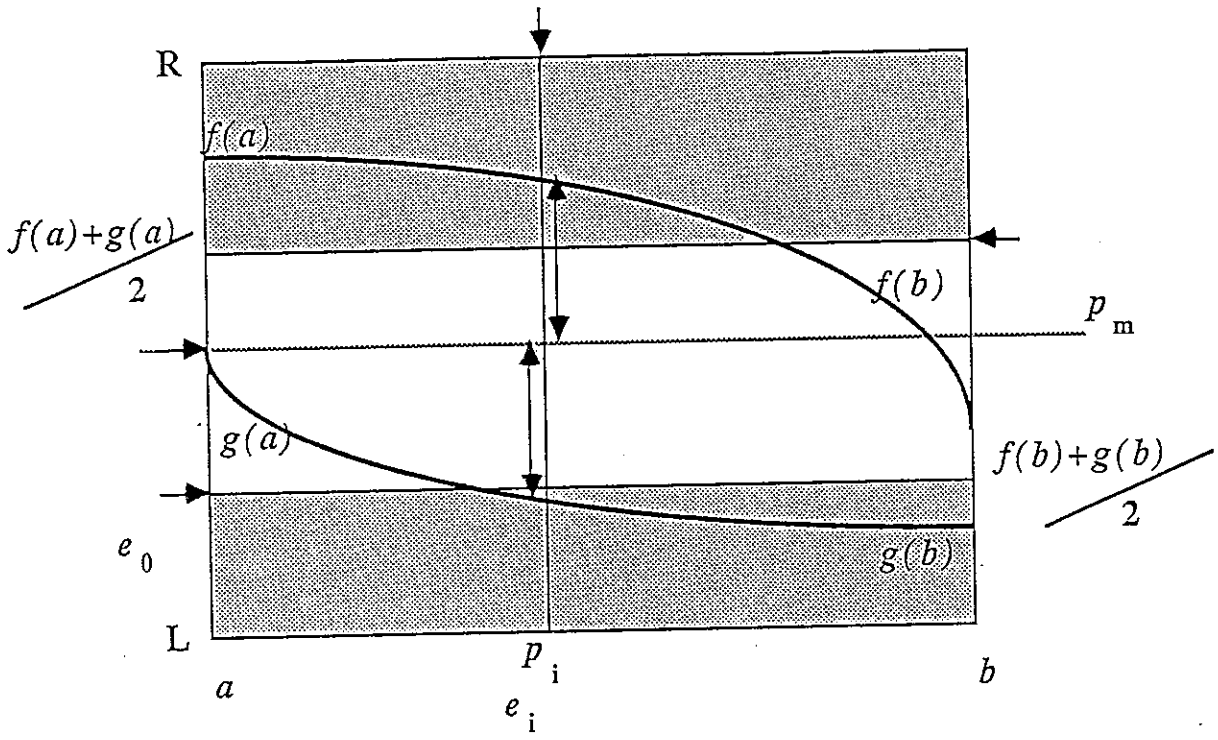

Fig. 2.14 Illustration for the local optimal solution.

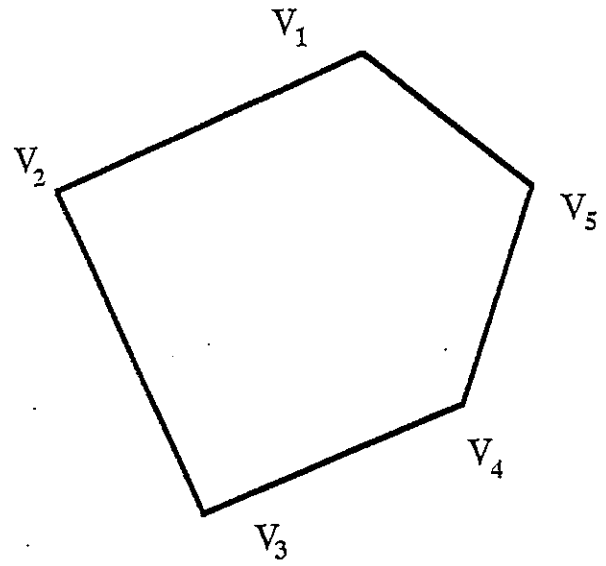

Fig. 3.1. Convex hull CH(B) of set B.

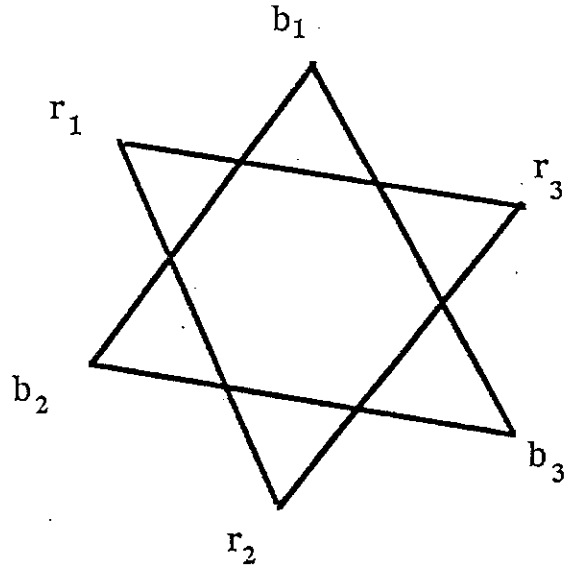

Fig. 3.2. A counterexample of Corollary 2.

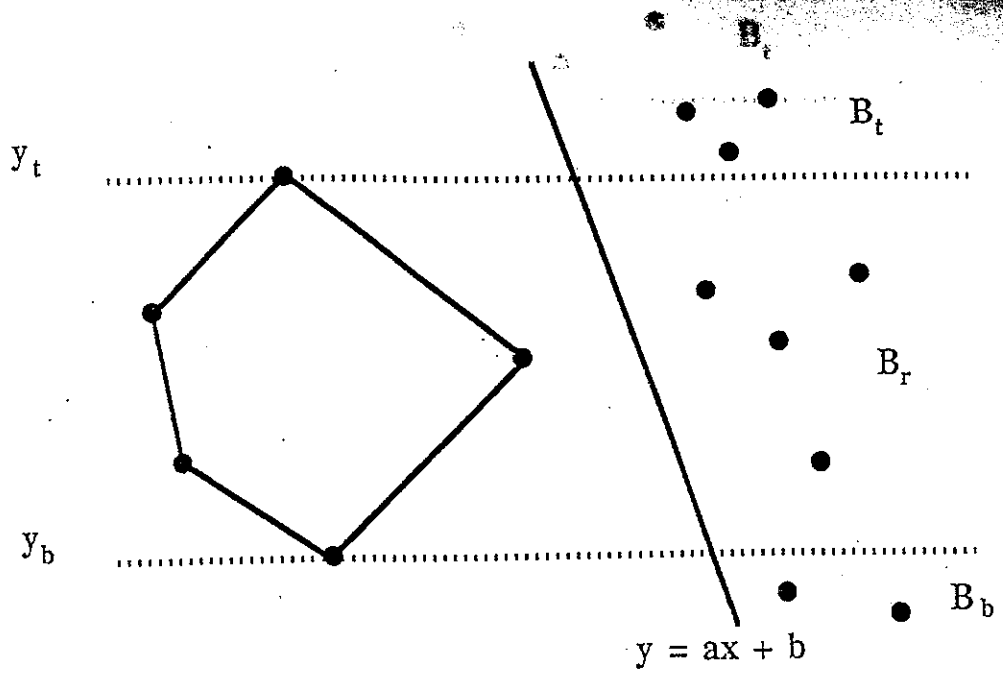

Fig. 3.3. A classification of set B.

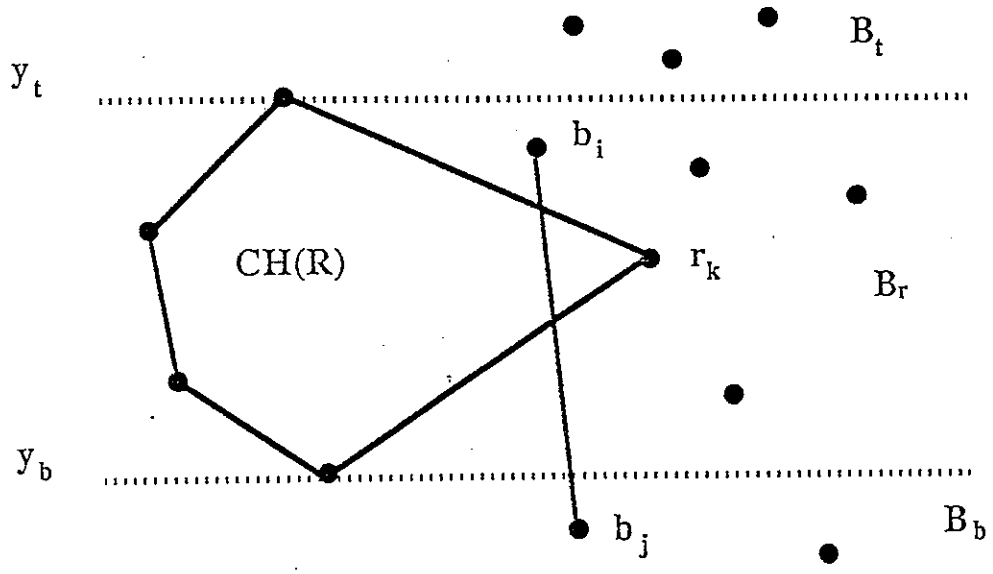

Fig. 3.4. Segment $\overline{b_i b_j}$ intersects CH(R) on two and exactly two points.

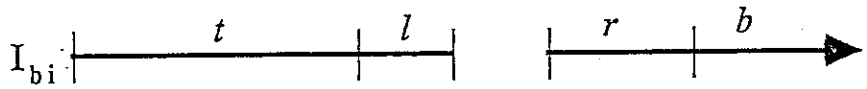

Fig. 3.5. The list of I_{bi}.

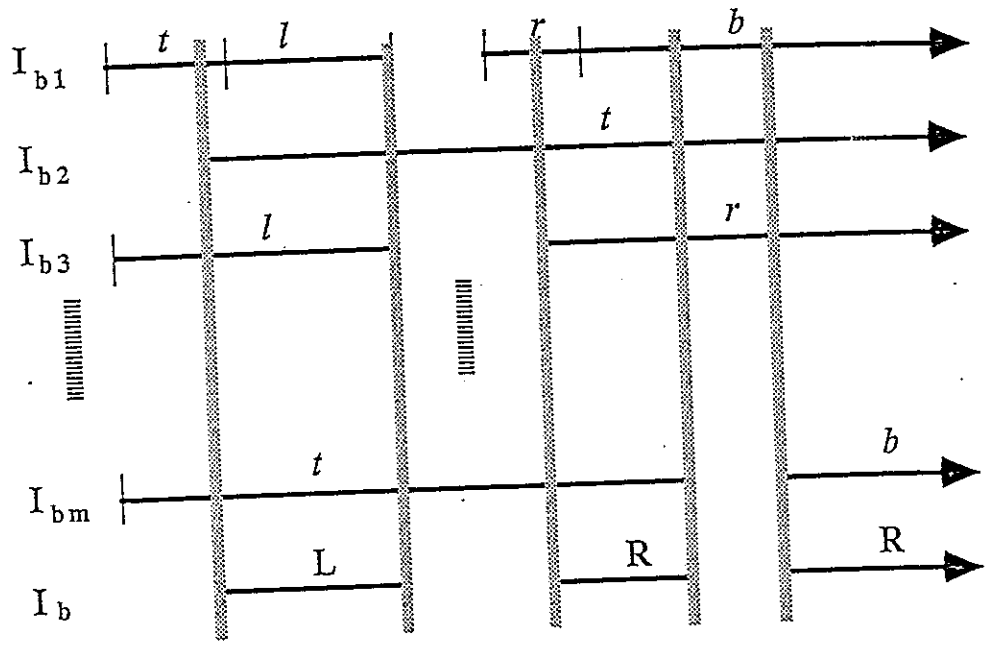

Fig. 3.6. Intersect all I_{bi}'s to be one I_b

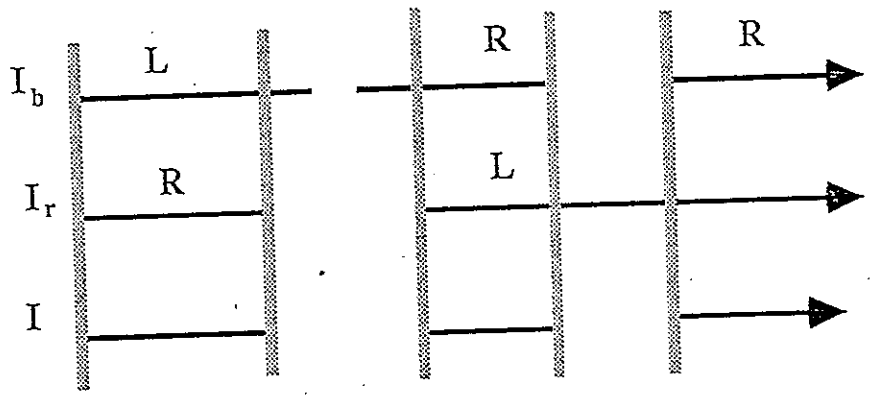

Fig. 3.7. Merge I_b with I_r to I.