Graph Invariants and Graph Isomorphism

Vijaya Balpande
Asst.Prof.: Computer Science and Engg.
J.L.Chaturvedi College of Engineering
Research Centre:-G.H.Raisoni College of Engineering

Dr. Anjali R Mahajan
Prof: Computer Science and Engg.
Priyadarshini Institute of Engineering and Technology
Nagpur, India

Abstract—In graph theory, Graph Isomorphism is an important issue. Information in the database can be stored in the form of graph. Graph represents the structural information in an efficient way. Graph Isomorphism problem is to determine if there exists one to one correspondence between the structures of two graphs. Graph isomorphism problem arises in many fields such as chemistry, switching theory, information retrieval, social networks, etc. Graph Invariants are used to determine the isomorphism between two graphs. Graph Invariant is only the necessary condition for graph Isomorphism. Most of the researchers believe that isomorphism problem is NP complete problem and the most difficult problem. Graph isomorphism is a NP complete or not is always a hot issue for the researchers to study. In this paper we discuss the graph invariants and graph isomorphism techniques.

Keywords- Graph Isomorphism, Subgraph Isomorphism, and Graph Invariant

I. INTRODUCTION

With the advent of multicore processors the burden of improving program execution speed has shifted to software developers from chip manufacturers. Mainly challenging problem in this perspective is the parallelization of irregular applications that deal with complex, pointer-based data structures such as trees, queues and graphs. It is difficult to parallelize manually the pointer-based data structure like graph.

Graphs are widely used in several areas for representing many different kinds of information: syntactic structures of a language, chemical compounds, geographic maps, computer networks, software systems architectures, database structures, etc. Many of these applications require some type of comparison between graphs, which can be formulated as a graph matching problem. In particular, in several important cases, the application entails the search for graph isomorphism or for graph–sub-graph isomorphism.

Graph isomorphism is a highly-studied problem in computer science. It is strongly suspected not to be NP-complete, yet no polynomial-time algorithm for it has been found despite much effort. Several subproblems of graph isomorphism are known to have polynomial algorithms.

Theoretically, we can determine whether or not two graphs G1 and G2 are isomorphic by keeping G1 fixed and reordering vertices of G2 to check if their adjacency matrices are identical. But this requires reordering and comparison nearly equal to n! where n is the number of vertices. Practically this is very inefficient as the running time grows factorially with n. An algorithm which gives the solution in running time proportional to constant power of n is desirable but no such algorithm has been discovered for determining if two graphs are isomorphic. Graph isomorphism problem is solved if we can find the complete set of invariants of G1 and G2. The problem of finding a complete set of invariants is known as coding of a graph. We can retrieve essential information by placing the complete set of invariants in sequence. Two graphs are isomorphic if and only if their codes are same.

Graph Invariant is the property of graph. Graph invariant is a number that is same for all graphs that is isomorphic to graph G. The graph invariants are number of vertices, number of edges, number of components, connectivity, coefficients in the characteristic polynomial of adjacency matrix. A complete set of invariants is a set of invariants that completely describes a graph within isomorphism [1]. Invariants are necessary condition for graph isomorphism. Graph isomorphism expresses a basic structural similarity between two graphs.

II. DEFINITIONS

A. Definition1:

The graph isomorphism is expressed as: Given two graphs G1=(V1,E1) and G2=(V2,E2), if there exist 1 to 1 mapping function f from V1 to V2 such that (i, j) ∈ E1, if and only if (f(i), f(j)) ∈ E2. The function f is called an isomorphism from G1 to G2. If the two graphs isomorphic to each other, it is denoted by G1 ≅ G2.

The problem is non polynomial, but it is not known to be polynomial and therefore not known to be in NC.

A Graph G = (V, E) consist of set of objects V = {v1, v2, ...} called vertices and E = {e1, e2, ...} called edges. An edge ek is identified with an unordered pair (vk, vj) of vertices and vi, vj are called end vertices of ek.

In a graph G the degree or valence (vi) of a vertex v is defined as the number of edges adjacent with v, loops being counted twice. The cardinality of edge E is denoted by |E| which defines the size of graph and cardinality of vertex |V| defines the order of graph.
B. Definition2:

A graph $G (V,E)$ is represented by adjacency matrix $A=(a_{ij})$ of order $n \times n$ where $V$ is a set of vertices $\{v_1, v_2, \ldots\}$

$$
a_{ij} = \begin{cases} 
[|v_i, v_j|], [v_i, v_j] \in E, & i \neq j \\
2[|v_i, v_j|], [v_i, v_j] \in E, & i, j=1, 2, \ldots \\
0, & [v_i, v_j] \notin E 
\end{cases} \quad (1)
$$

The degree of vertex $v_i$ is the sum of all 1’s in a row (column) of adjacency matrix corresponding to vertex $v_i$ and is denoted by

$$
\text{Deg}(v_i) = \sum_{j=1}^{n} a_{ij}, \quad i, j = 1, 2, \ldots, n \quad (2)
$$

For the mathematical proof of other property such as dual congruent, finding Jordon canonical explained in [2008].

C. Definition3:

The identity matrix $M$ of order $n \times n$ is represented as

$$
m_{ij} = \begin{cases} 
1, & \text{if } i = j \\
0, & \text{otherwise} 
\end{cases} \quad (3)
$$

Where $m_i$ is an element of $M$ on the $i^{th}$ row and $j^{th}$ column.

A permutation matrix is obtained from the identity matrix by any row and column permutation. If $M_1$ and $M_2$ are the two matrices of graph $G_1$ and $G_2$, respectively $M_1$ is said to be isomorphic to $M_2$ if there exists 1 to 1 mapping of function $f$ from the rows of $M_1$ to rows of $M_2$ and from column of $M_1$ to column of $M_2$. The function $f$ is called an isomorphism from $M_1$ to $M_2$. In other words, $M_1$ is isomorphic to $M_2$ if and only if there exists the permutation matrices $P_1$ and $P_2$ satisfying the following relation

$$
M_1 = P_1 M_2 P_2 \quad (4)
$$

III. GRAPH ISOMORPHISM

Researchers have done a lot of work on graph isomorphism and subgraph isomorphism. There are two ways to detect graph isomorphism and subgraph isomorphism. The first approach is given two unknown graphs, check whether the two graphs are isomorphic or if one graph is a subgraph of other. The second approach for graph/subgraph isomorphism detection is given a database of graph called model graph and an input graph to be tested. Subgraph isomorphism has been proved to be NP complete problem but there is no polynomial algorithm for graph isomorphism detection.

There are two types of Graph matching: Exact Matching and Inexact Matching. Exact graph matching is characterize by the fact that the mapping between the nodes of the two graphs must be edge-preserving in the sense that if there is an edge between the two nodes in the first graph, they are mapped to two nodes in the second graph that are linked by an edge as well. The different form of exact matching is graph isomorphism, subgraph isomorphism, monomorphism or Automorphism and Maximum common subgraph (MCS). For exact matching the different techniques exist like tree search based algorithms and canonical labeling. Ullmann Algorithm [2], B.T.Messmer and H.Bunke [3], D.G.Corneil and Gotelib [4] and Nauty Algorithm [5][6] are the algorithm based on tree search method and canonical labeling. In [5][6] algorithm based on canonical labeling is described. It is the practically available algorithm on site of the author specified in[6]. The other method for graph isomorphism is Indexing Method in [11][12].

In [3], the concept of decision tree is introduced to test graph /subgraph isomorphism. By performing the same transformation and merge operation on the model graph, the graph is organized as decision tree. The isomorphism detection algorithm has the polynomial time complexity in the number of input vertices if the constructing time for the decision tree is neglected. Space complexity for the algorithm [3] is high as the size of the decision tree increases exponentially with the number of vertices. The two pruning technique are used which reduces the size of the decision tree to be searched. The pruning technique does not guarantee the graph isomorphism detection in the polynomial time. The pruning method usually deletes the least reliable branch by statistical methods to improve the speed of classification and recognition and the capacity of accurate data classification. The essence is to eliminate isolated points. The algorithm the two graphs are isomorphic or if one graph is a subgraph of other. The second approach for graph /subgraph isomorphism detection is given a database of graph called model graph and supports both the graph isomorphism and subgraph isomorphism. Ullmann algorithm [2] follows the depth-first search technique. It is based on concept of backtracking. It is the most popular graph matching algorithm. It has simple enumeration algorithm for the isomorphism between a graph $G$ and a subgraph of another graph $H$ with the adjacency matrices $A_G$ and $A_H$. In this algorithm the matrix $M^0$ is constructed with number of rows as the number of vertices in graph $G$ and number of columns as number of vertices of Graph $H$. The matrix $M^0$ is a $(0-1)$ matrix constructed based on the following condition

$$
m_{ij} = \begin{cases} 
1, & \text{if degree } (V_H) \geq d(V_G) \\
0, & \text{otherwise} 
\end{cases} \quad (5)
$$

The matrix $M^0$ is used to permute the rows and columns of matrix of $AH$ to produce a further matrix $P$. All $M^0$ matrices are created by setting 1 of each row and column depending on the condition such that the elements of $M'$ are 1’s and 0’s such that
each row contains 1 and each column contains 0 or 1. The matrix $M'$ specifies an isomorphism between graph $G$ and $H$. If $P$ is same as $A_G$ then the graph $G$ is subgraph of $H$.

$$P = M' (M'A_{G})^T$$  \hspace{1cm} (6)

The pruning technique is applied by reducing the number of $1$’s thus reducing the search space. The algorithm is used to detect graph isomorphism, subgraph isomorphism and MCS. It tends to have high time complexity but still known to be the fastest algorithm.

In [7][8] VF and VF2 algorithm, unlike Ullmann’s algorithm, VF algorithm does not require the construction of a matrix; the adjacency list representation of the graph can be used directly. VF algorithm defines a heuristic based on the analysis of the sets of nodes adjacent to the ones already considered in the partial mapping. VF2 algorithm reduces the memory requirement from $O(N^2)$ to $O(N)$.

In 1970, Corneil and Gotlieb [4] proposed an algorithm for graph isomorphism testing based on a conjecture. These algorithms do not compare one graph against the other directly, but they work separately on one graph first and on the other next, to generate canonical labeling of the graphs that may be compared directly. They used a partitioning and refining method, which are the basis of most canonical labeling algorithms. The partitioning is done depending on the degree of vertices. The vertices having same degree belongs to same cell. The cell number is an index of all the vertices belonging to same cell. To each vertex in a cell a list is attached which indicates the neighborhood relation of vertex with other vertices in the same cell and remaining cells. The vertices in the cell are ordered lexicographically ordered. The refinement procedure is carried out on the basis of the list associated with the vertex. If the lists associated with vertices are similar and no further refinement is possible the procedure stops and these partitioning are the terminal connection partitioning for the graph. It performs partitioning on the set of vertices that result in a graph defined as terminal quotient graph. Further refinement is done and vertex quotient graph is defined. Based on the above defined graph representative and reordered graph are derived. The author has considered the representative graph is a necessary condition for graph isomorphism and reordered graph is a sufficient condition for graph isomorphism.

In [9], based on vertex invariants the graph isomorphism testing is carried out. Vertex invariants are various attributes assigned to a vertex which remain same after isomorphism mapping. The degree or valence of vertex, labels assigned to a vertex is the some examples of vertex invariants. These vertex invariants do not change in spite of isomorphism mapping.

In [10], vertex invariant and decision tree concept is used to test graph isomorphism. The vertex invariants are used to partition the matrix of the graph before graph isomorphism detection. The vertex invariant property of graph, the size of decision tree is reduced as compared to [3]. The technique is similar to breadth pruning technique which reduces the size of decision tree remarkably still the time complexity is almost equivalent.

Decision tree is the most widely used method for inductive conclusion and simple method for knowledge representation. The basic idea [3] of the isomorphism algorithm is that all possible permutation of adjacency matrix of each of the model graph was computed offline and the permutation matrices were represented as decision tree. The matrix of input graph is matched to those of adjacency matrices in the decision tree which are identical to it. At run time, the permutation matrices correspond to these adjacency matrices represent the graph or subgraph isomorphism.

By using the row-column element of each permutation matrix we can recognize the model graph into decision tree. A row-column element $x_i$ of $n \times n$ matrix is a vector and is represented as $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$. The representation of an adjacency matrix $A$ by its row–column element is illustrated in figure 1. The $x_i, x_{i2}, \ldots$ are the row column element of matrix $x$. A root node is present at the top of decision tree. At each level of the decision tree the classification is done by comparing the row column element of permutation matrix. For the first time, the classification is done by comparing the first row–column element of the input graph by the first row-column element $x_1$ of each permutation matrix. At the $n^{th}$ level of decision tree the classification is carried out by comparing the row-column element $x_n$ of the permutation matrices. Graph $G$ has 3 vertices and therefore it has $3! = 6$ permutation matrices. The row column element of the 6 permutation matrices were then organized as a decision tree.

The decision tree formed is of exponential size depending on the number of vertices and requires huge amount of storage if the number of vertices increases. A graph with $n$ vertices has $n!$ permutation matrices. A row-column element at level $n$ of decision tree would be $n!$ at the worst case. In [10], vertex invariants are used to reduce the amount of permutation matrices which subsequently reduces the size of decision tree. As the size of decision tree is directly proportional to the permutation matrices. The vertex invariants are used to partition the vertices of graph into equivalence classes such that all the vertices assigned to the same partition have the same values for the vertex invariants. The size of the decision tree is reduced remarkably.

IV. PROPOSED WORK

As Multi-core Processors are becoming ubiquitous, applications require to be parallelized for extracting high performance from Chip Multiprocessors. By analyzing data-dependence in a program, analyzing loop dependencies and applying loop transformation parallelism can be exploited in the given algorithm. Graph is used for representing information in bioinformatics, social networking, data mining, etc. For utilizing the power of multicore and to enhance the performance of any application parallelization of sequential algorithm is required. The probable approach to parallelize many of the irregular application is the Optimistic parallelization. In irregular Programs, to understand the nature of the parallelism it is important to study the appropriate algorithms and data structures. For each application, we must illustrate the algorithm and key data structures, and depict opportunities
for exploiting parallelism. In particular, manual or automatic parallelization can use the outcomes of data-dependence profiling.

2 3

b c

1 2 3

1 a l 1

0 b 0

3 0 1 c

a1 = (a)
a2 = (1, b, 0)
a3 = (1, 0, c, 1, 0)

Figure 1: Row-column element

to guide where to parallelize in a program. Automatic parallelizing compilers have proven successful for a large set of codes, they able to extract loop level parallelism but they fail to parallelize codes when data dependence information is incomplete. To overcome the problem of automatic parallelization and to exploit the concurrency provided by the CMPs different parallelizations are data parallelism, instruction level parallelism and task parallelism. Graph isomorphism is a highly-studied topic in computer science. Parallelization of Graph Isomorphism algorithm is required. Algorithm is needed to be implemented using the parallel programming technique to exploit the concurrency provided by the multicore processors. The different approaches for parallelization can be Parallel tree contraction[13], Tiling technique[14][15], Pipeline parallelism[16], Speculative Parallelization[17], Decoupled Software Pipelining[18], Speculative Decoupled Software Pipelining[19]. These are the loop parallelization technique that can be used to parallelize the graph isomorphism algorithm to improve the performance and retrieve the information efficiently.

V. CONCLUSION

With the advent of multicore processors the burden of improving program execution speed has shifted to software developers from chip manufacturers. Mainly challenging problem in this perspective is the parallelization of irregular applications that deal with complex, pointer-based data structures such as trees, queues and graphs. It is difficult to parallelize manually the pointer-based data structure like graph.

Parallelization is the need of multicore era. Graph isomorphism is used in various field. To improve the efficiency of the algorithm the parallelization techniques are useful to overcome the limitation of sequential algorithms.

REFERENCES

[1] Narsingh Deo: Graph Theory with Applications to Engineering and Computer Science ,Prentice Hall,Inc, 1995
[12] Shijie Zhang, Jiong Yang, and Wei Jin: SAPPER: Subgraph Indexing and approximate Matching in Large Graphs. 26th International Conference on Very Large Scale Database, September 13-17, 2010