AN ALGORITHM FOR SUBGRAPH ISOMORPHISM

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for
the Degree of
MASTER OF PHILOSOPHY

PAMULAPATI KRISHNA PRASAD

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067
1982

C E R T I F I C A T E

This dissertation entitled "An Algorithm For Subgraph Isomorphism" embodies the work carried out at the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi-110067.

This work is original and has not been submitted in part or full for any other degree or diploma of any other University.

P. Krishna Prasad
KRISHNA PRASAD
STUDENT

H.B. Mittal
Dr.H.B. MITTAL
SUPERVISOR

D.K. Banerji
Dr.D.K.Banerjee
Dean
School of Computer & Systems Sciences
Jawaharlal Nehru University
New Delhi-110067.

A C K N O W L E D G E M E N T S

I express my deep sense of gratitude to my thesis supervisor, Dr. H.B.Mittal, for his guidance and inspiration throughout the preparation of this work.

Thanks are due, also to Prof.D.K.Banerjee, Dean, all my teachers of the Course Work and general and library staff at the School of Computer&Systems Sciences, Jawaharlal Nehru University for providing me guidance and academic facilities.

P. Krishna Prasad
KRISHNA PRASAD

SYNOPSIS

In view of the various applications of graph theory, we would like to have a precise way of saying that a given graph is isomorphic to the subgraph of some other graph, even though they are drawn or labeled differently. To this end, a great deal of work has been done by various authors, notably by Ullmann, Ghahraman, Wong and Au, and by Cheng and Huang. Ullmann, Ghahraman, et.al, and Cheng and Huang gave backtrack algorithms involving a refinement procedure in the backtrack procedure. In what follows, we give a backtrack algorithm in which a refinement is adopted before backtracking procedure. We briefly survey, in Chapter(1), the techniques used by earlier authors for the subgraph isomorphism and related problems.

By making an appeal to the topology of graphs, we give, in Chapter(2), our refinement procedure and the backtrack algorithm. The backtrack algorithm given in Chapter(2) is a modified version of Ullmann's algorithm.

In § 2.2., we illustrate our refinement procedure by means of an example. In §2.3, we describe the algorithm, and in §2.4, we discuss the correctness of the

refinement procedure and efficiency of the algorithm.

The algorithm, given in §2.3, is considerably faster than all the other known algorithms for certain graphs.

****

C O N T E N T S

contd.....

****

CHAPTER 1

A BRIEF HISTORICAL SURVEY

1.1 Introduction:- In the past four decades we have seen a steady development of Graph Theory and it's applications which during the last ten to fifteen years have blossomed into a new period of intense activity. Some measure of this rapid expansion is indicated by the observation that until 1957 there was exactly one book on Graph Theory [14] and now we have about 3 dozen books on Graph Theory and that over a period of one year more than 500 papers on Graph Theory are published. The main reason for this acceleration in Graph Theory is in it's demonstrated application.

Any system involving a binary relation can be represented by a graph.

Because of their intutive diagrammatic representation, graphs have been found extremely useful in modeling systems arising in Physical Sciences [4, 7, 11] , Engineering [13, 5, 19] , Social Sciences [12] and economic problems [8].

1.2. Some Basic Definitions :- In what follows, we shall use terms and definitions given in Harary [11] .

Graph:- A graph G consists of a finite nonempty set V=V(G) of p points (also called nodes or vertices) together with a prescribed set X of q unordered pairs of distinct points of V. Each pair (u,v) of points is a line (also called edge) of G and e is said to join u and v, WHERE $e=(u,v)$.

Simple Graph:- We say that a graph is simple if it has no loops and no parallel edges.

Degree:- The degree of a point v_i in graph G, denoted d_i or deg v_i , is the number of lines incident with v_i. In a directed graph, the outdegree od(v) of a point v is the number of points adjacent from it, and the indegree id(v) is the number of points adjacent to it.

Directed Graph:- A directed graph or digraph D consist of finite nonempty set V of points together with a prescribed collection X of ordered pairs of distinct points. The elements of X are directed lines or arcs. By definition a digraph has no loops or multiple arcs.

Regular Graph:- A graph is called regular if all it's vertices are of the same degree. If this degree is K, then the graph is called K-regular or regular of degree K.

Adjacency Matrix:- The adjacency matrix $A=[a_{ij}]$ of a labeled graph G with P points is the P x P matrix in which $a_{ij}=1$ if v_i is adjacent with v_j and $a_{ij}=0$ otherwise. Thus there is a one-to-one correspondence between labeled graphs with P points and P x P symmetric binary matrices with zero diagonal.

Similarly, the adjacancy matrix of a labeled diagraph D is defined as $A=A(D)=[a_{ij}]$, where $a_{ij}=1$ if arc (v_i, v_j) is in D and is zero otherwise. Thus A(D) is not necessarily symmetric.

The incidence matrix $B=[b_{ij}]$ associated with a graph G, is defined to be a P x Q matrix, in which the points and lines are labelled, and in which $b_{ij}=1$ if v_i and e_j are incident and $b_{ij}=0$ otherwise, where P and Q are the number of points and edges of the graph respectively.

Incidence Matrix:- The incidence matrix $B=[b_{ij}]$ associated with a graph G, is defined to be a PXQ matrix, in which the points and lines are labelled, and in which $b_{ij}=1$ if V_i and e_j are incident and $b_{ij}=0$ otherwise, where P and Q are the number of points and edges of the graph respectively.

Degree Sequence:- A degree sequence of graph is merely a listing of the degrees of the vertices of the graph. Indegree and outdegree sequences are similarly defined. In terms of the adjacency matrix, the degree sequence can be generated by summing the rows and columns corresponding to each vertex.

Order of a Graph:- The number of vertices of a finite graph is called it's order.

Path:- A walk of a graph G is an alternating sequence of points and lines v_o , e_1, v_1 ,....., v_{n-1}, e_n, v_n, beginning and ending with points, in which each line e_i is incident with the two points immediately preceding and following it. This walk joins v_o and v_n, and may also be denoted v_o, v_1, v_2,....., v_n, it is closed if $v_o=v_n$ and is open otherwise. It is a trail if all all the lines are distinct, and a path if all the points are distinct.

Circuit:- A circuit is a path $a_1, a_2, \ldots\ldots a_q$ in which initial vertex a_i coincides with the final vertex a_q.

Connected Graph:- A graph is connected if every pair of points are joined by a path. A maximal connected subgraph of G is called a connected component or simply a component of G. Thus, a disconnected graph has at least two components.

Subgraph:- A graph G is a subgraph of G^1 if all the nodes and edges of G are in G^1.

Complete Graph:- A simple graph in which there exists an edge between every pair of vertices is called a Complete Graph or clique.

Homomorphism:- Given two relational structures R and S over the same predicate set and on sets X and Y respectively, we say that a function F: $X \rightarrow Y$ is a homomorphism if for any predicate p, $p(f(x_1), \ldots, f(x_n))$ holds in S whenever $p(x_1, \ldots, x_n)$ holds in R. We write $F: R \rightarrow S$ if f is a homomorphism.

Monomorphism:- A monomorphism is a homomorphism which is one to one, i.e. $f(x_1) = f(x_2)$ implies $x_1 = x_2$.

Morphism:- A morphism is a monomorphism from one structure to another structure.

Relational Structure:- A finite relational structure is a set of elements with given properties and relations between them.

1.3. Isomorphism:- In drawing the geometric diagram of a graph we have great freedom in the choice of the location of the nodes and in the form of the lines joining them. This may make the diagrams of the same graph look entirely different.

In view of various applications of Graph Theory, we would like to have a precise way of saying that two graphs are really the same even though they are drawn or labelled differently.

Definition:- Two graphs G_1 and G_2 are said to be isomorphic if there exists a one-to-one correspondence between their vertices and between their edges such that the incidence relationship is preserved.

The isomorphism problem is that of finding a good algorithm for determining whether two given graphs are isomorphic. The isomorphism problem has great practical significance. For example, each organic compound

can be represented by its graphical structure. The properties of chemical compounds change with their graphical structure. Thus, it is important to develop techniques to recognize isomorphic graphs as having the same structure. Again, the matching of a structured search query against data structures in the data base of an information retrieval system becomes a search for isomorphism when the data structures are interpreted as digraphs. The practical need has stimulated search for efficient procedures for deciding whether two given digraphs are isomorphic.

The digraph isomorphism problem is more general than the graph isomorphism problem. Infact, two given directed graphs may not be isomorphic but their underlying graphs may be isomorphic.

In what follows, we shall consider only undirected graphs and related results.

1.4. Lebelling Problem:- Suppose that we are analysing a picture or scene, with the aim of describing it, and that we have detected a set of objects $a_1,\ldots,a_n$ in the scene, but have not identified them unambiguously. The relationships that exist among the objects

can often be used to reduce, or even eliminate, the ambiguity.

Labelling [16] is a discrete model of ambiguity reduction process and is similar to the filtering scheme of Waltz [21] .

Let $A=\{a_1,\ldots,a_n\}$ be the set of objects to to be labeled, and $L=\{l_1,\ldots,l_m\}$ the set of possible labels. For any given object a_i, let $L_i \subseteq L$ be the set of labels that are compatible (i.e. possible for) object a_i, $1 \leq i \leq n$. For each pair of objects (a_i, a_j), where $i \neq j$, let $L_{ij} \subseteq L_i \times L_j$ be the set of compatible pairs of labels; thus $(l, l^1) \in L_{ij}$ means that it is possible that a_i be labeled l and a_j be labeled l^1. Here L_{ij} depends on the relationship between a_i and a_j in the scene. If a_i and a_j are irrelevant to one another, then there are no restrictions on the possible pairs of labels that they can have, so that $L_{ij} = L_i \times L_j$.

By a labelling $\alpha = (L_1,\ldots,L_n)$ of A we mean an assignment of a set of labels $L_i \subseteq L$ to each $a_i \in A$. We say that the labeling α is contained in the labelling $\alpha^1 = (L_1^1, \ldots, L_n^1)$ if $L_i \subseteq L_i^1$, $1 \leq i \leq n$; in this case we write $\alpha \leq \alpha^1$.

The labelling α is called consistent if, for all i and j, we have

$$(\{\lambda\} \times L_j) \cap L_{ij} \neq \emptyset \quad \text{for all} \quad \lambda \in L_i.$$

For $i \neq j$, this means that for each pair of objects (a_i, a_j) and each label λ in L_i, there exists a label λ' in L_j that is compatible with λ, i.e., $(\lambda, \lambda') \in L_{ij}$. For $i = j$, the condition reduces to

$$\lambda \in L_i \text{ implies } (\lambda,\lambda) \in L_{ii},$$

in other words every label in L_i is a possible label for a_i.

There always exist consistent labellings; in particular, the null labelling $\alpha_0=(\emptyset,\ldots,\emptyset)$ is trivially consistent. On the other hand, if $\alpha=(L_1,\ldots,L_n)$ is non null consistent labelling, then every L_i must be nonempty. It is easily seen that there exists a greatest consistent labelling, i.e., a labelling α^∞ such that

1. α^∞ is consistent
2. For any consistent labelling α we have $\alpha \subseteq \alpha^\infty$.

α^∞ may be null, i.e., there may not exist a non-null consistent labelling. We call a labelling unambiguous if it is consistent and assigns only a single label to each object.

A useful way of representing labellings is in terms of <u>labelling network</u>. This is a graph G whose nodes are the pairs (i,l), for all $1 \leq i \leq n$ and all $l \in L_i$. The nodes (i,l) and (j, l') are joined by an arc if and only if $(l, l') \in L_{ij}$. To any labelling $\alpha = (L_1,\ldots,L_n)$ there corresponds a subgraph G_α of G whose nodes are the pairs (i,l) for all $l \in L_i$. α is consistent if and only if, for each node (i,l) of G_α and each j, there exists a node (j,l') of G_α that is joined to (i,l) by an arc. α is unambiguous if and only if it is consistent and has only one node (i,l), for each i. Hence, if α is unambiguous, the subgraph G_α is a clique.

1.5. <u>Subgraph Isomosphism</u>:- In many problems in pattern recognition it is important to know if a particular feature is embedded in a pattern under investigation. Such problems are basically labelling problems and are easily described in the terminology of Graph Theory as a search for a subgraph isomosphism.

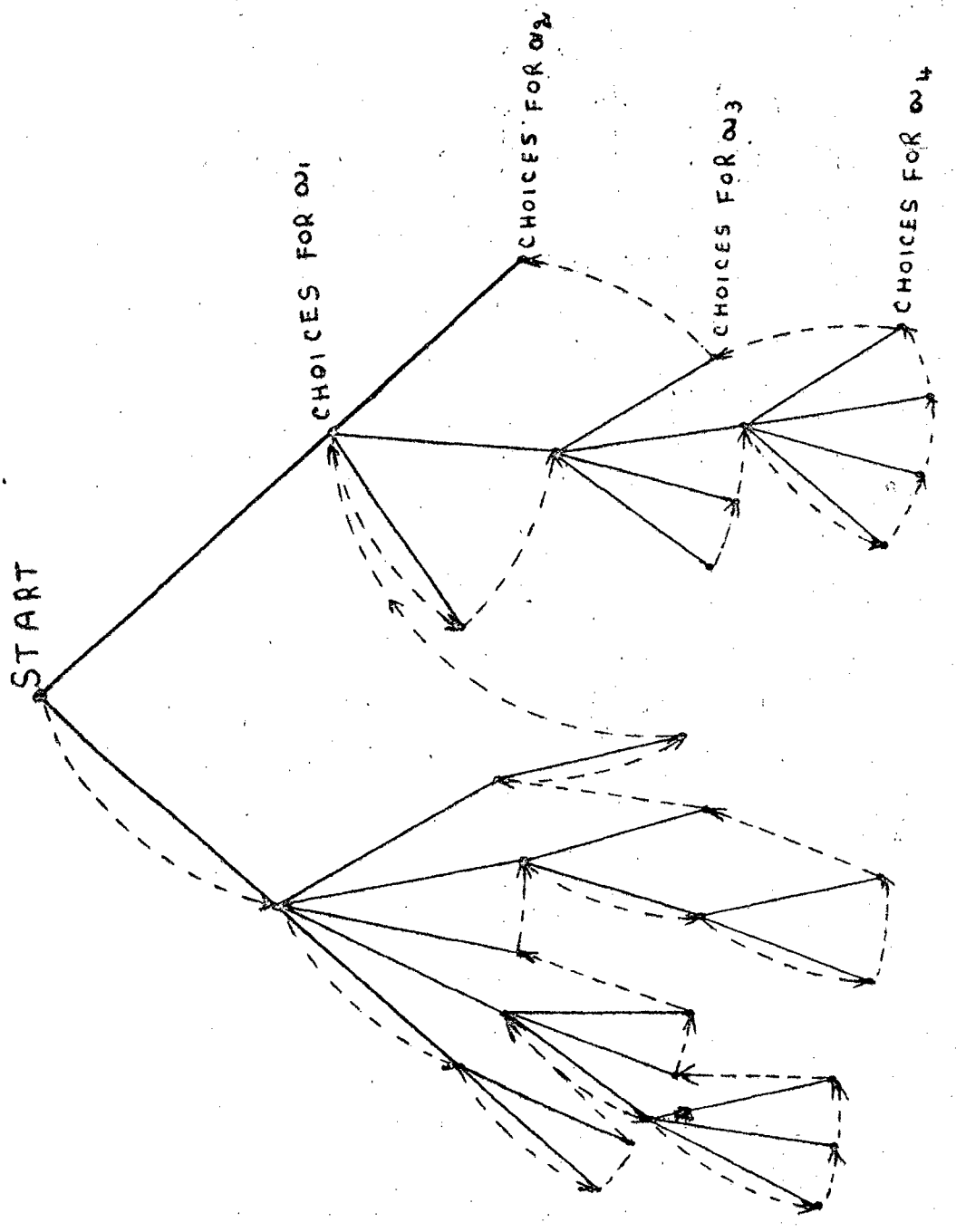
<u>Definition:</u>- For any two graphs G and H, G is said to be a subgraph of H if there exists a one to one correspondence between the vertices of G and a set of vertices of H

which preserves incidence relationships.

In the terminology of category theory, subgraph isomosphism is generalized as Graph monomorphism. If R is a relational structure describing a view of a known object and S is a relational structure describing a picture which is presented for analysis, we translate the question "Is the object in the picture" to the question "Is there a monomorphism f: R → S. Similarly given a repertoire of relational structures $R_1,....R_n$ representing views of known objects, our aim may be to find all monomorphisms from the R_i to S, $1 \leq i \leq n$.

1.6. Backtracking:- Using a computer to answer such questions as "How many ways are there to .....", "List all possible .....", or "Is there a way to .....", usually requires an exhaustive search of the set of all potential solutions.

A general technique for organising such searches, called backtrack [2,19, 15], works by continually trying to extend a partial solution. At each step of the search, if an extension of the current partial solution is not possible, we "backtrack" to a shorter partial solution and try again.


START
CHOICES FOR ω_1
CHOICES FOR ω_2
CHOICES FOR ω_3
CHOICES FOR ω_4

Backtrack, however, is only a general technique. Its straight forward application typically results in algorithms whose time requirements are prohibitive. In order to be useful, it is regarded only as a framework within which to approach the problem.

The Generalized Algorithm:- In the most general case, we assume that the solution to a problem consists of a vector $(a_1, a_2, ...)$ of finite but undetermined length, satisfying certain constraints. Each a_i is a member of a finite, linearly ordered set A_i. Thus the exhaustive search must consider the elements of $A_1 \times A_2 \times ..... \times A_i$, for $i = 0,1,2,....$ as potential solutions. Initially we start with the null vector $(\ )$ as our partial solution, and the constraints tell us which of the members of A_1 are candidates for a_1 ; call this subset S_1. We choose the least element of S_1 as a_1, and now we have the partial solution (a_1). In general, the various constraints that describe the solutions tell us which subset S_k of A_k constitutes candidates for the extension of the partial solution $(a_1, a_2..... a_{k-1})$ to $(a_1, a_2,.....,a_k)$. If the partial solution $(a_1, a_2,.....,a_{k-1})$ admits no possibilities for a_k, then $S_k = \emptyset$, and so we backtrack

and make a new choice for a_{k-1}. If there are no new choices for a_{k-1}, we backtrack still farther and make a new choice for a_{k-2}, and so on.

We picture this process interms of depth first tree traversal (pre-order). The subset of $A_1 \times A_2 \times \ldots \times A_i$ for $i = 0,1,2,\ldots$, that is searched, is represented as a search tree as follows: The root of the tree (the 0th level) is the null vector. Its sons are the choices for a_1 and in general, the nodes at the kth level are the choices for a_k, given the choices made for $a_1, a_2, \ldots, a_{k-1}$ as indicated by the ancestors of these nodes. Backtrack traverses the nodes of the tree as indicated by dashed lines. In asking whether a problem has a solution $(a_1, a_2, \ldots)$, we are asking whether any nodes in the tree are solutions. In asking for all solutions, we want all such nodes.

1.7. Algorithms and Their Complexity:- Algorithms can be evaluated by a variety of criteria. Most often we shall be interested in the rate of growth of the time or space required to solve larger and larger instances of a problem. With a problem is associated an integer, called the size of the problem, which is a measure of the quantity of input data. For example, the size of a graph problem might be the number of edges or the number of vertices.

The time needed by an algorithm expressed as a function of the size of a problem is called the time complexity of the algorithm. The limiting behaviour of the complexity as size increases is called the asymptotic time complexity.

It is the asymptotic complexity of an algorithm which ultimately determines the size of problem that can be solved by the algorithm. If an algorithm processes inputs of size n in time cn^2, for some constant c, then we say that the time complexity of that algorithm is $O(n^2)$, read "order n^2".

A polynomial time algorithm is defined as one whose running time, that is the number of elementary bit operations it performs, on an input string of length n is bounded above by some polynomial P_n. P is the class of all problems that can be solved by such an algorithm. All problems with algorithms whose running time or number of outputs are necessarily exponential in the number of inputs are not in P.

The state of an algorithm is defined to be the combination of the location of the instruction currently being executed and the values of all variables. An algorithm is said to be deterministic if for any given state there is at most one valid next state. Thus

a deterministic algorithm can do only one thing at a time. A nondeterministic algorithm is one in which, for any given state, there may be more than one valid next state. Thus, a nondeterministic algorithm can do more than one thing at a time. The class NP is defined to be the class of all problems that can be solved by non-deterministic algorithms that run in polynomial time. Clearly $P \subseteq NP$.

A problem P is defined to be NP-hard if a deterministic polynomial time algorithm for its solution can be used to obtain a deterministic polynomial time algorithm for every problem in NP. Thus, a problem is NP-hard if it is at least as hard as any problem in NP. An NP-hard problem in NP is called NP-complete, such problems are at least as hard as any problem in NP, but no harder [15].

1.8. Subgraph Isomorphism Algorithms - A Survey:-

Subgraph Isomorphism problem is much more complicated than graph Isomorphism problem. However, in what follows we shall consider only graph isomorphism problem, eventhough, digraph isomorphism problem is more general than the isomorphism problem.

If we have two graphs G and H of orders m and n respectively, $m < n$ and if the two graphs are labeled arbitrarily, then the subgraph isomorphism problem can be solved by brute force enumeration method. However, such a method can be efficient for very small graphs.

Thus, in order to find a good algorithm (i.e. an algorithm which is not exponential, but polynomial in the length of input) some indirect methods have been used based on various properties of the graphs. First, procedures have been developed that partition the set of vertices of the two graphs on the basis of a common property shared by all vertices in the block of the partition. One such common property is the degr sequence of the vertices and vertices with the same degree are placed in one block. By submitting the

two graphs to a battery of procedures, based on a variety of properties "one hopes" eventually to establish subGRAPHisomorphism or lack of it. A number of heuristic algorithms have been devised which fall in this class. A second approach is the brute force enumeration method followed by some refinement procedure. This consists of the class of backtrack algorithms. Some authors have devised backtrack algorithms alongwith a refinement procedure which does the early pruning of infeasible subtrees from the backtrack tree.

A third approach is algebraic and is basically category theoretic and aims to find all monomorphisms of one graph into another. However, this approach is logically the same as the second approach above.

In what follows, we give a brief account of all the heuristic and backtrack procedures known for the subgraph isomorphism and related results.

Backtrack Algorithm of Salton and Sussengurth [18]:-

Salton and Sussengurth developed a topological structure matching procedure for subgraph isomorphism. Based on the degree sequence of the two graphs, they partitioned the vertices of the graphs into disjoint classes and developed heuristical algorithm to match vertices of the two graphs on the basis of their connectivity patterns.

Result of Sakai, Nagao and Matsushima [17]:- In an attempt to analyse photographs taken by aeroplanes, earth satellites, etc., Sakai, Nagao and Matsushima gave an algorithm to detect topologically equivalent substructures in two pictures. They treated the problem of finding parts in two pictures which are in a linear transformation relation.

Generalized Result of Barrow, Ambler and Burstall: [1] : Barrow, Ambler and Burstall [1] in 1972 developed techniques for scene analysis, that is, of deducing from a single two dimensional image the organization of the scene which it depicts, in terms of objects and their interrelationships. To this end they considered the idea of a finite relational structure and described

hierarchical matching process to find whether one structure is a substructure of another. They separated the process into two parts, one of which is dependent on the notion of relational structures and their monomorphisms, and the other is a more general algebraic notion of a hierarchical descriptive system couched in category theoretic terminology, and hence allowing other interpretations of the notion of morphism and structures, other than relational structures, which can be used to cope with the case where the relations are replaced by real valued functions and approximate matches are desired.

Thus they developed formalisms to compare a hierarchy of structures with a given picture structure and find all monomorphisms from the structures in the hierarchy to the picture structure. Such monomorphisms describe the picture by saying that some known object occurs in it.

A Procedure by Waltz:- Waltz [21] considered the problem of ascertaining shapes of unfamiliar objects and that of factoring out shadows when looking at scenes. To this end he gave procedure which construct three dimensional descriptions from line drawing which are obtained

from scenes composed of plane faced objects under various lighting conditions and assigned labels to line segments and junctions in the scene. He subdivided one or more edge labels into several new labels, embodying finer distinctions and then recomputed the junction label lists to include these new distinctions. Creating a large list of junction labels, Waltz describes method of using selection rules to eliminate as many labels as possible on the basis of relatively local information and developed filter programme to remove labels which cannot be part of any total scene labelling based on the context of the junction.

Method of Rosenfeld, Hummel and Zucker:- Rosenfeld, Hummel and Zucker[16]described several models for analysing a picture or scene with the aim of describing it unambiguously, by using the relationship that exist among the objects in the picture. To this end they gave a parallel algorithm for constructing the greatest consistent labelling of the picture under consideration. Their algorithm is basically a parallel version of the filtering process used by Waltz[21].

Backtrack Algorithm by Ullmann:- Ullmann[20] in 1976, improved upon the brute force enumeration procedure for detecting subgraph isomorphism by using the topology of the graph. His method is similar to that of Salton and Sussenguth[18] except that his process does not work on the two graphs separately and in the organization of the refinement procedure.

Ullman designed the enumeration algorithm to find all of the isomorphisms between a given graph G of order m, and a further given graph H of order n, $m \leq n$, given by their adjacency matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ respectively. Defining M^0 to be an m X n element matrix $M^0 = [m_{ij}]$ in accordance with

$$m_{ij}^0 = \begin{cases} 1 \text{ if the degree of the } j_{th} \text{ vertex of H is greater than or equal to the degree of the } i_{th} \text{ vertex of G.} \\ 0 \text{ otherwise.} \end{cases}$$

The algorithm defines an m (rows) X n(columns) matrix M^1 whose elements are 1's and 0's such that each row contains exactly one 1 and no column contains more than one 1. The matrix $M^1 = [m_{ij}^1]$ is used to permute

the rows and columns of B to produce a further matrix C. Specifically, $C = [C_{ij}] = M^1 (M^1 B)^T$, where T denotes transposition. If it is true that

$$(a_{ij} = 1) \Rightarrow (C_{ij} = 1) \quad \ldots\ldots (1)$$

for all $1 \leq i \leq m$, $1 \leq j \leq n$, then M^1 specifies an isomorphism between G and a subgraph of H. In this case, if $m^1_{ij} = 1$, then the j_{th} point of H corresponds to the i_{th} point of G in this isomorphism.

To reduce the amount of computation required for finding subgraph isomorphisms, Ullmann employed a refinement procedure that eliminates some of the 1's from the matrices M, thus eliminating successor nodes in the tree search. The refinement procedure"refine" works in the following manner:

Let $V_{\alpha i}$ and $V_{\beta j}$ be vertices in G and H respectively and let $\{V_{\alpha 1}, \ldots, V_{\alpha x}, \ldots V_{\alpha \gamma}\}$ be the set of direct descendants of $V_{\alpha i}$. Let M^1 be the matrix associated with any given isomorphism under $M(= M^0)$. By definition of subgraph isomorphism it is necessary that if $V_{\alpha i}$ corresponds to $V_{\beta j}$ in the isomorphism, then for each $X=1,2,\ldots \gamma$ there must exist a point $V_{\beta y}$ in H that is adjacent to $V_{\beta j}$, such that $V_{\beta y}$ corresponds to $V_{\alpha x}$ in

the isomorphism. If $V_{\beta y}$ corresponds to $V_{\alpha x}$ in the isomorphism, then the element of M^1 that corresponds to $\{V_{\alpha x}, V_{\beta y}\}$ is 1. Therefore if $V_{\alpha i}$ corresponds to $V_{\beta j}$ in any isomorphism under M, then for each $X = 1,2,\ldots, \gamma$ there must be a 1 in M corresponding to some $\{V_{\alpha x}, V_{\beta y}\}$ such that $V_{\beta y}$ is adjacent to $V_{\beta j}$. Thus, if $V_{\alpha i}$ corresponds to $V_{\beta j}$ in any isomorphism under M, then

$$(a_{ix} = 1) \Rightarrow (\exists y): (m_{xy} \cdot b_{yj} = 1) \ldots\ldots (2)$$

for all $1 \leq x \leq m$, and $1 \leq y \leq n$.

The refinement procedure simply tests each 1 in M to find whether condition (2) is satisfied. For any $m_{ij} = 1$ such that (2) is not satisfied, $m_{ij} = 1$ is changed to $m_{ij} = 0$. Such changes may cause condition (2) to be no longer satisfied for further 1's in M, so that further changes can be made, and so on. Infact, the refinement procedure applies condition (2) in turn to each 1 in M, and it then does this over and over again until there is an iteration in which all the 1's in M are processed and none of them is changed to 0. However, the refinement procedure may leave M unchanged. A necessary and sufficient condition for subgraph

isomorphism is that the refinement procedure leaves M^1 unchanged. This follows because if M^1 is unchanged by the refinement procedure then (2) holds for each 1 in M^1. Therefore M^1 specifies a one to one mapping of G into H such that if two vertices are adjacent in G then the two corresponding vertices in H are adjacent.

The algorithm uses an n-bit binary vector $\{F_1,\ldots, F_i,\ldots,F_n\}$ to record which columns have been used at an intermediate state of the computation; $F_i=1$ if the i_{th} column has been used. The algorithm also uses a vector $\{H_1,\ldots, H_d,\ldots, H_m\}$ to record which column has been selected at which depth: $H_d = K$ if the K_{th} column has been selected at depth d. In the algorithm, the matrix M_d is a stored copy of matrix M at depth d.

The Algorithm [20]

Step 1. $M=M^0$, $d=1$, $H_1=0$
for all $i=1,\ldots,m$ set $F_i = \emptyset$
refine M, if exit FAIL, then terminate algorithm

Step 2. If there is no value of j such that $m_{dj}=1$ and $F_j = 0$ then got to step 7.
$M_d = M$
if $d = 1$ then $K=H_1$ else $K = 0$

Step 3. $K = K + 1$

if $m_{dk} = 0$ or $F_k = 1$ then got to step 3.

for all $j \neq K$ set $m_{dj} = 0$

refine M, if exit FAIL then go to step 5.

Step 4. If $d < m$ then go to step 6 else give output to indicate that an isomorphism has been found.

Step 5. If there is no $j > K$ such that $m_{dj} = 1$ and $F_j = 0$ then goto step 7.

$M = M_d$

go to step 3.

Step 6. $H_d = K$, $F_K = 1$, $d = d+1$

go to step 2.

Step 7. If $d = 1$ then terminate algorithm.

$F_K = 0$, $d = d - 1$, $M = M_d$, $K = H_d$

go to step 5.

Algebraic Method of Ghahraman, Wong and Au:- Ghahraman, Wong and Au [9] used the backtrack procedure of Ullmann [20] and applied necessary conditions for the existence of a monomorphism during the search of the decision tree to prune infeasible subtrees and to reduce the extent of search. They proposed two necessary conditions (strong and weak) for the existence of a subgraph isomorphism in terms of a cluster. The weak necessary conditions is equivalent to the refinement process in [20] and the strong necessary condition imposed additional requirements which can lead to an early pruning of infeasible subtrees.

Procedure of Cheng and Huang:- Cheng and Huang [6] use a combination of refinement and tree search by using the Ullmann's 0,1 matrix representation and Berztiss [3] elementary K formula concept. They express the constraints as a set of constraint nodes. In their result, node is an expression which specifies the restrictions about possible mappings between vertices. They gave a parallel implementation of their technique.

CHAPTER 2

A FAST SUBGRAPH ISOMORPHISM ALGORITHM

2.1. Introduction:- Given two graphs G and H, the problem of determining whether G is a subgraph of H has important applications in Pattern Recognition, Image Analysis, Information Processing, etc. The problem can be solved by brute force enumeration method which is principally a decision tree search algorithm. However, such a method can be practical for graphs of small size.

Most of the previous works related to subgraph isomorphism problem involve (1) a refinement procedure, and (2) an exhaustive tree search.

In view of the works of Salton and Sussengurth [18], Sakai, Nagao and Matsushima [17], Barrow, Ambler and Burstall [1], Waltz [21], and Rosenfeld, Hummel and Zucker [16], Ullmann [20] in 1976 gave a backtrack algorithm for subgraph isomorphism problem.

Ullmann's algorithm consists of a decision tree search method which makes use of the degrees of the vertices of the two graphs. The matching of vertices is made by a refinement procedure based on the connectivity property of the two graphs.

Ghahraman, Wong and Au[9]proposed an algorithm totally similar to that of Ullmann which uses a strong necessary condition for early pruning of infeasible subtrees.

Cheng and Huang[6]made use of Ullmann's algorithm [20] and K-formula of Baetiss [3] to give an algorithm for subgraph isomorphism. They used the K-formula to implement the refinement procedure of Ullmann [20].

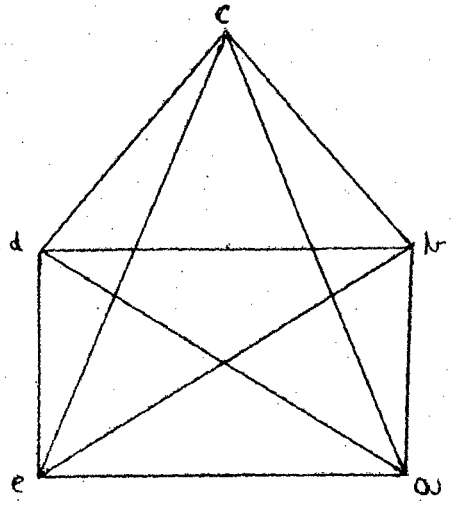
In what follows we propose to improve upon the brute force enumeration method, for subgraph isomorphism, by introducing a procedure to eliminate meaningless search performed in all the earlier algorithms. To this end, based on the topology of graph G, we make a preprocessing of Graph H and then use Ullmann's algorithm on the reduced graph. Thus we introduce a refinement procedure before searching for a subgraph isomorphism.

In §2.2 we illustrate the proposed procedure by means of an example. In §2.3 we describe the algorithm. We conclude the chapter in § 2.4, where we discuss correctness and efficiency of the algorithm.

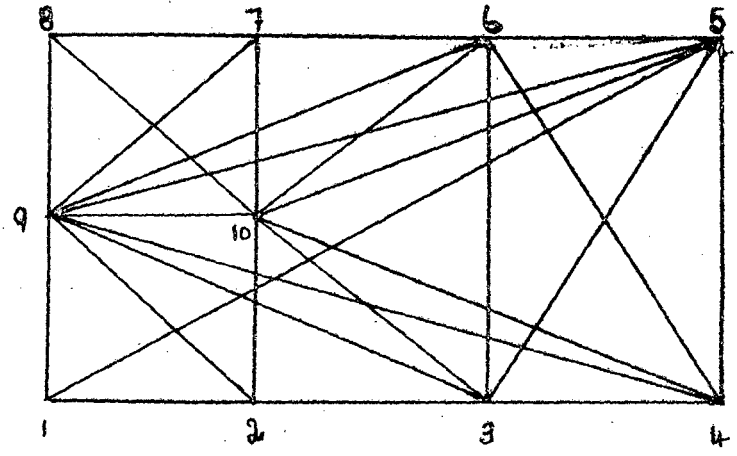
2.2. Some Basic Results and Definitions:- We assume that we are given two graphs G and H of orders m and n respectively, $m \leq n$. In order to test whether G is isomorphic to a subgraph of H, we shall subject H to a refinement procedure which is based on the topology of G. In order to fix ideas, we show, by means of an example, that if G is isomorphic to a subgraph S of H, then G will remain isomorphic to the same subgraph S of H^1, where H^1 is a subgraph of H and is obtained by deleting all the vertices of H which cannot be associated with any vertex of G under any isomorphism.

Example: Let the graphs G and H be of orders 5 and 10 respectively. We assume that the two graphs are given in the form of their adjacency matrix.

We assume that the vertices in the two graphs have been labeled according to some order (this does not imply that the vertices are ordered in any way).



GRAPH - G



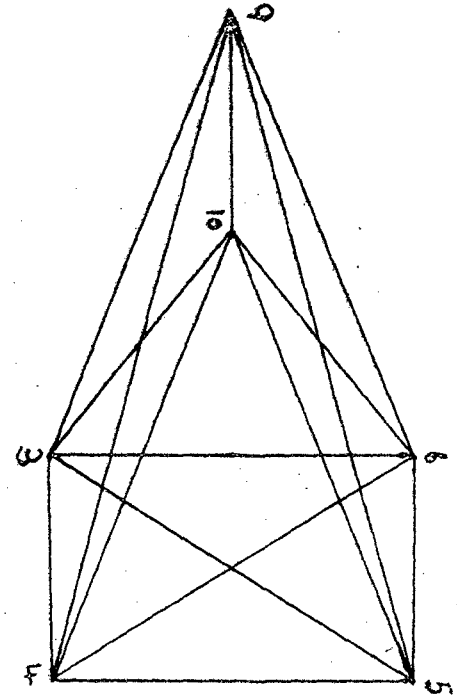
GRAPH - H

	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	1	1
c	1	1	0	1	1
d	1	1	1	0	1
e	1	1	1	1	0

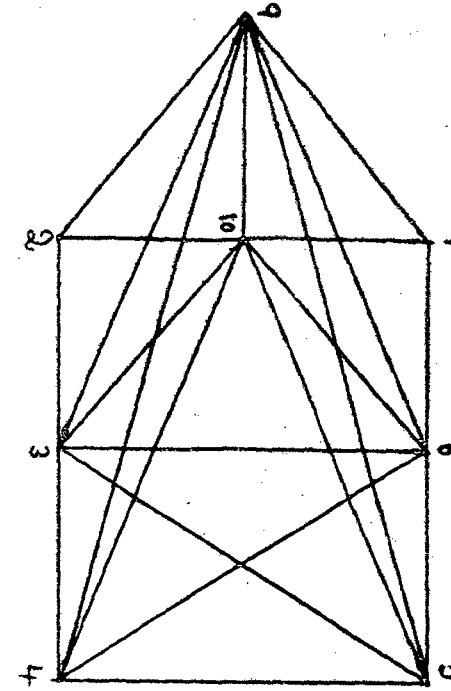
Adjacency matrix of G

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	1	0	0	0	1	0
2	1	0	1	0	0	0	0	0	1	1
3	0	1	0	1	1	1	0	0	1	1
4	0	0	1	0	1	1	0	0	1	1
5	1	0	1	1	0	1	0	0	1	1
6	0	0	1	1	1	0	1	0	1	1
7	0	0	0	0	0	1	0	1	1	1
8	0	0	0	0	0	0	1	0	1	1
9	1	1	1	1	1	1	1	1	0	1
10	0	1	1	1	1	1	1	1	1	0

Adjacency matrix of H



GRAPH - H_1



GRAPH - H_2

	2	3	4	5	6	7	9	10
2	0	1	0	0	0	0	1	1
3	1	0	1	1	1	0	1	1
4	0	1	0	1	1	0	1	1
5	0	1	1	0	1	0	1	1
6	0	1	1	1	0	1	1	1
7	0	0	0	0	1	0	1	1
9	1	1	1	1	1	1	0	1
10	1	1	1	1	1	1	1	0

Adjacency matrix of H_1

	3	4	5	6	9	10
3	0	1	1	1	1	1
4	1	0	1	1	1	1
5	1	1	0	1	1	1
6	1	1	1	0	1	1
9	1	1	1	1	0	1
10	1	1	1	1	1	0

Adjacency matrix of H_2

For testing whether G is a subgraph of H, we make use of the neighbourhood structure of the vertices of G and H. To this end, we first calculate the degree sequences for G and H.

Labels of vertices of G-: a b c d e

Degree sequence for G -:(4, 4, 4, 4, 4)

Labels of vertices of H : 1 2 3 4 5 6 7 8 9 10

Degree sequence of H :(3 4 6 5 6 6 4 3 9 8)

We see from the degree sequence for H, that the vertices labeled 1 and 8 have degree less than the degree of every vertex in G. Hence no vertex of G can be matched with vertices labeled 1 and 8 in H. Thus, if G is a subgraph of H, G will remain to be a subgraph of H even if the vertices 1 and 8 and all edges incident with them are removed from H. Removing vertices labeled 1 and 8 from H, we get a new graph H_1.

Again from the adjacency matrix for H_1, we calculate the degree sequence for H_1.

Label of vertices of H_1 : 2 3 4 5 6 7 9 10

Degree Sequence for H_1 : (3, 6, 5, 5, 6, 3, 7, 7)

Again, the vertices labeled 2 and 7 in H_1 have degree less than the degree of every vertex in G. In case G is a subgraph of H_1, G must remain subgraph of H_1 even if all the vertices labelled 2 and 7 alongwith edges incident with them are removed from H_1 . Deleting vertices labeled 2 and 7 from H_1, we get a new graph H_2.

Again, from the adjacency matrix of H_2, we calculate the degree sequence for H_2 .

Label of vertices of H_2 : 3 4 5 6 9 10

Degree Sequence for H_2 : (5, 5, 5, 5, 5, 5)

Infact, in the above example all the subgraphs of H isomorphic to G can be easily found by inspection of H_2. The refinement procedure reduced the complexity of the problem considerably.

Our algorithm for subgraph isomorphism is to apply the backtrack enumeration procedure after refining the graph H.

2.3. The Algorithm:- The graphs G and H are assumed to be given in the form of their adjacency matrix. Let the graphs G and H be of orders n and m respectively, $n \leq m$. We label the vertices of G and H arbitrarily and using adjacency matrix, we calculate the degree sequence of vertices of G, and find the smallest degree from the degree sequence. Next we calculate the degrees of vertices of H one by one. If the degree of any vertex of H is found to be less than the smallest entry in the degree sequence of G, then we delete that from H alongwith all the edges incident with that vertex, without re-labeling the graph, once again we calculate the degree of vertices of the resulting graph and if the degree of any vertex is less than the degree of every vertex of G, then that vertex is deleted. We keep repeating this procedure until every vertex is of degree at least as great as the smallest entry in the degree sequence of G. If at any stage the order of the reduced graph of H becomes less than that of G, then the algorithm terminates.

Let the vertices of H be labeled V_i, $1 \leq i \leq n$. Our subgraph isomorphism algorithm consists of two sub-algorithms, namely Algorithm 1 and Algorithm 2. Algorithm 1 is a refinement procedure which is performed on

the vertices of graph H. Also, we assume that, for $1 \le i \le n-1$, v_i is adjacent to v_{i+1}. Algorithm 2 consists of the brute force enumeration method given by [20], which operates on the nxm matrix M^0, defined by

$$m^0_{ij} = \begin{cases} 1 \text{ if the degree of } j_{th} \text{ vertex of H is greater} \\ \text{than or equal to the degree of the } i_{th} \\ \text{vertex of G.} \\ 0 \text{ otherwise.} \end{cases}$$

where $1 \le i \le n$, $1 \le j \le m$. In defining the matrix M^0, we assume that the consecutive rows labeled v_i and v_{i+1}, of M^0, are such that v_i is adjacent to v_{i+1} in graph G, $1 \le i \le n-1$.

Algorithm 2 is the enumeration algorithm designed to find all of the isomorphisms between a given graph G and subgraphs of a further given graph H. Let the adjacency matrices of G and H be given by $A=[a_{ij}]$ and $B=[b_{ij}]$, respectively.

We define an M^1 matrix to be a n(rows)xm (columns) matrix whose elements are 1's and 0's, such that each row contains exactly one 1 and no column contains more than one 1. The matrix $M^1=[m^1_{ij}]$ is used to permute the rows and columns of matrix B to produce a further matrix C. Specifically, we define $C=[c_{ij}] = M^1(M^1B)^T$, where T denotes transposition. If it is true that

$$(a_{ij}=1) \Longrightarrow (c_{ij}=1), \quad \ldots\ldots\ldots\ldots\ldots (1)$$

then M^1 specifies an isomorphism between G and a subgraph of H. In this case, if $m^1_{ij}=1$, then the j_{th} vertex of H corresponds to the i_{th} vertex of G in this isomorphism.

Algorithm 2 works by generating all possible matrices M^1

such that for each and every element m^1_{ij} of M^1, $(m^1_{ij}=1) \Rightarrow (m^0_{ij}=1)$. For each such matrix M^1 the algorithm tests for isomorphism by applying condition(1). Matrices M^1 are generated by systematically using the adjacency structure of G and changing to 0 all but one of the 1's in each of the rows of M^0_j subject to the condition that no column of a matrix M^1 may contain more than one 1. In the search tree, the terminal nodes are at depth d=n and they correspond to distinct matrices M^1. Each nonterminal node at depth $d<p_\alpha$ corresponds to a distinct matrix M which differs from M^0 in that in d of the rows, all but one of the 1's has been changed to 0.

The algorithm uses a m-bit binary vector $\{F_1, \ldots, F_i, \ldots, F_m\}$ to record which columns have been used at an intermediate stage of the computation; $F_i=1$ if the i_{th} column has been used. The algorithm also uses a vector $\{H_1, \ldots, H_d, \ldots, H_n\}$ to record which column has been selected at which depth; $H_d=K$ if the K_{th} column has been selected at depth d.

ALGORITHM

ALGORITHM (1) :

Step 1. Calculate the degree of vertices of G. Find the least degree x. set i=0.

Step 2. i=i+1

Step 3. Calculate the degree $d(v_i)$ of the i_{th} vertex v_i of H. If $d(v_i) < x$, go to step 5

Step 4. If m<n=(the total number of vertices deleted), then output "No Isomorphism". If i≠m, construct M^o, go to Algorithm 2.

Step 5. Delete the i_{th} vertex of H. If i=m, construct, M^o, go to Algorithm 2. If i<m, go to step 2.

ALGORITHM (2) :

Step 1. $M=M^o$; d=1; $H_1=0$
For i=1,2,.....,m, set $F_i=0$

Step 2. If there is no value of j such that $m_{dj} = 1$ and $F_j = 0$ then go to step 7;
$M_d = M$
If d=1 then $K=H_1$, else K=0

Step 3. K=K+1
If $m_{dk}=0$ or $F_k=1$ then go to step 3
for all j≠k set $m_{dj}=0$

Step 4. If d < n then go to Step 6 else use condition (1) and give output if an isomophism is found

Step 5. If there is no j>k such that $m_{dj}=1$ and $F_j=0$ then go to step 7.
$M=M_d$;
go to step 3.

Step 6. $H_d=K$, $F_k=1$, d=d+1, go to step 2

Step 7. If d=1 then algorithm terminates.
$F_k=0$, d=d-1, $M=M_d$, $K=H_d$, go to step 5.

2.4 CONCLUSIONS:-

Given any two graphs G and H, of orders n and m respectively, n≤m, if G is isomorphic to a subgraph S of H, then G will remain to be isomorphic to the subgraph S of H even if all the vertices of H which cannot be matched to any vertex of G are deleted from H. In fact, we have a theorem to be proved.

Theorem:- If G is isomorphic to a subgraph S of H, then G will remain to be isomorphic to the subgraph S_1 of the subgraph H^1, of H, obtained by deleting all vertices of H which are of degree less than the smallest degree in the degree sequence of G.

Proof:- If G is isomorphic to a subgraph S of H, then G and S must have the same number of vertices and edges and same topological structure. (i.e., the same degree sequence and connectivity). Thus, if there is a vertex x of H which is of degree less than the degree of every vertex in G, then under an isomorphic mapping, no vertex of G can be associated with x. Thus there cannot be a mapping of G onto a subgraph of H which contains x and some vertices of S. Thus x and all the edges incident with x do not contribute to the structure of S and hence can be removed from H without effecting S. Since x and S are arbitrary, the result follows.

Given any two graphs G and H, if G is isomorphic to a subgraph of H, then clearly G will remain to be isomorphic to the same subgraph of H after deleting vertices which cannot be matched to any vertex of G. Such a refinement destroys the topology of graph H but retains all structure which is

relevent to the subgraph isomorphism. Again, since the adjacent rows of the matrix M^0 have label of vertices of G, which are adjacent in G, the use of Algorithm 2 on M^0 leads to a traversal of the graph G in a systematic manner. By using a suitable graph invariant, in the backtrack procedure, the number of matrices M^1 generated can be reduced. Such a traversal of the Graph G elemi-nates the useless search which is performed in the algorithm given by Ullman [20] .

Thus, the use of refinement procedure in Algorithm 1, and systematic search performed by Algorithm 2, reduces the complexity of the subgraph isomorphism problem con-siderably. Thus our algorithm improves upon the results given by earlier authors.

BIBLIOGRAPHY

1. Barrow, H.G., Ambler, A.P., and Burstall, R.M. — Some techniques for recognizing Structures in pictures. Frontiers of Pattern-Recognition, Ed. S. Watanabe, Academic Press, New York, 1972; pp.1-29.

2. Bitner, J.R., and Reingold, E.M. — Backtrack Programming techniques. Communication of A.C.M., Vol.16, 1975, pp.651-656.

3. Berztiss, A.T. — A Backtrack procedure for isomorphism of directed graphs. Journal of A.C.M., Vol.20, 1973, pp.365-377.

4. BuSacker, R.G. and Saaty, T.L. — Finite graphs and Networks: An introduction with applications. Mc Graw-Hill Book Company, New York, 1965.

5. Chen, W.K. — Applied graph theory, North-Holland Publishing Company, Amsterdam, 1971.

6. Cheng, J.K. and Huang, T.S. — A subgraph isomorphism algorithm using resolution. Pattern Recognition, Vol 13, 1981, pp.371-379.

7. Christofides, N. — Graph theory ; An Algorithmic approach, Academic Press, New York, 1975.

8. Ford, R.L. and Fulkersen, D.R. — Flows in Networks; Princeton Univ. Press, Princeton, New Jersey, 1962.

9. Ghahraman, D.E., Wong, A.K.C. and Au, T. — Graph monomorphism algorithms; I EEE Trans. on Systems, Man and Cybernetics. Vol. SMC-10, 1980, pp.189-196.

10. Golomb, S., and Baumert, L. — Backtrack Programming; Journal of A.C.M., Vol.12, 1965, pp.516-524.

11. Harary, F. — Graph Theory ; Addison-Wesley Publishing Company, Reading, Massachussets, 1972.

12. Harary, F., Narman, R.Z and Darwin, C. — Structural models; An introduction to the theory of directed graphs; John Wiley and Sons, Inc., New York, 1965.

13. Henley, E, and Williams,R.A. — Graph theory in modern engineering, Computer aided design, Control, Optimization, Reliability analysis; Academic Press, New York, 1973.

14. Kônig , D. — Theorie der endlichen und unèndlichen Graphen; Lei Pzig, 1936.

15. Reingold, E.M., Nevergelt, J, and Deo, N. — Combinatorial algorithms; Theory and Practice; Prentice Hall, Inc, Englewood Cliffs, New Jersey, 1977.

16. Rosenfeld, A, Hummel R.A., and Zucker,S.W. — Scene labelling by relaxation operations; I E E E Trans. on Systems, Man and Cybernetics, Vol.SMC-6, 1976, pp.420-433.

17. Sakai, T., Nagao, M, and Matsushima,H. — Extraction of invariant picture sub-structure by Computer; Computer Graphics and Image Processing., Vol.1, 1972, pp.81-96.

18. Saltan, G and Sussenguth,E.H. — Some flexible information retrieval systems using structure matching procedures; A F I P S Conference Procedings, Vol.25, 1964, pp.587-597.

19. Seshu, S. and Reed, M.B. — Linear graphs and electrical Networks, Addison-Wesley Publishing Company, Inc., Reading, Massachussetts, 1961.

20. Ullmann, J.R. — An algorithm for subgraph isomorphism. Journal of A.C.M., Vol.23, 1976, pp.31-42.

21. Waltz, D.L. — Understanding line drawings of scenes with shadows; P.H. Winston, ed, The psychology of computer vision; McGRow-Hill, New York, 1975, pp.19-91.

****