Algorithms for Finding Isomorphic Subgraphs

by
Pradeep Kumar Giri

M. Tech (C. S.)
Roll No: MTC-9917.

under the guidance of
Prof. B. K. Roy
Applied Statistic Unit.

Indian Statistical Institute,
203-B. T. Road, KOLKATA-35

Certificate of Approval

This is to certify that the dissertation entitled *Algorithm for Finding Isomorphic Subgraphs* submitted by Pradeep Kumar Giri, towards partial fulfillment of the requirement for M. Tech. in Computer Science degree of the Indian Statistical Institute, Kolkata, is an acceptable work for the award of the degree.

Date : July 23, 2001.

(Supervisor)

(External Examiner)

ACKNOWLEDGEMENT

I gratefully acknowledge the inspiring guidance, advice, enthusiasm and criticisms of my supervisor Prof. B. K. Roy, throughout the course of this dissertation.

I express my heartiest regards and sincere thanks to my teachers for their excellent courses they have offered which helped me in this work.

Finally I would like to thank all my friends and class mates in ISI, without whose co-operation and support this work would not have been a success.

Pradeep Kumar Giri

Pradeep Kumar Giri

Contents

1 Introduction

In general subgraph isomorphism problem, given a "*text*" G and a "*pattern*" H, one must either detect an occurrence of H as a subgraph of G, or list all occurrences of H as a subgraph of G. The subgraph isomorphism problem consists in deciding for two given graphs whether one graph is isomorphic to a subgraph of other, *i.e.*, whether there is a bijective mapping from the vertex set of one graph to a subset of vertex set of second graph such that the edge connections are preserved. This problem is of considerable practical, as well as theoretical, importance. Theoretically, subgraph isomorphism is a common generalization of many important graph problems including finding Hamiltonian paths, cliques, matchings, girth, and shortest paths [9]. Variations of subgraph isomorphism problem have been used to model various practical problems such as molecular structure comparison [1], integrated circuit testing [5], microprogrammed controller optimization [12], analysis of Chinese ideographs [13], robot motion planning [14], semantic network retrieval [15], and polyhedral object recognition [18]. One of the possible applications of subgraph isomorphism is, for given the structural formulas of chemical compounds, to finding whether a chemical compound is a subcompound of another compound [7]. Subgraph isomorphism may be useful in scene analysis for detecting a relationally described object that is embedded in a scene [2,17].

In this report we are mainly concentrated on the occurrences of a graph in another graph. In section 2 we have mentioned some results for different class of graphs which are deciding atleast one occurrence of a graph as a subgraph of another graph. But the number of occurrence may be more than one. In section 3 we have considered the problem of distinct occurrences a graph in another graph and proved that both decision and optimal version of this problem is NP-complete. An algorithm, for finding all but not distinct occurrences of a graph as a subgraph of another graph, is discussed in section 4.

2 Review Works

For any two given graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, we say they are isomorphic if there is a bijection $f : V_G \rightarrow V_H$ such that $(u, v) \in E_G$ if and only if $(f(u), f(v)) \in E_H$. The problem, for deciding whether two graphs are isomorphic is not known to be NP-complete nor known to be in P [8]. However, to deciding whether a graph is a subgraph of another graph can be proved NP-complete.

For a given positive integer k, a graph G is said to be k-regular if the degree of each vertex of G is equal to k. Note that an 1-regular graph is the collection of vertex disjoint edges and 2-regular graph consists of a collection of vertex disjoint cycles. In polynomial time one can decide whether a graph contains an 1-regular graph as well as a 2-regular graph. But the problem is not same when $k > 2$. A 3-regular graph is called a cubic graph. The result of the problem that decides whether a graph contains a cubic subgraph is NP-complete was attributed to Chvàtal in [11]. The problem of deciding whether an arbitrary graph has a k-regular subgraph, for some given $k \geq 3$, is NP-complete via logspace reduction [19].

A graph is said to be planar if it can be drawn on a surface such that no two edges intersect. If the graph H is either K_3 or K_4 then there can be atmost $O(|V_G|)$ instances of H as a subgraph of G, and that these instances can be listed in linear time [3]. Any *wheel* graph of a given fixed size can be detected in a planar graph, in linear time [9]. A characterization of graphs that occurring $O(n)$ times as subgraphs of planar graphs is that they are the 3-connected planar graphs [10]. The special cases of finding $C_3 = K_3$ and $C_4 = K_{2,2}$ in planar graphs can be performed in linear time [6]. Richards [16] gives $O(nlogn)$ algorithms for finding C_5 and C_6 subgraphs.

The regular subgraph of degree 4 and 5 are called quartic and quintic subgraphs, respectively. The problems deciding whether a planar graph has a quartic subgraph, and whether a planar graph has a quintic subgraph, are NP-complete via logspace reduction [19].

3 NP-completeness

3.1 Introduction

In graph isomorphism problem (**GIso**), given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, it is required to determine whether they are isomorphic, *i.e.*, whether there is a bijection $f : V_G \rightarrow V_H$ such that $(u, v) \in E_G$ if and only if $(f(u), f(v)) \in E_H$. On the other hand, the subgraph isomorphism problem (**SGI**) asks whether the given graph H is isomorphic to a subgraph of G. It is quite clear that when H is a graph with l vertices, both the enumeration and decision problem can be solved in polynomial $O(n^l)$ time. But it is not easy to decide when H is an arbitrary graph. We shall denote an instance of this problem as a pair $\langle G, H \rangle$ of graphs. In this section we have discussed three problems, including **SGI**, related to subgraph isomorphism and show that these are NP-complete.

3.2 Subgraph Isomorphism problem

Given two undirected graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the subgraph isomorphism problem is to determine whether G contains a subgraph isomorphic to H, *i.e.*, to determine whether there exists a subset $V \subseteq V_G$ and a subset $E \subseteq E_G$ such that $|V| = |V_H|$ and $|E| = |E_H|$, and there exists an injective function $f : V_H \rightarrow V$ satisfying $(u, v) \in E_H$ if and only if $(f(u), f(v)) \in E$.

The related decision problem, SGI, for the subgraph-isomorphism is

$$\textbf{SGI} = \{\langle G, H \rangle : G \text{ contains a subgraph isomorphic to } H\}.$$

The following lemma shows that **SGI** is in class NP.

Lemma 3.1 **SGI** *is in class* NP.

Proof Given an instance $\langle G, H \rangle$ of **SGI**and a certificate (V, E, f), we have to verify whether the following conditions are satisfied:

i. $V \subseteq V_G$ and $|V| = |V_H|$.

ii. $E \subseteq E_G$ and $|E| = |E_H|$.

iii. $(u, v) \in E_H \Rightarrow (f(u), f(v)) \in E$.

iv. f is injective.

For any two given sets A and B, it can be verified in $O(|A| + |B|)$ time whether $A \subseteq B$ and $|A| = |B|$. Thus the conditions (i) and (ii) can be verified in polynomial time.

For given $(u, v) \in E_H$, to verify whether $(f(u), f(v)) \in E$, we need only one comparison. Since $|E_H| \leq O(|V_H|^2)$, the condition (iii) can be verified in polynomial time.

To verify whether $f : V_H \rightarrow V$ is injective we will take the elements of V in an array **V** and can verify in polynomial time using the following algorithm.

```
begin
    for every x∈V_H do
        search the position of f(x) in V;
        if the position is not marked then mark this position;
        else give output "f is not injective";
    end for;
    give output "f is injective";
end;
```

Thus all conditions can be verified in polynomial time, and hence **SGI** is in class NP. ∎

Given a graph $G = (V, E)$ and a positive integer k, the clique problem is to determine whether G contains a clique of size k, *i.e.*, to determine whether there exists a subset $V' \subseteq V$ such that $|V'| = k$ and any two vertices u, v in V' are adjacent to each other in G. The related decision problem, **CLIQUE**, for the clique detection is

$$\textbf{CLIQUE} = \{\langle G, k\rangle : G \text{ contains a clique of size } k\}.$$

It is known that **CLIQUE** is NP-complete[]. The following lemma uses a reduction from **CLIQUE** to show **SGI** is NP-hard.

Lemma 3.2 **SGI** *is* NP*-hard.*

Proof Given an instance $\langle G, k\rangle$ of **CLIQUE**, construct an instance $\langle G', H'\rangle$ of **SGI** as follows:

i. G' is a copy of G.

ii. $H' = K_k$, a complete graph with k vertices.

Clearly the construction takes polynomial time. To complete the proof, we show that the graph G has a clique of size of k if and only if the graph G' has a subgraph isomorphic to K_k. But it is trivial, since G' is à copy of G. This complete the proof. ∎

Summarising the above two lemmas we have the following result.

Theorem 3.1 **SGI** *is NP-comlete.*

3.3 Subgraph Occurence Problem

One may be interested to find the number of distinct occurances of a given graph H in another given graph G. Given two undirected graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, and a positive integer p, the subgraph occurance problem is to determine whether G contains p number of subgraphs isomorphic to H, such that no two subgraphs have identical vertex set, *i.e.*, to determine whether there exist p distnct subsets $V_1, \cdots, V_p$ of the vertex set V_G and p distinct subsets $E_1, \cdots, E_p$ of the edge set E_G of G, and p injective functions $f_1 : V_H \rightarrow V_1, \cdots, f_p : V_H \rightarrow V_p$, such that $|V_i| = |V_H|$, $|E_i| = |E_H|$, and $(u, v) \in E_H \Rightarrow (f(u), f(v)) \in E_i)$, for $1 \leq i \leq p$ and the subsets $V_1, \cdots, V_p$ are pairwise distinct. The related decision problem, **SGO**, subgraph occurance problem is

SGO $= \{\langle G, H\rangle : G$ contains atleast p distinct subgraphs isomorphic to $H\}$.

An instance of this decision problem can be denoted as $\langle G, H, p\rangle$, where G and H are two undirected graphs, and p is a positive integer. The lemma given bellow shows that this problem is in class NP.

Lemma 3.3 **SGO** *is in class NP.*

Proof Let $\langle G, H, p\rangle$ be a given instance of **SGO**. Then for a given certificate $(V_1, \cdots, V_p; E_1, \cdots, E_p; f_1, \cdots, f_p)$, it is to verify whether the following conditions are satisfied:

i. for $i = 1, \cdots, p$

   (a) $V_i \subseteq V_G$ and $|V_i| = |V_H|$.

   (b) $E_i \subseteq E_G$ and $|E_i| = |E_H|$.

   (c) $(u, v) \in E_H \Rightarrow (f_i(u), f_i(v)) \in E_i$.

(d) $f_i : V_{II} \rightarrow V_i$ is injective.

ii. the sets $V_1, \cdots, V_p$ are pairwise distinct.

For each $i = 1, \cdots, p$, the verification of condition (i) is similar to that we have used to verify a certificate in SGI. Thus the condition (i) can be verified in polynomial time.

For any two sets A and B with $|A| = |B| = n$ one can easily check that A and B are distinct if and only if $|A \cap B| < n$. Thus, the verification of two sets whether they are distnct, can be done in polynomial time by using following algorithm:

```
begin
    int Card := 0;
    for every x∈A do
        search x in B;
        if x is in B, then increase Card by 1;
    end for;
    if Card < n then give output "A and B are distinct."
    else give output "A and B are identical."
end;
```

The searching for an element in B takes $O(n)$ time. Since $|A| = n$, the above algorithm will take $O(n^2)$ time. In order to prove the sets $V_1, \cdots, V_p$ are pairwise distinct we have to run the above algorithm for every pair of sets V_i and V_j. Since the number of pairs of sets is $\binom{p}{2} = \frac{1}{2}p(p-1)$, the total time required for verifying the condition (ii) is $O(p^2n^2)$, a polynomial time.

This completes the proof. ∎

Now we will show that **SGO** is NP-hard by using a reduction from SGI.

Lemma 3.4 **SGO** *is NP-hard.*

proof : Let $\langle G, H \rangle$ be an instance of **SGI**. A reduction algorithm which reduces the instance $\langle G, H \rangle$ to an instance $\langle G', H', p \rangle$ of **SGO** constructs the graphs G' and H', and integer p as follows:

i. G' is a copy of G.

ii. H' is a copy of H.

iii. $p = 1$.

Since we can construct a copy of a graph in polynomial time this reduction will take polynomial time to construct an instance of **SGO** from any instance of **SGI**, and hence this reduction is a polynomial time reduction.

On the other hand, if $\langle G', H', 1\rangle$ is an instance of **SGO** corresponding to the instance $\langle G, H\rangle$ of **SGI**, it is trivial that G contains a subgraph isomorphic to H if and only if G' contains atleast 1 subgraphs isomorphic to H', because G' is a copy of G and H' is a copy of H.

This completes the proof. ∎

In last two lemmas we have already proved that **SGO** is in the class NP and is also NP-hard, hence the following result:

Theorem 3.2 SGO *is NP-complete.*

3.4 Maximum Subgraph Occurance problem

To find all distinct occurrence of a graph H in another graph G is an optimal problem. So here we may define the maximum subgraph occurrence problem (**MSGO**) as follows:

$$\textbf{MSGO} = \{\langle G, H\rangle : \text{list all distinct subgraphs of } G \text{ isomorphic to } H\}.$$

An optimizatin problem is NP-complete if the associated decision problem is NP-complete [8]. We have already proved that **SGO** is NP-complete. Hence the following result:

Theorem 3.3 MSGO *is NP-complete.*

4 An Algorithm for Subgraph Isomorphism

4.1 Introduction

Because of reduction from Hamiltonian path and Clique finding problems, the Subgraph Isomorphism decision problem is NP-complete. So subexponential algorithms are unlikely. It is well known that isomorphism can be determined by brute-force enumeration. In this section we will discuss an algorithm which determines subgraph isomorphoism by means of a brute-force tree-search enumeration procedure and another algorithm is also introduced that attains efficiency by inferentially eliminating successor nodes in the tree search. In section 2 we will discuss the original part of the algorithm which describes a brute-force enumeration procedure that is actually a depth-first tree-search algorithm. Section 3 consists of a procedure that is entered that each node in the tree search. This procedure is a reduction procedure which reduces the number of successor nodes that must be searched, and hence reduces the total computer time required for determining isomorphism.

A graph $G = (V, E)$ is an ordered pair consisting of a finite set of vertices V and a finite set of edges E. Here we have considered only undirected graphs. We will denote the degree of ith vertex of the graph G by the symbol $d_G(i)$.

4.2 A Simple Enumeration Algorithm for Subgraph Isomorphism

In this section a simple tree-search algorithm is formulated to find all of the isomorphisms between a graph $H = (V_H, E_H)$ and subgraphs of another given graph $G = (V_G, E_G)$. The number of vertices of H and G are m and n, respectively. The adjacency matrix of H and G are $A = [a_{ij}]$ and $B = [b_{ij}]$, respectively.

To get a vertex induced subgraph of G with m vertices, we define a matrix M to be a $m \times n$ matrix whose entries are 1's and 0's, such that each row contains exactly one 1 and each column contains atmost one 1. This matrix can be used to permute the rows and columns of B to produce another matrix C, the adjacency matrix of a vertex induced subgraph H' (say) of G. Specifically, we define $C = [c_{ij}] = MBM^T$, where T denotes transposition of a matrix. Thus ith vertex of the subgraph H' will correspond to the jth

vertex of G iff $m_{ij} = 1$. Since B is symetric and hence

$$C^T = (MBM^T)^T = (M^T)^T B^T M^T = MBM^T = C;$$

the matrix C is also symetric. So M gives an isomorphism between H and an edge induced subgraph of G if

$$(\forall \ \ 1 \leq i < j \leq m) \ \ ((a_{ij} = 1) \ \ \Rightarrow \ \ (c_{ij} = 1)). \tag{1}$$

Also M will give an isomorphism between H and a vertex induced subgraph of G if

$$(\forall \ \ 1 \leq i < j \leq m) \ \ ((a_{ij} = 1) \ \ \Leftrightarrow \ \ (c_{ij} = 1)). \tag{2}$$

In both cases, if $m_{ij} = 1$, then the ith vertex of H corresponds to jth vertex of G in this isomorphism. If we had any a priori reason to be sure that the ith vertex of H could not correspond to the jth vertex of G in any subgraph isomorphism, then the matrix M with $m_{ij} = 1$ will not give an isomorphism and hence, in this case, we always consider the matrices M with $m_{ij} = 0$ for these particular values of the indices i and j. One of the a priori reason, the ith vertex of H could not correspond to jth vertex of G in any isomorphism, is that when $d_H(i) > d_G(j)$. So at the start of the enumeration algorithm, we construct a $m \times n$ element matrix $W = [w_{ij}]$ as follows:

$$w_{ij} = \begin{cases} 1 & \text{,when } d_H(i) \leq d_G(j), \\ 0 & \text{,otherwise.} \end{cases}$$

The enumeration algorithm works by generating all possible matrices M such that for each and every element m_{ij} of M, $(m_{ij} = 1) \Rightarrow (w_{ij} = 1)$. For each such matrix M the algorithm tests for isomorphism by applying condition (1). Matrices M are generated by systematically changing to 0 all but one of 1's in each of the rows of W, subject to the condition that each column of a matrix M may contain atmost one 1.In the search tree, the terminal nodes are at depth $d = m$ and they correspond to distinct matrix M. Each nonterminal node at depth $d < m$ corresponds to a distinct matrix M' which differs from the matrix W in first d rows, and in each row of these d rows all elements but one of the 1's has been changed to 0.

The algorithm uses an n-bit binary vector $(F_1, \cdots, F_j, \cdots, F_n)$ to record which columns have been used at an intermediate state of the computation by setting $F_i = 1$ if the ith column has been used. The algorithm also used another vector $(R_1, \cdots, R_d, \cdots, R_m)$ of m integers to record which column

has been selected at which depth by assigning the integer k to R_d if the kth column has been selected at depth d.

Let M be a matrix corresponding to a terminal node. Then each row of M contains exactly one 1 and each column of it contains atmost one 1. In order to testing the condition (1) for the matrix M it is necessary to compute the matrix C, which is the product of three matrices. Without computing the matrix multiplication we can compute matrix C by using the information stored in the vector $(F_1, \cdots, F_j, \cdots, F_n)$. Note that for $1 \leq i \leq m$ and $1 \leq k \leq n$, we have $m_{ik} = 1$ iff $R_i = k$. Let $X = [x_{ij}]$ be the product of B and M^T e.i. $X = BM^T$. Then for $1 \leq i, j \leq m$,

$$\begin{aligned} c_{ij} &= \sum_{k=1}^{n} m_{ik} x_{kj} \\ &= m_{iR_i} x_{R_i j} \\ &= x_{R_i j} \\ &= \sum_{l=1}^{m} b_{R_i l} m_{jl} \\ &= b_{R_i R_j} m_{jR_j} \\ &= b_{R_i R_j} \end{aligned}$$

i.e. $c_{ij} = b_{R_i R_j}, \forall 1 \leq i, j \leq m$. So the condition (1) reduced to the following condition

$$(\forall \ \ 1 \leq i < j \leq m) \ \ ((a_{ij} = 1) \ \ \Rightarrow \ \ (b_{R_i R_j} = 1)). \tag{3}$$

and condition (2) reduced to the following condition

$$(\forall \ \ 1 \leq i < j \leq m) \ \ ((a_{ij} = 1) \ \ \Leftrightarrow \ \ (b_{R_i R_j} = 1)). \tag{4}$$

When this condition is true, one isomorphism has been found, where the ith vertex of H will correspond to R_ith vertex of G, for $i = 1, \cdots, m$.

For the algorithm we shall use the symbol := to denote assignment. Thus $d := d + 1$ means set d equal to $d + 1$. The simple enumeration algorithm is given below:

Step 1 : $d := 1; R_1 := 0;$
for all $i := 1, \cdots, n$, set $F_i = 0;$
Step 2 : If there is no value of j such that $1 \leq j \leq n$,
$w_{dj} = 1$, and $F_j = 0$ then goto Step 7;
$k := 0;$
Step 3 : $k := k + 1;$
If $w_{dk} = 0$ or $F_k = 1$ then goto Step 3;
$R_d := k;$
Step 4 : If $d < m$ then goto Step 6;
else use condition (3) or (4) and give output if an isomorphism is found;
Step 5 : If there is no j such that $k < j \leq n$, $w_{dj} = 1$
and $F_j = 0$ then goto Step 7;
else goto Step 3;
Step 6 : $F_k := 1; d := d + 1;$
goto Step 2;
Step 7 : If $d = 1$ then terminate algorithm;
$F_k := 0; d := d - 1; k := R_d;$
goto Step 5;

The figure-1 is the flow chart of this algorithm.

4.3 Algorithm Employing Refinement Procedure

To reduce the amount of computation required for finding subgraph isomorphism we employ a procedure, called refinement procedure, that does not select some 1's from the matrix M', thus eliminating the computation for successor nodes in the tree search.

Suppose that the matrix $M' = [m'_{ij}]$ corresponding to a nonterminal node in the search tree at depth d. Then M' differs from W in the first d rows, where in each row all elements but one of the 1's has been changed to 0. We say that an isomorphism is an isomorphism under M' if its terminal node in the search tree is a successor of the node with which M' is associated. If $m_{ij} = 0$ for all isomorphism under M', then if $m'_{ij} = 1$ we can change $m'_{ij} = 1$ to $m'_{ij} = 0$ without losing any of the isomorphism under M'. Now we will work out a condition that satisfied necessarily if $m_{ij} = 1$ for any isomorphism under M'. If this necessary condition is not satisfied and $m'_{ij} = 1$, then the refinement procedure changes $m'_{ij} = 1$ to $m'_{ij} = 0$.

Let v_i be the ith vertex in V_H, and u_j be the jth vertex in V_G. Let $\{v_{i_1}, v_{i_2}, \cdots, v_{i_\alpha}\}$ be the set of all vertices of H that are adjacent to v_i in H and $\{u_{j_1}, u_{j_2}, \cdots, u_{j_\beta}\}$ be the set of all vertices of G that are adjacent to u_j in G. Let us consider a matrix M corresponding to an isomorphism under M'. If v_i corresponds to u_j in this isomorphism then by definition of subgraph isomorphism it is necessary that for each $l = 1, 2, \cdots, \alpha$ there must exist a vertex u_{j_k} in V_G that is adjacent to u_j, such that u_{j_k} corresponds to v_{i_l} in the isomorphism. Recall that if u_{j_k} corresponds to v_{i_l} in the isomorphism, then the elements of M that corresponds to $\{v_{i_l}, u_{j_k}\}$ is 1. Therefore if v_i corresponds to u_j in any isomorphism under M',then for each $l = 1, 2, \cdots, \alpha$, there must be a 1 in M' corresponding to some $\{v_{i_l}, u_{j_k}\}$ such that u_{j_k} is adjacent to u_j. In otherword, if v_i corresponds to u_j in any isomorphism under M' then

$$(\forall\ 1 \leq l \leq m)\ ((a_{il} = 1)\ \Rightarrow\ (\exists k, 1 \leq k \leq n)(m'_{lk} b_{kj} = 1)). \tag{5}$$

Since we are not computing the matrix M' corresponding to a nonterminal node in the search tree explicitly, it is essential to work out a condition equivalent to the condition (5) in terms of the data stored in the integer vector $(R_1, \cdots, R_d, \cdots, R_m)$ and in the matrix W. The following two conditions together is equivalent to condition (5).

$$(\forall\ 1 \leq l \leq d)\ ((a_{il} = 1)\ \Rightarrow\ (\exists k, 1 \leq k \leq n)(m'_{lk} b_{kj} = 1)). \tag{6}$$

$$(\forall\ d < l \leq m)\ ((a_{il} = 1)\ \Rightarrow\ (\exists k, 1 \leq k \leq n)(m'_{lk} b_{kj} = 1)). \tag{7}$$

For $1 \leq l \leq d$, if $a_{il} = 1$, then we have already selected an 1 from the lth row of W, and it is from the column R_l. Since in any isomorphism under M' the vertex v_l will corresponds to the vertex u_{R_l}, the vertex u_{R_l} must be adjacent to the vertex u_j in G. Thus the condition (6) reduced to the following condition:

$$(\forall\ 1 \leq l \leq d)\ ((a_{il} = 1)\ \Rightarrow\ (b_{R_l j} = 1)). \tag{8}$$

Again for $d < l \leq m$, since the lth row of W and M' are same the condition (7) can be written as

$$(\forall\ d < l \leq m)\ ((a_{il} = 1)\ \Rightarrow\ (\exists k, 1 \leq k \leq n)(w'_{lk} b_{kj} = 1)). \tag{9}$$

Thus the conditions (8) and (9) together equivalent to the condition (5).

Suppose we have already selected d 1's from the first d rows of W, one from each row and no two 1's from a single column. Then before selecting an

1 from the $(d+1)$th row the refinement procedure tests each 1 in each row of M' to find whether the conditions (8) and (9) are satisfied. If any 1 in the first d rows does not satisfies the conditions (8) and (9), then the procedure jumps to its FAIL exit, because there is no advantage in continuing. But for any 1 in the remaining rows if it does not satisfies the conditions (8) and (9), then it will be set to 0 in M'. To keeping track to this position we have used another matrix $W' = [w'_{ij}]$ of integers of size $m \times n$. Initially the matrix W' is as follows:

$$w'_{ij} = \begin{cases} m & \text{,when } w_{ij} = 1 \\ 0 & \text{,otherwise.} \end{cases}$$

If w'_{ij} will be set to the integer d during the execution of the refinement procedure, if its value is greater than or equal to d and its position does not satisfy the conditions (8) and (9). If any one of these rows contains no values greater than d then the procedure jumps to its FAIL exit. Otherwise the procedure terminates at its SUCCEED exit.

The corresponding algorithm of the refinement procedure for subgraph isomorphism is as follows:

Step 1 : $d := 1$; $R_1 := 0$;
for all $i := 1, \cdots, n$, set $F_i = 0$;
Refinement; if exit FAIL then terminate algorithm;

Step 2 : If there is no value of j such that $1 \leq j \leq n$,
$w'_{dj} > d$, and $F_j = 0$ then goto Step 7;
$k := 0$;

Step 3 : $k := k + 1$;
If $w'_{dk} \leq d$ or $F_k = 1$ then goto Step 3;
$R_d := k$; $F_k := 1$; $d := d + 1$;
Refinement; if exit FAIL then goto Step 5;

Step 4 : If $d \leq m$ then goto Step 2;
else give output to indicate that an
isomorphism has been found;

Step 5 : $F_k := 0$; $d := d - 1$; $k := R_d$;
If there is no j such that $k < j \leq n$ and $w_{dj} = 1$
and $F_j = 0$ then goto Step 7;
else goto Step 3;

Step 7 : If $d = 1$ then terminate algorithm;
else goto Step 5;

The algorithm of the refinement procedure, before searching an 1 from$d$th

row, is as follows:

```
begin
        for i = 1, ···, d − 1 do
            j := H[i];
            for k = 1, ···, d − 1 do
                if a_ik = 1 and b_jH[k] = 0 then return FAIL;
        for k = d, ···, m do
            if a_ik = 1
                for l = 1, ···, n do
                    if w'_ik ≥ d and F_l = 0, then break;
                if l > n, then exit FAIL;
    for i = d, ···, m do
        for j = 1, ···, n do
            if w'_ij ≥ d
                for k = 1, ···, d − 1 do
                    if a_ik = 1 and b_Hkj = 0, then w'_ij = d;
                for k = d, ···, m do
                    if a_ik = 1
                        for l = 1, ···, n do
                            if w'_kl > d and F_l = 0, then break;
                        if l > n, then w'_ij = d;
            for j = 0, ···, n do
                if w'_ij ≥ d, then break;
            if l > n, then exit FAIL;
end;
```

We have implemented this algorithm in C, and tested the program for some special graphs whether they are isomorphic to subgraphs of a second graph. For a given special graph H program generate 50 graphs of fixed number of vertices randomly using a pseudo-random number generator to construct the adjacency matrices. The program computes the average number of isomorphisms found, and the average time required to find all isomorphisms. The outputs for K_5, $K_{3,3}$, and $K_{3,3,3}$ are given in the next page. In this experiment we have consider the probability of occurrence of an edge in K_5, $K_{3,3}$, and $K_{3,3,3}$ is 30, 30, and 50 respectively.

TABLE I. Result of Experiments with K_5 Detection.

No. of vertices	Average No. of isomorphisms	Average Time in seconds
20	9.6	0.97
22	9.6	1.33
24	24.00	1.82
26	50.40	2.44
28	50.40	3.08
30	76.80	3.92

TABLE II. Result of Experiments with $K_{3,3}$ Detection.

No. of vertices	Average No. of isomorphisms	Average Time in seconds
10	4.32	0.86
12	10.08	1.85
14	40.32	3.55
16	112.32	6.46
18	264.96	10.67
20	411.84	16.40

TABLE III. Result of Experiments with $K_{3,3,3}$ Detection.

No. of vertices	Average No. of isomorphisms	Average Time in seconds
17	311.04	34.73
18	77.76	49.43
19	362.88	72.15
20	570.24	104.92
21	1192.32	125.17
22	1399.68	162.29
23	3991.68	215.56

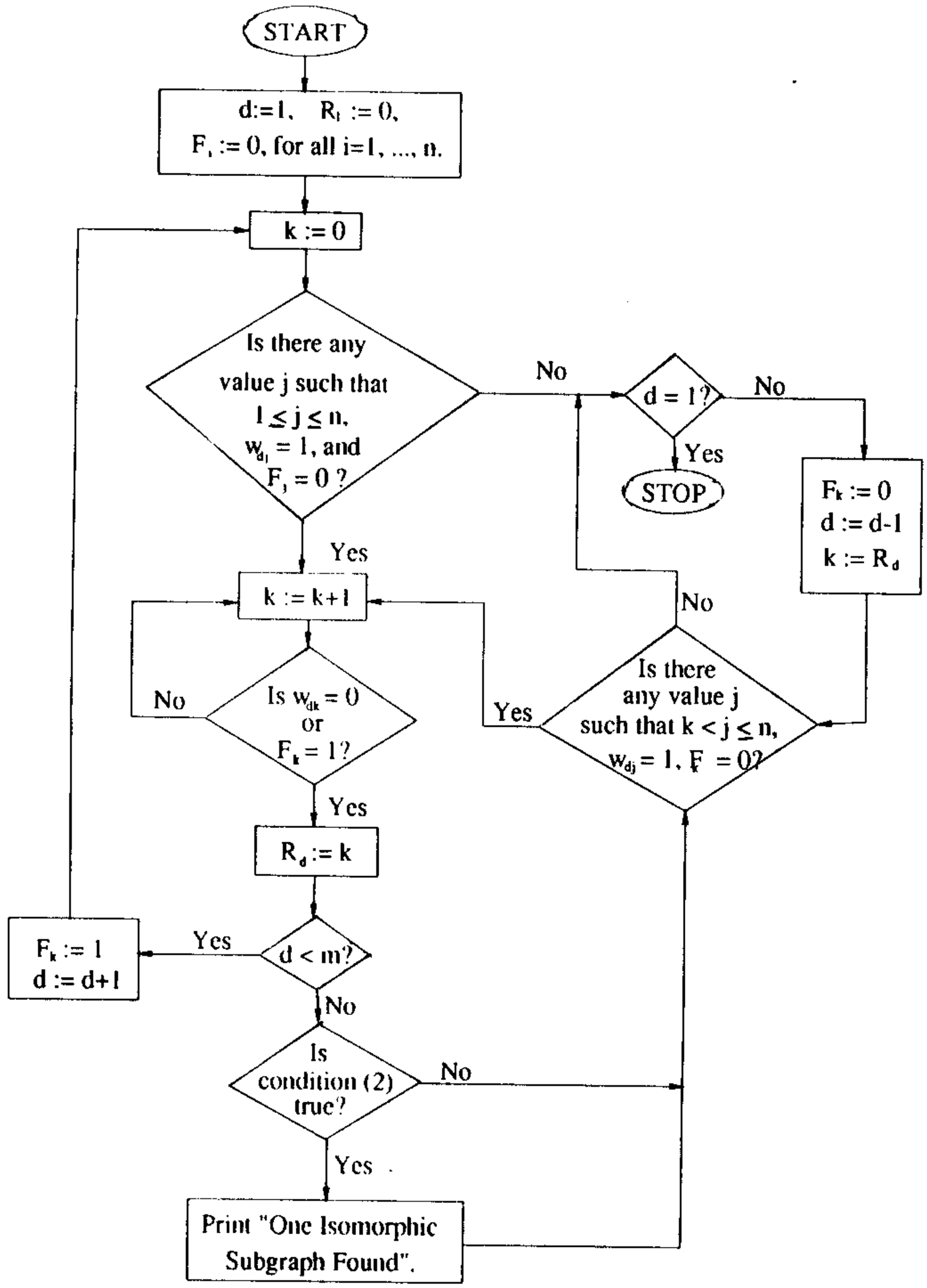
START
d:=1, R_i := 0,
F_i := 0, for all i=1, ..., n.
k := 0
Is there any value j such that $1 \le j \le n$, $w_{dj} = 1$, and $F_j = 0$?
No
d = 1?
No
Yes
STOP
$F_k := 0$
d := d-1
$k := R_d$
Yes
k := k+1
No
Is there any value j such that $k < j \le n$, $w_{dj} = 1$, $F_j = 0$?
Yes
No
Is $w_{dk} = 0$ or $F_k = 1$?
Yes
$R_d := k$
$F_k := 1$
d := d+1
Yes
d < m?
No
Is condition (2) true?
No
Yes
Print "One Isomorphic Subgraph Found".

Figure 1:

Bibliography

[1] Artymiuk, P. J., Bath, Grindley, H. M., Pepperrell, P. A., Poirrette, A. R., Rice, D. W., Thorner, D. A., Wild, D. J., Willet, P., Allen, F. H., and Taylor, R., *Similarity searching in databases of three-dimensional molecules and macromolecules*, J. Chemical Information and Computer Sciences, Vol. 32, pp. 617-630, 1992.

[2] Barrow, H. G., Ambler, A. P., and Burstall, R. M., *Some techniques for recognizing structures in pictures*, In Frontiers of Pattern recognition, S. Watanabe, Ed., Academic Press, New York, pp. 1-29, 1972.

[3] Bar-Yehuda, R. and Even, S., *On approximating a vertex cover for planar graphs*. In Proc. 14th ACM Symp. Theory of Computing, pp. 303-309, 1982.

[4] Bron, C., Kerbosch, J., *Finding All Cliques of an Undirected Graph[H]*, Communications of the ACM, Vol. 16, No. 9, pp. 575-577, September 1973.

[5] Brown, A. D., Thomas, P. R., *Goal-oriented subgraph isomorphism technique for IC device recognition*, IEE Proceedings I (Solid-State and Electron Devices), Vol. 135, pp. 141-223, 1988.

[6] Chiba, N. and Nishizeki, T., *Arboricity and subgraph listing algorithms*, SIAM J. Computing, Vol. 14, pp 210-223, 1985.

[7] Corneil, D. G.,Gotlieb, C. C., *An Efficient Algorithm for Graph Isomorphism*, Journal of the Association for Computing Mechinary, Vol. 17, No. 1, pp. 51-64, January 1970.

[8] Du, D. Z., and Ko, K., *Theory of Computational Complexity*, John Wiley & Sons, INC, New York, 2000.

[9] Eppstein, D., *Subgraph Isomorphism in Planar Graphs and Related Problems*, J. Graph Algorithms and Applications, Vol. 3, No. 3, pp. 1-27, 1999.

[10] Eppstein, D., *Connectivity, graph minors, and subgraph multiplicity*, J. Graph Theory, Vol. 17, pp. 409-416, 1993.

[11] Garey, M. R. and Johnson, D. S., *Computers and Intractability: a guide to the Theory of NP-completeness*. W. H. Freeman and Co., San Francisco, 1979.

[12] Guha, A., *Optimizing codes for concurrent fault detection in microprogrammed controllers*. Proc. IEEE Intl. Conf. Computer Design: VLSI in Computers and Processors (ICCD '87), pp. 486-489, 1987.

[13] Hong, D., Youshou, W., and Xiaoqiag, D., *An ARG representation for Chinese characters and a radical extraction based on the representation.* In 9th IEEE Intl. Conf. Pattern Recognition, Vol. 2, pp 920-922, 1988.

[14] Lang, S. Y. T., and Wong, A. K. S., *A sensor model registrations technique for mobile robert localization*, In Proc. 1991 IEEE Intl. Symp. Intelligent control, pp. 298-305, 1991.

[15] Levinson, R., *Pattern associativity and the retrieval of semantic networks*, Computers and Mathematics with Applications, Vol. 23, pp. 573-600, 1992.

[16] Richards, D., *Finding short cycles in planar graphs using separators*, J. Algorithms, Vol. 7, pp. 382-394, 1986.

[17] Sakai, T., Nagao, M., and Matsushima, H., *Extraction of invariant picture substructures by computer*, Computer Graphics and Image Process 1, Vol. 1, pp. 81-96, April 1972.

[18] Stahs, T. and Wahl, F., *Recognition of polyhedral objects under perspective views*, Computers and Artificial Intelligence, Vol. 11, pp. 155-172, 1992.

[19] Stewart, I. A., *Finding Regular Subgraphs in both Arbitrary and Planar Graphs*, Discrete Applied Mathematics, Vol. 68, pp. 223-235, 1996.

[20] Ullmann, J. R., *An Algorithm for Subgraph Isomorphism*, Journal of the Association for Computing Mechinary, Vol. 23, No. 1, pp. 31-42, January 1976.