A Computer Solution of Polygonal Jigsaw-Puzzles

Ejvind Lynning

DAIMI PB - 9
February 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF AARHUS
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark

University of Århus, Århus, Denmark

Brandeis University, Waltham, Massachusetts U. S. A.

A COMPUTER SOLUTION OF POLYGONAL JIGSAW-PUZZLES

by

Ejvind Lynning

January 1974

University of Århus, Århus, Denmark
Brandeis University, Waltham, Massachusetts U. S. A.

A Computer Solution of Polygonal Jigsaw-Puzzles

by
Ejvind Lynning

January 1974

Author's Address:

Institute of Mathematics, University of Aarhus
Department of Computer Science
Ny Munkegade
8000 Aarhus C
DK-Denmark

Phone: 06-128355

Abstract

This paper describes a program to solve any jigsaw-puzzle involving pieces of polygonal shape. An efficient solution has been found to depend on a number of ad hoc strategies which are described in detail in the paper. The puzzles are solved by successively placing individual pieces in the region to be covered using a depth-first tree search algorithm. A formal representation of regions, pieces, and placings of pieces is defined. The main idea behind the chosen representation is to orient clockwise the polygons making up a region, and to orient counter-clockwise the pieces to be placed. Placing a piece means computing a valid new region, i.e., one or more clockwise oriented polygons, constructed from the previous one by removing the part corresponding to the piece which is placed. The data structure and the procedures required to examine where pieces can be placed and to perform the placings are also described. All the puzzles that have so far been presented to the program have been succesfully solved in a reasonable time.

Key Words and Phrases: Artificial intelligence,problem solving, pattern recognition, puzzles, polygonal puzzles, jigsaw-puzzles, backtrack programming, tree search algorithms.

CR Categories: 3.6, 3.63, 3.64

1. Introduction

Apictorial two-dimensional jigsaw-puzzles can be classified into two types:

(1) those containing pieces of arbitrary shapes, typically involving curved boundaries, and

(2) those in which all pieces are polygons.

By placing points on the perimeters of the pieces of a type-1 puzzle and drawing lines to connect them one can approximate curvilinear boundaries by their polygonal counterparts. Allowing the number of such points to increase, type-1 puzzles can be thought of as limit cases of type-2 puzzles.

A computer solution to puzzles of the first type has been presented by Freeman and Garder [3]. In their work the authors concentrate on classifying and matching curvilinear boundaries, and give less emphasis to the combinatorial aspect of the problem.

An example of a type-2 puzzle that lends itself to computer solution not entailing substantial programming effort is Golomb's polyominoes. The pieces of these puzzles consist of concatenated squares of equal size, such that two juxtaposed squares always share a full edge. Programs to solve these puzzles are not difficult to write because of the nature of the pieces; a grid can be superimposed on the region to be covered, and a simple algorithm will determine whether a given candidate piece can be placed in the region. Although such problems are highly combinatorial, a reduction of the search space may be achieved by mathematical analysis, as Golomb does in [4]. Yet another case of a type-2 puzzle is the one in which the pieces have a particular shape (not necessarily formed by the concatenation of simple geometrical figures). Tangrams belong to this class of puzzles. A program that heuristically attempts to solve this particular kind of puzzle has been described in [1].

If the pieces areallowed to be of any polygonal shape, the programming machinery required to determine where pieces can be placed becomes far more complex. Furthermore, a solution of the general puzzle defies mathematical analysis.

Some of the reasons for selecting polygonal puzzles as a worthwhile programming problem are the following:

(1) Such problems are in general not trivially solved by human beings. As a matter of fact interest in polygonal puzzles

dates back to ancient Greece. One such puzzle credited to Archimedes is said to have been popular among Greeks and Romans*.

(2) The simplicity of the pieces allows emphasis to be given to the combinatorics of fitting pieces in regions, rather than to matching individual boundaries.

(3) This class of puzzles poses a host of challenging problems in choosing a convenient representation for pieces and regions, in selecting a data-structure for this representation, and in devising efficient strategies for obtaining a solution.

(4) Only a computer solution to this type of problem can be envisaged since mathematical analysis is not applicable.

(5) To the knowledge of this author no one has yet presented an efficient solution to this specific type of puzzle.

This paper describes a program to solve any jigsaw-puzzle involving pieces of polygonal shape. As in reference [2], only apictorial puzzles are considered. The program's capabilities are as follows. Consider a number of pieces, such as those in Figure 1. The program is able to assemble them into any polygonal shape in which they fit, such as those in Figure 2**, which also depicts the solutions of the problems. Notice that each piece can occur in any position, rotated or flipped over.

It was found convenient to use the pieces of Figure 1 and the regions they can form as testing cases for the program, because a large repertoire of regions were available using the same pieces.

* The author is thankful to Mr. Martin Gardner of The Scientific American for providing this information.

** The pieces shown in Figure 1 are the ones used in the commercially available puzzle "Ellzup".

2. Representation

Any simple algorithm for computer solution of a jigsaw-puzzle involves the following steps (described according to Floyd's notation for non-deterministic algorithms [2]*):

```
(1) choose an available piece - if none is available the algo-
    rithm has succeeded.
(2) if
      this piece can be placed in the region to be covered
    then
      cover the appropriate area, remove the piece from the
      set of available pieces, goto (1)
    else
      the attempt has failed (backtrack).
```

It is important to notice that placing a piece in a polygonal region may divide the remainder of that region into two or more polygons. Such a splitting may again occur to any of these polygonal areas when more pieces are placed.

In a computerized solution, step (2) will be reiterated a large number of times, and it is therefore of the utmost importance that this step be computed efficiently. This section contains a description of a method for representing pieces and regions that allows one to efficiently examine whether or not a given piece can be placed in a given region.

In the following formalization of the concepts of regions, pieces, and placings of pieces, $\mathbb{R}$ is the set of real numbers, $\mathbb{R}_+$ the set of positive real numbers, and $\mathbb{N}$ the set of natural numbers.

A polygon P_i is a figure consisting of n_i points $p_{i1}, p_{i2}, \ldots, p_{in_i}$, $n_i \geq 3$, $p_{ij} \in \mathbb{R}^2$, and of the line segments $p_{i1}p_{i2}, p_{i2}p_{i3}, \ldots, p_{in_i}p_{i1}$, where $p_{ij}p_{ij+1}$ is to be thought of as belonging to the vector space $\mathbb{R}^2$. Notice that the numbering of vertices assigns an orientation to a path along the perimeter of a polygon; in what follows all polygons are understood to be oriented. It is required that the sides have no common points except their end points.

* The expressions "choose", "has succeeded", and "has failed" correspond to the instructions "choice", "success", and "failure" of Floyd's flowchart language.

A region R is defined to be a finite, indexed set of polygons P_i, such that no two polygons intersect, except possibly at a vertex point. This definition of a region corresponds to the notion of a shape to be filled with puzzle pieces; the shape is allowed to consist of any number of disconnected polygons.

A piece Q is a relocatable polygon, i.e., a polygon which can be rotated and translated freely. A set A of available pieces is a finite set of pieces Q_i. Pieces are assumed to be oriented counter-clockwise while polygons in regions are oriented clockwise.

Let $\mathcal{R}$ be the set of all regions R, and $\mathcal{A}$ be the set of all sets A of available pieces. Placing a piece in a region, thereby producing a new region, is defined by the partial function ρ: $\mathcal{R} \times \mathcal{A} \times \mathbb{N}^4 \to \mathcal{R} \times \mathcal{A}$, which is obtained as follows:

Given $R \in \mathcal{R}$, $A \in \mathcal{A}$ and four integer arguments i, j, k, l, such that (R, A, i, j, k, l) ∈ domain (ρ), first rotate and translate the k'th piece Q_k, so that $q_{k\,l} = p_{i\,j}$, and $q_{k\,l}q_{k\,l-1} = t \cdot p_{i\,j}p_{i\,j+1}$, $t \in \mathbb{R}_+$, i.e., the vectors $q_{k\,l}q_{k\,l-1}$ and $p_{i\,j}p_{i\,j+1}$ have the same orientation. Then consider the pseudo-polygon* $P_i' = p_{i1}p_{i2} \cdots p_{i\,j-1}\, q_{k\,l}q_{k\,l+1} \cdots q_{k\,m_k}\, q_{k1} \cdots q_{k\,l-1}\, p_{i\,j} \cdots p_{i\,n_i}$, given by this sequence of points. Apply to P_i' the operator σ which does the following when applied to a pseudo-polygon $P = p_1 \ldots p_n$.

For each p_i, p_i is deleted if either (a): p_i belongs to any straight line containing both p_{i-1} and p_{i+1}, or (b): $p_i = p_{i-1}$. This process is repeated until neither (a) nor (b) holds for any p_i. In some cases $\sigma(P_i')$ may split into two or more disconnected polygons or vanish into a single point. Finally, substitute $\sigma(P_i')$ for P_i in R and delete Q_k from A to obtain the value ρ(R, A, i, j, k, l).

Figure 3 illustrates an application of the function ρ. Let R be the region in Figure 3a, and let A consist of the pieces in Figure 3b. Then in computing ρ(R, A, 1, 2, 1, 3) one first obtains $P_1' = p_{1\,1}\, q_{1\,3}q_{1\,1}\, q_{1\,2}\, p_{1\,2}p_{1\,3}p_{1\,4}$ seen in Figure 3c. Applying σ, nodes $p_{1\,2}$ and $p_{1\,3}$ are deleted, forming $\sigma(P_1')$ as seen in Figure 3d. Thus ρ(R, A, 1, 2, 1, 3)

*A pseudo-polygon is defined as any sequence of points $p_1 \ldots p_n$, $p_i \in \mathbb{R}^2$ and the line segments p_ip_{i+1}, i=1, .., n−1, and p_np_1 where sides may intersect. This implies that the set of polygons is a proper subset of the set of pseudo-polygons.

results in the region in Figure 3e and the one available piece of Figure 3f (points have been renamed and reindexed).

In the above formalization, the question in step (2) of the sketched algorithm of whether or not a given piece can be placed in a given region is merely the question of whether R, A, and the respective vertex-specifications belong to the domain of ρ or not.

This is determined for any set of arguments (R, A, i, j, k, l) by evaluating $\rho(R, A, i, j, k, l)$ as described above, as if $(R, A, i, j, k, l) \in$ domain(ρ). By examining the result it is possible to determine whether or not $\sigma(P_1')$ is the result of a valid placing.

Two line segments which are not parallel are said to intersect if any point belongs to both of them. The intersection will be called a strong intersection if the point of intersection is not an end point for either of the two line segments; otherwise it will be called a weak intersection. Parallel line segments, in particular segments of the same line, cannot intersect according to this definition.

To establish whether $\sigma(P_1')$ is the result of a valid placing or not, it is first checked whether or not any two sides of this pseudo-polygon, not immediately consecutive, intersect - weakly or strongly. If not, the pseudo-polygon is a single polygon, and a valid placing has been made. Otherwise, if only weak intersections occur, the pseudo-polygon is split into disjoint sub-polygons at the intersection points, and the operator σ is applied to each of these sub-polygons. If the sub-polygons have a correct orientation the placing is valid, otherwise not. If strong intersections occur in the pseudo-polygon, the placing is not valid. It is possible for the pseudo-polygon or one of the possible sub-polygons to be the nil-polygon (one consisting of a single point). Such nil-polygons do not effect the validity of a placing. The different types of valid and invalid placings are illustrated in Figures 3 and 4. Each of the situations in Figure 4 shows an initial configuration consisting of a polygon and a piece to be placed. The vertices where the placing is to be done are indicated by arrows. Then each subfigure shows the pseudo-polygon resulting from applying ρ and also, if applicable, the result of splitting this pseudo-polygon into sub-polygons and applying σ to these. Figure 3 shows the case of no intersections - weak or strong. Figure 4a shows the case of a strong intersection. In Figure 4b two weak intersections occur, splitting the pseudo-polygon into three sub-polygons, of which one is seen to be the nil-polygon. In Figure 4c one weak intersection occurs, splitting the pseudo-polygon into two sub-polygons, of which one has the wrong orientation. Figures 3 and 4b represent valid placings; Figures 4a and 4c do not. The reader not convinced by these examples is referred to the appendix in which an outline of a formal proof of the sufficiency of this test is given.

The above approach to answering the question in step (2) of the algorithm outlined in the beginning of this section provides a convenient framework in which to program a computer solution.

3. Data Structure

The data structure used to represent in the computer memory the regions and pieces of the previous section and to facilitate computation of the function ρ can be conveniently described using records [5].

A polygon, a pseudo-polygon or a piece is represented by a circular list of records (one record per vertex) having the following structure:

	Field	Type
(1)	x-coordinate	real
(2)	y-coordinate	real
(3)	length of issuing side	real
(4)	angle of the side, i.e., angle from x-axis to the orientation of the side	real
(5)	forward pointer to the next record in the list	reference

In the case of puzzle-pieces the coordinates x and y are initially irrelevant, and are assigned values only when a piece is used by the ρ-function.

A region R is represented as a linear list of records, each record corresponding to a polygon or pseudo-polygon. These records contain the following information:

	Field	Type
(1)	pointer to the list representing a polygon or a pseudo-polygon	reference
(2)	area of the polygon*	real
(3)	forward pointer to the next record in the list, or nil if none exists	reference

A set A of available pieces is represented as a list of records,

* This quantity as well as the area of pieces is utilized in the strategies described in section 6.

one record for each different piece:

	Field	Type
(1)	pointer to the list representing the piece	reference
(2)	pointer to the list representing the mirror image of the piece	reference
(3)	area of the piece	real
(4)	forward pointer to the next record in the list or nil if none exists	reference

In the case of symmetric pieces (1) and (2) are identical.

Figures 5b and 5c show the records representing the region and the set of available pieces appearing in Figures 3a and 3b. Figure 5a illustrates the meaning of the contents of the individual fields of the records.

4. Basic procedures

This section contains a description of a set of procedures that perform essential functions on the list structured data representing regions and pieces.

The procedures described below are those which examine or change the contents of the records. Others that are used to construct and administer the list structure, but are not inherent to the problem of solving polygonal puzzles, are not described here.

ANGLE CHECK. Its parameters are (1) polygon, (2) piece, (3) vertex, and (4) piece vertex. The first two parameters are pointers to the lists representing a polygon in a region and a piece, respectively; the last two parameters are integers specifying vertices of the polygon and the piece. ANGLE CHECK is a procedure of the boolean type which examines whether the interior angle of a given piece vertex is or is not too large to fit within the interior angle at the given vertex of a polygon.

PLACE. Its parameters are the same as those of ANGLE CHECK. PLACE performs the first part of the ρ-function; i.e., it aligns the piece to be placed by rotating it and forms a list representing the pseudo-polygon described in section 2. PLACE yields a pointer to this list as its result.

S. Its parameter is pseudo-polygon, which is a pointer to the list representing a pseudo-polygon. S represents the σ-operator of section 2. It does the following to the list pointed to by its argument:

(1) if the length field of any of its records contains a zero this record is deleted.

(2) if the angle fields of two successive records are equal modulo π or modulo 2π, these records are collapsed into one containing the proper length and angle fields.

This process is repeated until neither of the two conditions hold, and finally the coordinate fields are adjusted. The result of S is a pointer to this modified list.

LEGAL PSEUDO-POLYGON. Its parameters are (1) pseudo-polygon, and (2) sub-region. The first parameter is a pointer to the list representing a pseudo-polygon, and the second is a pointer to the list representing a region. LEGAL PSEUDO-POLYGON is a boolean procedure, which determines whether the sequence of calls to the procedures PLACE and S has resulted in a valid placing or not. This question is the crucial one of section 2, and this procedure is the nucleus of the algorithm presented in the following section. The procedure works as described in section 2, calling the auxiliary procedures INTERSECT which examines whether or not two line segments intersect, and SPLIT which splits a pseudo-polygon at weak intersection points. If a valid placing has been made, a list representing the one or more polygons resulting from it is constructed, and a pointer to it is assigned to the output-parameter sub-region; if not sub-region is given the value nil.

SUBSTITUTE. Its parameters are (1) R, (2) sub-region, and (3) polygon. The first two parameters are pointers to lists representing regions; the last parameter is a pointer to the list representing a polygon. This function yields a pointer to a copy of the list pointed to by R, in which the record containing the pointer polygon is replaced by the list pointed to by sub-region.

DELETE. Its parameters are (1) A, and (2) piece. The first parameter is a pointer to a list of available pieces, the second is a pointer to the list representing a piece. This function yields a pointer to a copy of the list pointed to by A with the record containing the pointer piece omitted.

MIRROR IMAGE. Its parameters are the same as those of DELETE. This boolean procedure yields the value true if the pointer piece occurs in the first field of a record in the list pointed to by A, and the second field of this record is different from piece. In this case also the value of piece becomes that of the second field. Otherwise the result of MIRROR IMAGE is false.

5. Main Algorithm

Consider a region R and a set A of available pieces; assuming that the pieces can be assembled to fill the region, there is, for any $P_i \in R$ and any vertex p_{ij} of P_i, having an interior angle less than π, some piece $Q_k \in A$ and a vertex q_{kl} of Q_k, such that the value $\rho(R, A, i, j, k, l)$ is defined; i.e., the piece Q_k can be placed so that q_{kl} covers p_{ij} and so that the side $q_{kl}q_{kl-1}$ covers part or all of $p_{ij}p_{ij+1}$.* Since any polygon has at least one vertex at which the interior angle is less than π, one may choose a polygon in the region concerned, and a vertex of that polygon having such an angle, whenever a placing of a piece is sought. If indeed the region can be assembled from the pieces available, some piece can be placed there. This approach reduces the size of the search tree that must be explored and leads to the algorithm expressed in the following procedure, written in Algol 60.

The algorithm searches through a tree of depth equal to the number of pieces in the puzzle. Nodes in the search tree are the pairs (R, A) that result from the placings of pieces performed by the algorithm. The number of branches emanating from each node is limited by the total number of vertices of the puzzle pieces, the vertices of non-symmetric pieces counted twice.

The procedures "not empty", "first available piece", "number of vertices", and "next available piece" are self-explanatory from the description of the data-structure.

* The reader is urged to check, by examples, the validity of this statement.

```
boolean procedure solve (R, A);
value R, A; integer R, A;
begin integer polygon, piece; integer vertex, piece vertex;
      comment R, A, polygon, and piece are pointers to list-structured
              data;
      solve:=true;
      if not empty (A) then
      begin L: ; comment this part will later be replaced by statements
                 which assign values to the variables polygon and
                 vertex;
         choose a piece:
         for piece:=first available piece (A), next available piece
         (A, piece) while piece ≠ nil do
         begin choose vertex of piece or: its mirror image:
            for piece vertex:=1 step 1 until number of vertices
            (piece) do
            if anglecheck (polygon, piece, vertex, piece vertex)
            then
            begin pseudo-polygon:=s (place (polygon, piece, vertex,
               piece vertex));
               if legal pseudo-polygon (pseudo-polygon, sub-
               region) then
               begin if solve (substitute (R, polygon, sub-region),
                  delete(A, piece)) then goto exit end
            end of loop for choosing a vertex;
         if mirror image (A, piece) then goto its mirror image
         end of loop for choosing a piece;
         backtrack:solve:=false
      end of A not empty;
exit:
end of solve;
```

The program to solve a puzzle consists merely of a call to this procedure with initial values of R and A supplied as parameters. If a puzzle has more than one solution the program can be commanded to find them all by simulating a failure just after a solution is found.

6. Strategy

The search tree that must be explored by any algorithm solving polygonal puzzles by placing one piece at atime can become very large. If a puzzle consistsof n different, non-symetric pieces each having m vertices, a bound on the total number of nodes in the tree is given by $N = \sum_{i=0}^{n} 2^i m^i \frac{n}{(n-i)!}$; for n=10, m=4, $N \approx 4.4 \cdot 10^{15}$. The configuration (R, A) corresponding to a node in such a tree requires a considerable amount of information to be stored (50 - 150 words, depending on the number of polygons in R, the number of vertices of these polygons, and the number of different pieces available). Therefore, any practical algorithm to explore such a tree should be of the depth-first type, as the one described in the previous section.

An attempt to enhance efficiency by ordering the nodes of the tree according to some notion of "cost" [6] can involve only the ordering of the immediate successor-nodes of each node explored. By inspection of a large number of configurations corresponding to nodes in the trees explored by the program, no very helpful criterion has been identified which distinguishes nodes on the path to a solution from the "bad" ones (i.e., pairs (R, A) so that R cannot be assembled from A), obtained from the same immediate parent node. It has been found helpful to arrange the pieces in the set A according to their size. Larger pieces are more difficult to place than smaller ones. Therefore, an attempt to first place the larger pieces provides a way to avoid unnecessary evaluations of subtrees resulting from erroneous placings of small pieces.

A different type of strategy which has proved valuable in efforts to improve the algorithm is one that attempts to limit the number of nodes that can possibly be reached in the search tree. The most important such strategy is described in terms of the following procedures.

MINIMAL AREA. Its parameter is R, which is a pointer to the list representing a region. This function computes the area of each polygon in the region, and its result is a pointer to the list representing the smallest of them. The smallest polygon in a region is always chosen for the placing of the next piece. Since the number of possible placings in this polygon is small, a quick recovery is more likely in case a backtrack is needed.

BEST VERTEX. Its parameter is polygon, which is a pointer to the list representing a polygon. This function assigns a value to each vertex of a polygon based on the length of the sides having that vertex as an end point, and on the interior angle of that vertex: high priority is given to vertices having small adjacent sides and small interior angles, since these features make it likely that only few placings can be performed at the chosen vertex, thus limiting the number of successor-nodes in the search tree. The result of this function is the number of the vertex having the highest value. Vertices having an interior angle larger than π cannot be chosen (see section 5).

In order to incorporate these strategies into the algorithm of section 5, calls to these procedures are inserted at the label L.

L:polygon:=minimal area (R); vertex:=best vertex (polygon);

The reason for allowing a region to consist of several polygons should now be apparent. If one were to always place a piece in such a way that the region to be covered would not be split into two or more sub-regions, the vertices of the region would have to be exhaustively matched with the vertices of each piece. This exhaustive matching is inefficient compared to the method adopted here which entails matching only the "best vertex" in order to force inevitable backtrackings to take place as early as possible.

So far, the only criterion for a backtrack has been that of a node having no successor nodes generated by placing pieces. A further reduction in the size of the search tree to be explored can be gained by noting some intrinsic characteristics of regions and their corresponding sets of available pieces that render an assembly impossible. If a node encountered possesses one of these characteristics, an immediate backtrack is forced. Thus an evaluation of the subtree of this node is avoided. Such characteristics include:

(1) The region contains a polygon smaller than the smallest available piece.

(2) A polygon in the region contains a side shorter than the shortest side occuring in an available piece, and the in-

terior angle at the end points of this side are both less than π. This is illustrated in Figure 6. The sides marked s are assumed to be shorter than any side occuring in an available piece. Figure 6a shows the case described above, where no piece can be placed along side s; in the case of Figure 6b where one of the angles is larger than π, one may well be able to place a piece.

(3) The region is identical to a region that has already been explored by some other path in the tree, having the same set of available pieces, and found not possible to assemble. In order to recognize this situation a list of "bad" regions and the corresponding sets of available pieces is required. To create such a list efficiently a unique hashing scheme is needed which can identify a region by a few real numbers. In the general case of polygonal puzzles such a unique scheme may prove very difficult to construct.

7. Final Remarks

Any list-processing language could have been used to implement the algorithm described in section 5. However, since Algol-compilers were available on the computers to which the author had access (GIER and PDP-10), and because of the author's familiarity with Algol-programming, it was decided to write the program in Algol, implementing the necessary record-handling by auxiliary procedures.

All the strategies described in the previous section have been implemented in the program* except the one entailing the ordering of the successor nodes for each node explored. The evidence collected through solving many puzzles indicates that the algorithm presented here is amenable only to marginal improvement through the implementation of such a strategy.

A large number of the problems entailing the pieces of Figure 1 have been solved. The number of nodes evaluated has varied from the minimum of 10 (the number of pieces) to 230, the typical number being in the range from 25 to 50. The time for solving the problem has varied accordingly from 4 seconds to 2-1/2 minutes (PDP-10 CPU-time). Two of the (partial) search trees that were explored in finding solutions of two problems are shown in Figures 7 and 8. In these figures dots indicate pieces that have been placed.

Acknowledgements

I whish to express my sincere gratitude to Dr. Jacques Cohen of Brandeis University for innumerable helpful suggestions, as well as for making the necessary facilities available to me and assisting with many practical details.

*The hashing scheme used to recognize characteristic (3) is only applicable to regions derived from the pieces of Figure 1. Characteristics(1) and (2) are recognized in the general case of polygonal regions.

References

[1] Deutsch, E. S. and Hayes, K. C. "A Heuristic Solution to the Tangram Puzzle.", Machine Intelligence 7, (eds. Melzer, B. and Michie, D.), 1972, 205-240.

[2] Floyd, R. W. "Nondeterministic algorithms," J. ACM 14 (Oct. 1967) 636-644.

[3] Freeman, H. and Garder, L. "Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition," IEEE Transactions on Electronic Computer EC-13 (Apr. 1964) 118-127.

[4] Golomb, S. W. _Polyominoes,_ Charles Scribner's Sons, New York, 1965.

[5] Hoare, C. A. R. "Record Handling," in _Programming Languages,_ Academic Press, New York, 1968, 291-347.

[6] Nilsson, N. J. _Problem Solving Methods in Artificial Intelligence_, McGraw-Hill, New York, 1971, Chap. 3.

Appendix

Oriented polygons and pseudo-polygons, weak and strong intersections, and the operator σ have been defined in section 2.

Theorem: Let P be a clockwise oriented polygon $p_1 p_2 \ldots p_n$, and Q a counter-clockwise oriented polygon (a piece) $q_1 q_2 \ldots q_m$. Let p_i be a vertex of P at which the interior angle is less than π and q_k a vertex of Q, so that the interior angle at q_k is smaller than or equal to that of p_i. Then if Q cannot be placed inside P, so that q_k covers p_i, and the side $q_k q_{k-1}$ covers part or all of the side $p_i p_{i+1}$, the pseudo-polygon $P' = \sigma(p_1 p_2 \ldots p_{i-1} q_k q_{k+1} \ldots q_m q_1 \ldots q_{k-1} p_i \ldots p_n)$ will either contain two sides that intersect strongly, or it can be split at weak intersection points into polygons, of which at least one has a counter-clockwise orientation.

Sketch of proof: Notice that the two sides of Q which have q_k as an end point must stay inside the angle between the two sides of P having p_i as an end point. Therefore, there are essentially three cases of placings or attempted placings:

(1) the one in which Q does not reach across P to any point of a side opposite to p_i.

(2) the one in which Q touches some side(s) across from p_i, thereby cutting the remainder of P in two.

(3) the case in which Q extends beyond the boundaries of P. This case can be subdivided into:

  (a) a side of Q intersects strongly with a side of P.

  (b) the point(s) where the boundary of Q penetrates the boundary of P are only weak intersection points.

These cases are illustrated in Figure 9.

It is clear that what must be proved is that in case 3b the pseudo-polygon P' can indeed be split into polygons of which at least one has the counter-clockwise orientation. The proof can be derived from the following two observations:

(1) after application of σ, at least one weak intersection point must remain, otherwise Q does not penetrate P. At this intersection point a splitting can be performed.

(2) after such a splitting has been made the part of Q extending beyond the boundaries of P will be represented as one or more counter-clockwise oriented polygons.

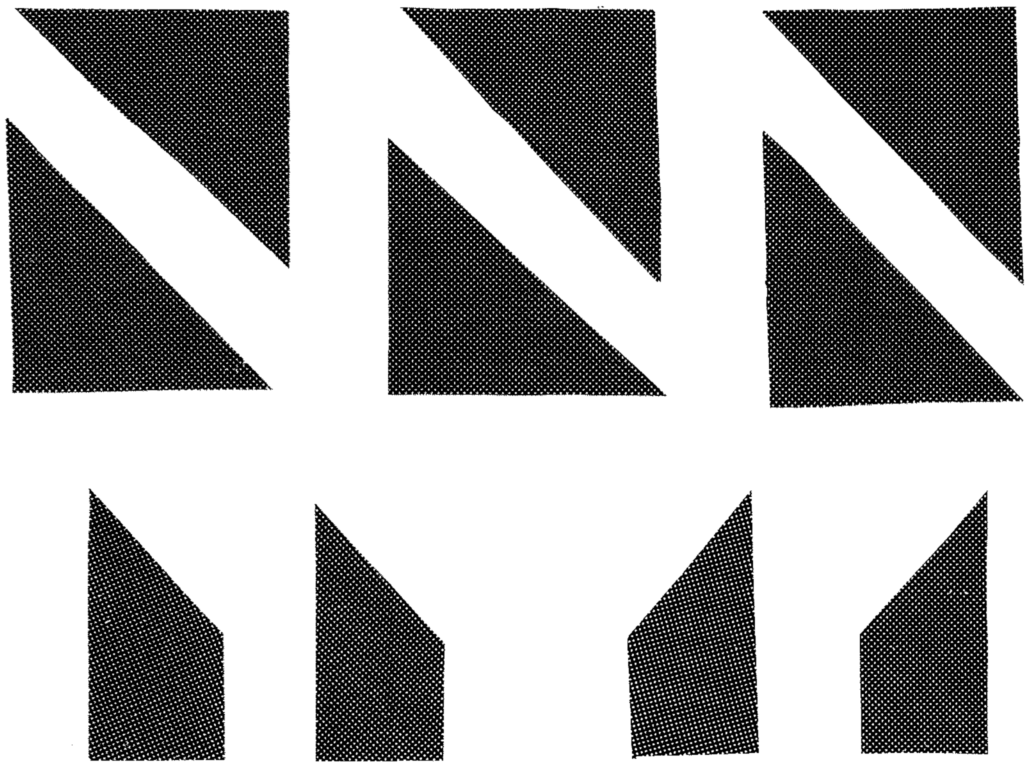


FIGURE 1

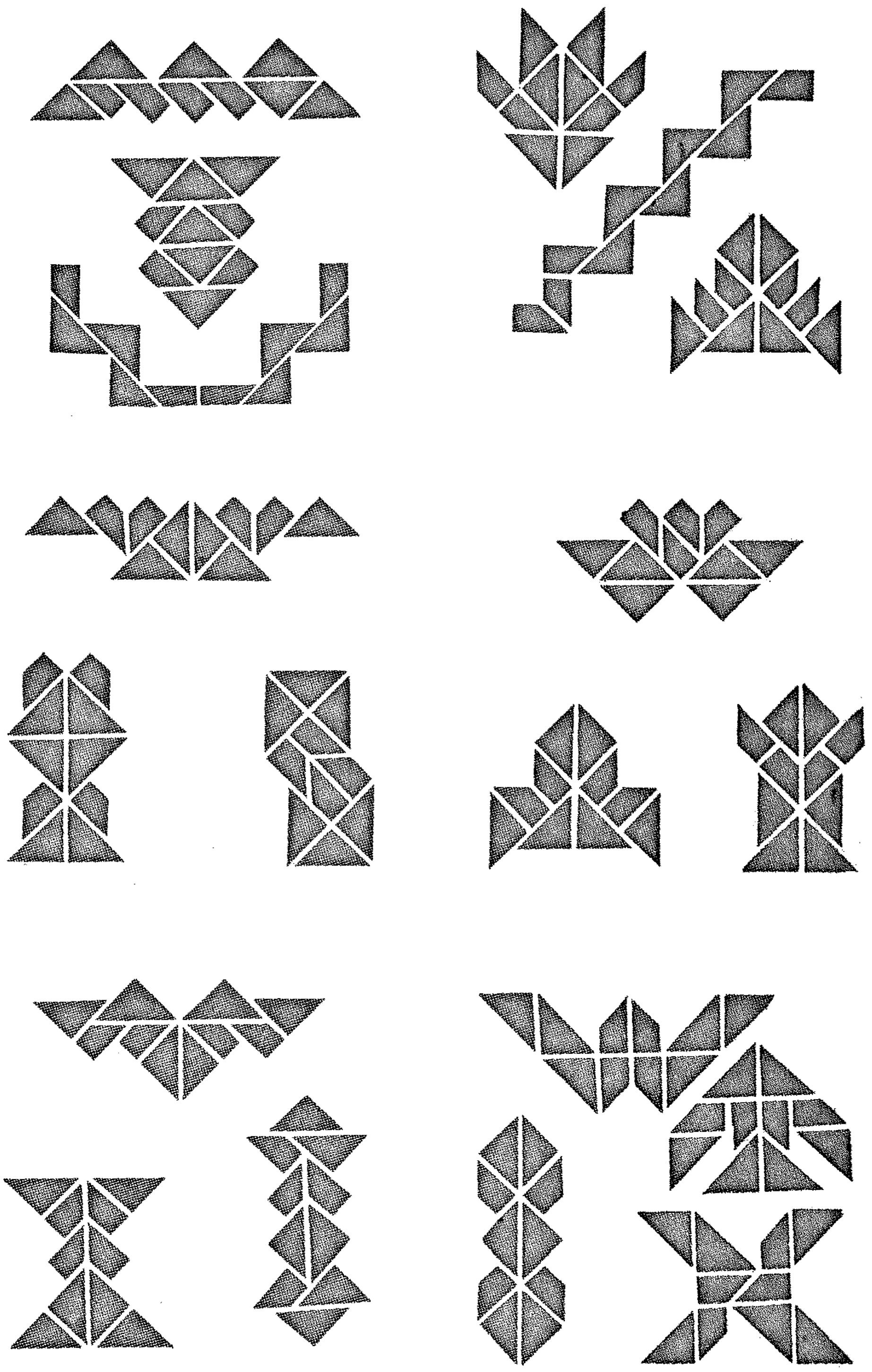
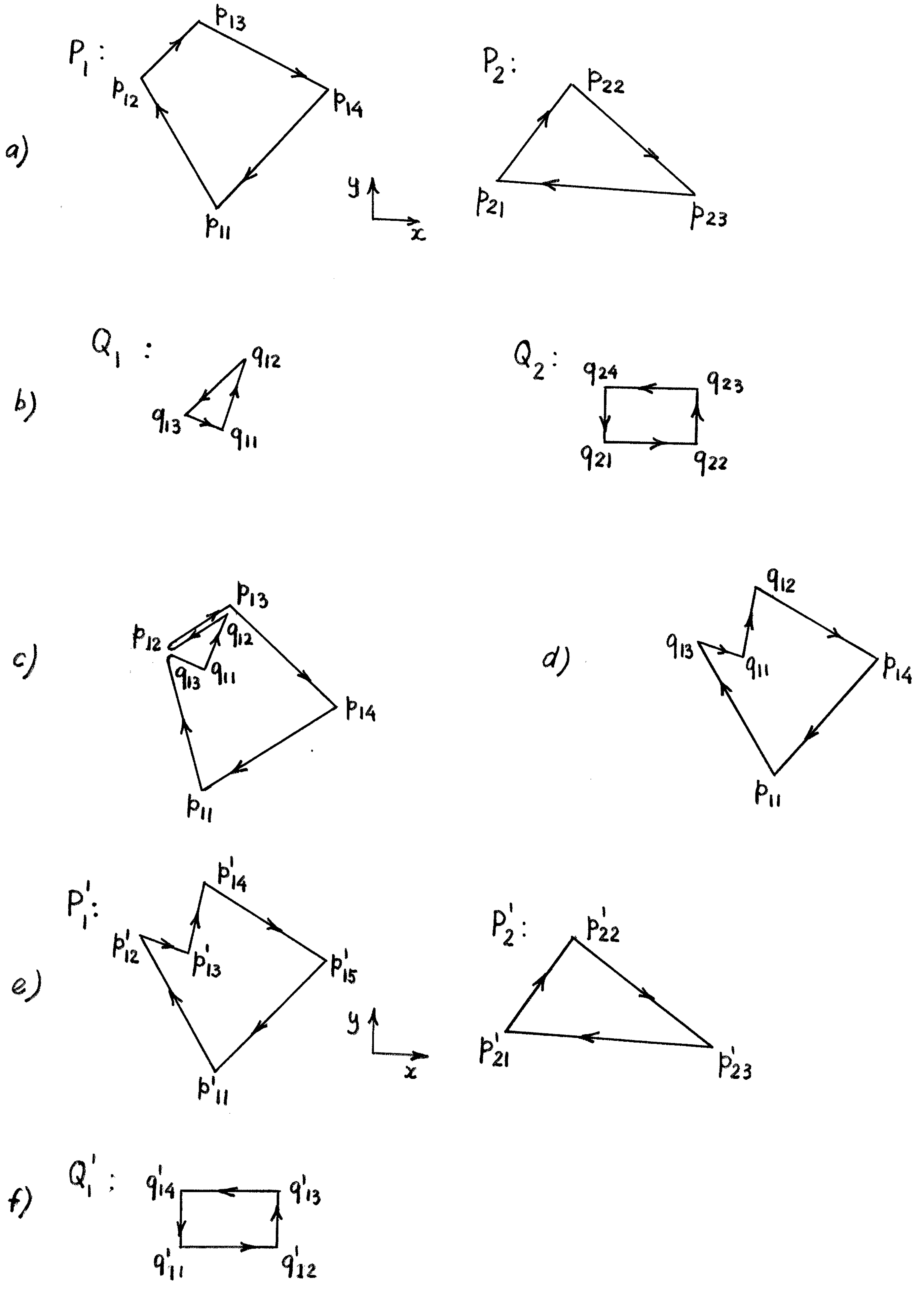


FIGURE 2



a)
P_1 :
p_{13}
p_{12}
p_{14}
p_{11}
y
x
P_2 :
p_{22}
p_{21}
p_{23}
b)
Q_1 :
q_{12}
q_{13}
q_{11}
Q_2 :
q_{24}
q_{23}
q_{21}
q_{22}
c)
p_{13}
p_{12}
q_{12}
q_{13}
q_{11}
p_{14}
p_{11}
d)
q_{12}
q_{13}
q_{11}
p_{14}
p_{11}
e)
P_1':
p_{14}'
p_{12}'
p_{13}'
p_{15}'
p_{11}'
y
x
P_2' :
p_{22}'
p_{21}'
p_{23}'
f)
Q_1' ;
q_{14}'
q_{13}'
q_{11}'
q_{12}'

FIGURE 3

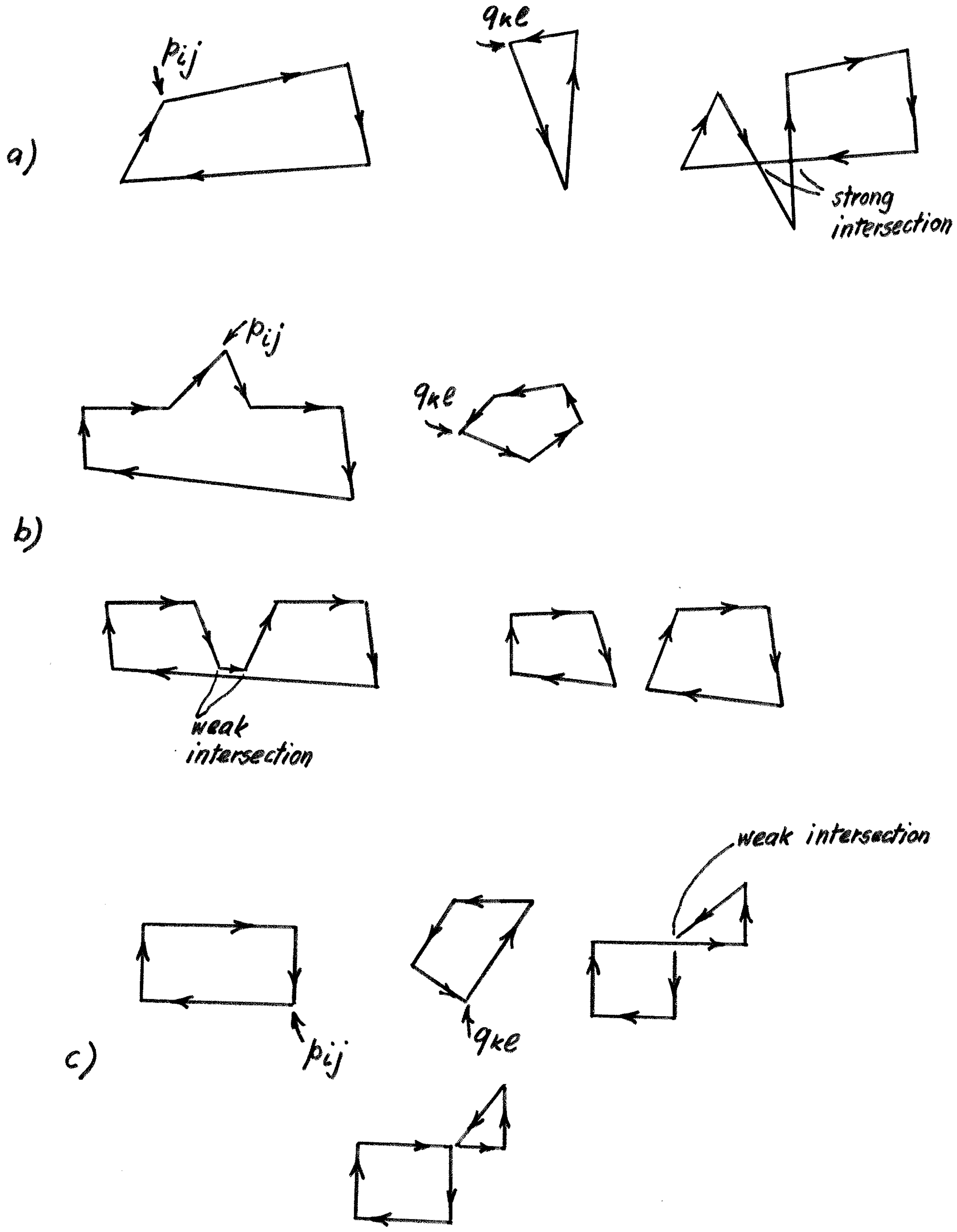
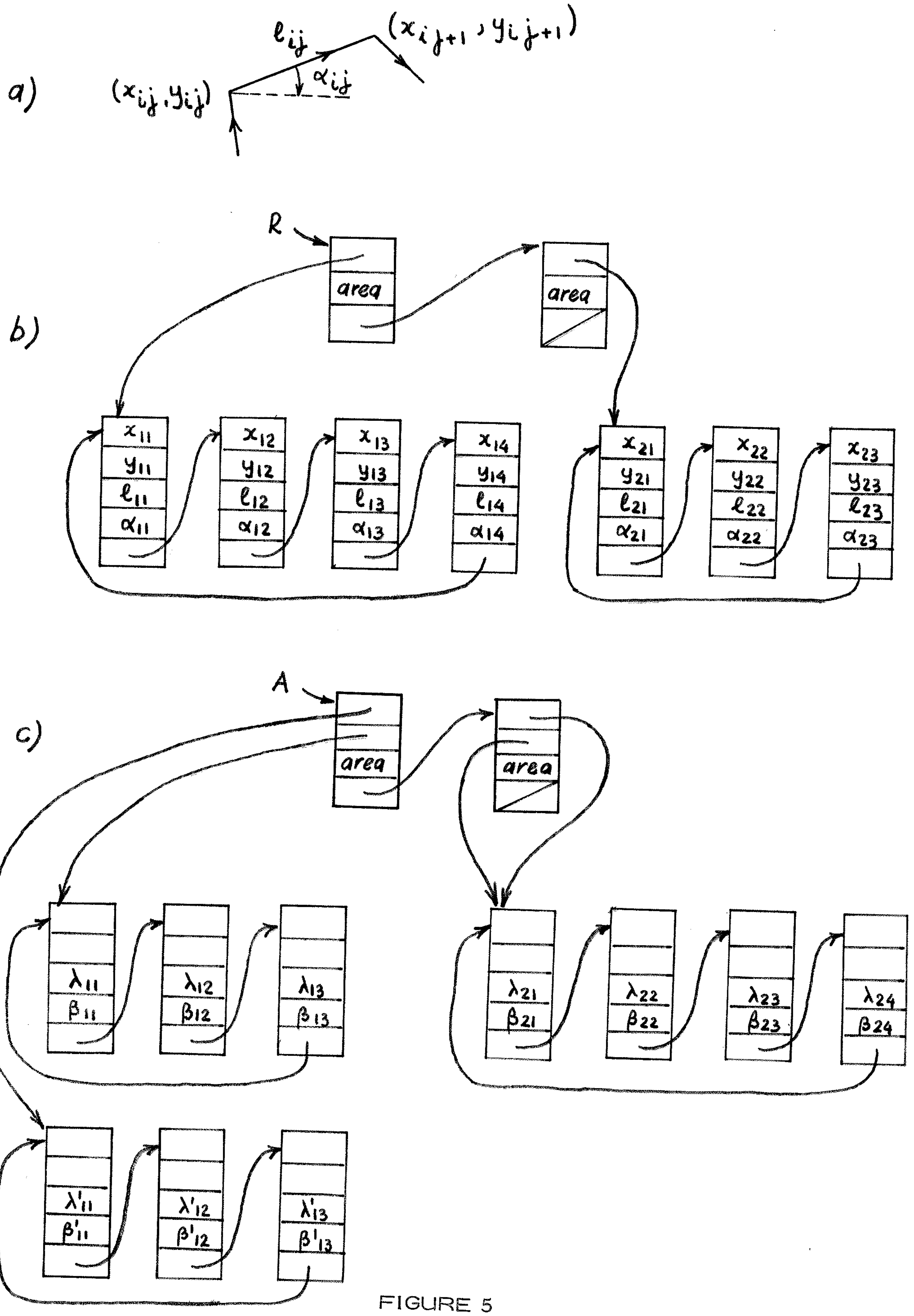

a)
pij
qkl
strong intersection
b)
pij
qkl
weak intersection
c)
pij
qkl
weak intersection

FIGURE 4



a)
(x_{ij}, y_{ij})
l_{ij}
α_{ij}
(x_{ij+1}, y_{ij+1})
b)
R
area
area
x_{11} y_{11} l_{11} α_{11}
x_{12} y_{12} l_{12} α_{12}
x_{13} y_{13} l_{13} α_{13}
x_{14} y_{14} l_{14} α_{14}
x_{21} y_{21} l_{21} α_{21}
x_{22} y_{22} l_{22} α_{22}
x_{23} y_{23} l_{23} α_{23}
c)
A
area
area
λ_{11} β_{11}
λ_{12} β_{12}
λ_{13} β_{13}
λ_{21} β_{21}
λ_{22} β_{22}
λ_{23} β_{23}
λ_{24} β_{24}
λ'_{11} β'_{11}
λ'_{12} β'_{12}
λ'_{13} β'_{13}

FIGURE 5

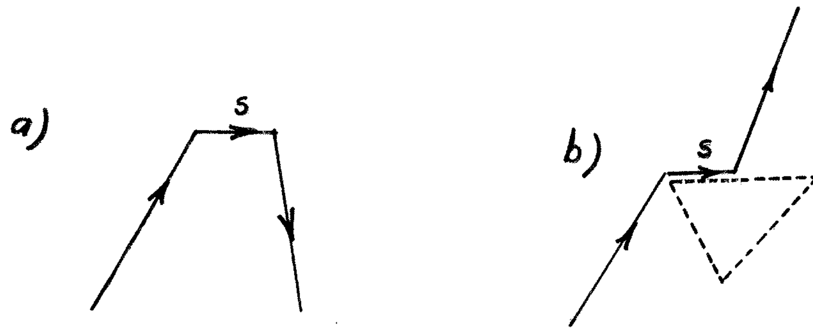


FIGURE 6

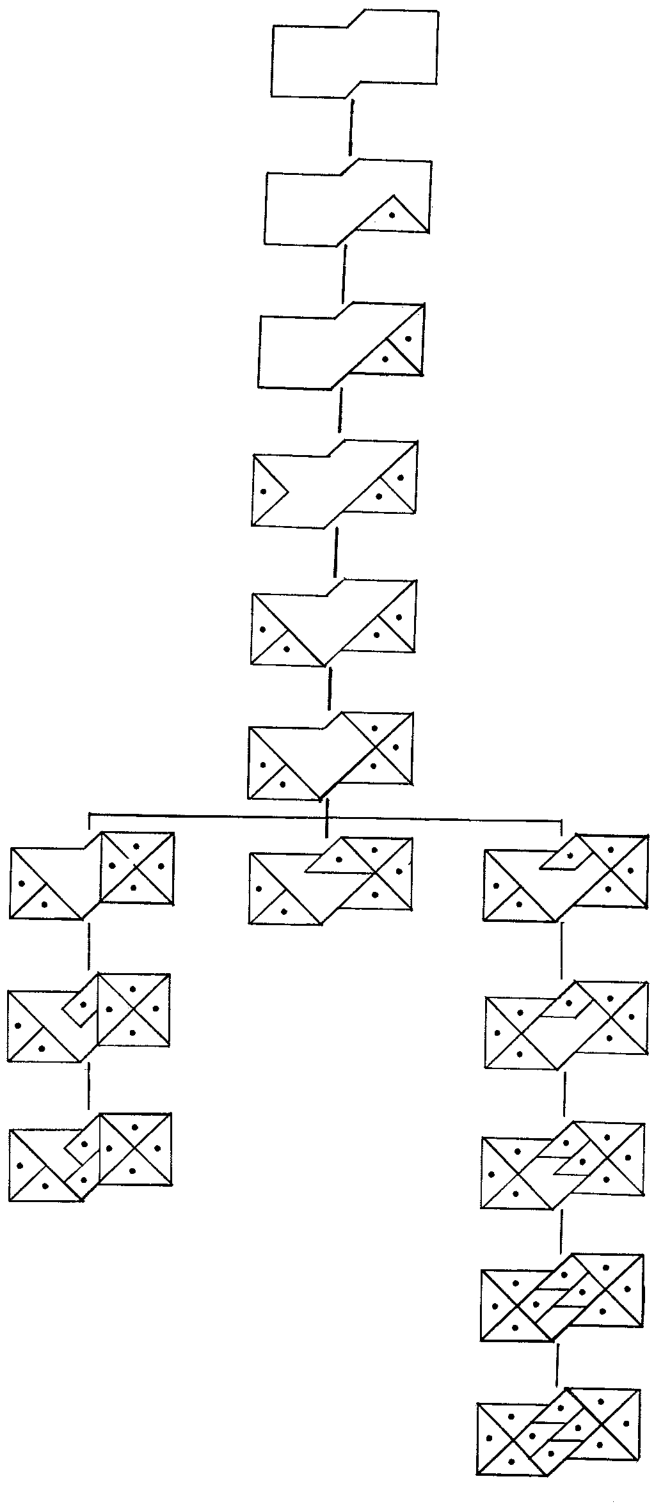


FIGURE 7

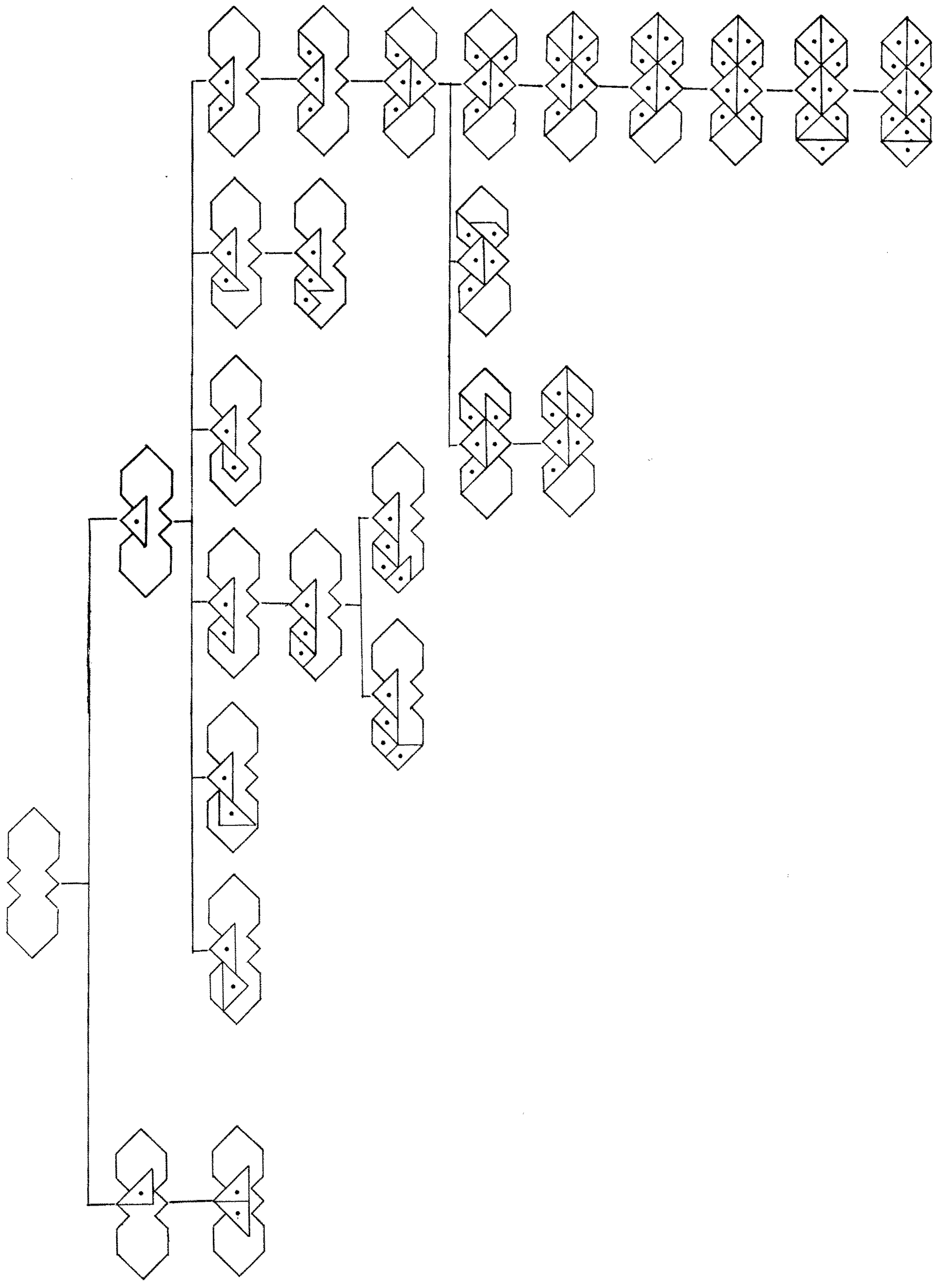


FIGURE 8

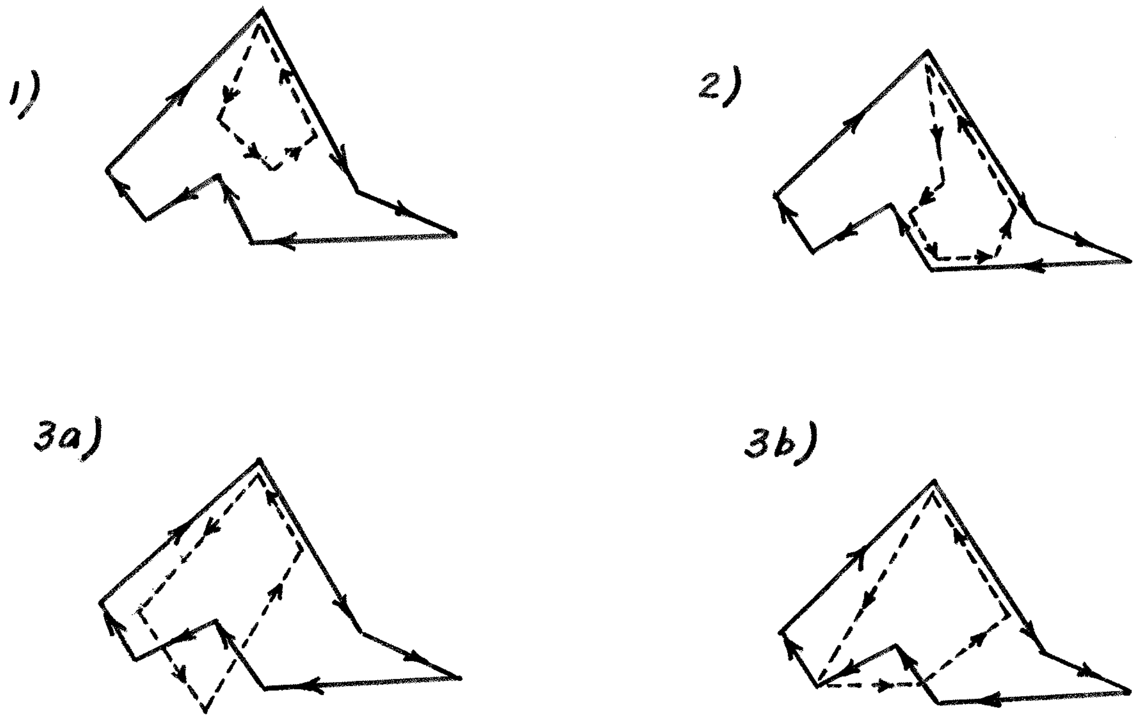


FIGURE 9

MANUSCRIPT DOCUMENTATION UNIT

A. Title:

A Computer Solution of Polygonal Jigsaw-Puzzles

B. Abstract:

This paper describes a program to solve any jigsaw-puzzle involving pieces of polygonal shape. An efficient solution has been found to depend on a number of ad hoc strategies which are described in detail in the paper. The puzzles are solved by successively placing individual pieces in the region to be covered using a depth-first tree search algorithm. A formal representation of regions, pieces, and placings of pieces is defined. The main idea behind the chosen representation is to orient clockwise the polygons making up a region, and to orient counter-clockwise the pieces to be placed. Placing a piece means computing a valid new region, i.e., one or more clockwise oriented polygons, constructed from the previous one by removing the part corresponding to the piece which is placed. The data structure and the procedures required to examine where pieces can be placed and to perform the placings are also described. All the puzzles that have so far been presented to the program have been successfully solved in a reasonable time.

C. Content Indicators:

Key Words and Phrases: Artificial intelligence, problem solving, pattern recognition, puzzles, polygonal puzzles, jigsaw-puzzles, backtrack programming, tree search algorithms.

CR Categories: 3.6, 3.63, 3.64

D. References:

[1] Floyd, R.W. "Nondeterministic algorithms," J. ACM 14 (Oct. 1967) 636-644.

[2] Freeman, H. and Garder, L. "Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition," IEEE Transactions on Electronic Computer EC-13 (Apr. 1964) 118-127.

[3] Golomb, S.W. _Polyominoes_, Charles Scribner's Sons, New York, 1965.

[4] Hoare, C.A.R. "Record Handling" in _Programming Languages_, Academic Press, New York, 1968, 291-347.

[5] Nilsson. N.J. _Problem Solving Methods in Artificial Intelligence_, McGraw-Hill, New York, 1971, Chap. 3.