

# **Efficient method for Deadlock Detection and Avoidance using Two Way Search**

*Thesis submitted in partial fulfillment of the requirements for the award of degree of*

**Master of Engineering**  
in  
**Computer Science and Engineering**

Submitted By  
**Megha Tyagi**  
**801032014**

Under the supervision of:  
**Dr. Deepak Garg**  
Associate Professor, CSED



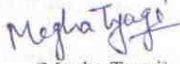
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT  
THAPAR UNIVERSITY  
PATIALA – 147004  
**June 2012**

## CERTIFICATE


---

I hereby certify that the work which is being presented in the thesis entitled, "Efficient method for Deadlock Detection and Avoidance using Two Way Search", in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Deepak Garg and refers other researcher's work which are duly listed in the reference section.


The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

  
(Megha Tyagi)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

  
(Dr. Deepak Garg)  
Associate Professor  
CSE Dept.  
Thapar University,  
Patiala

Countersigned by

  
(Dr. Maninder Singh)  
Associate Professor and Head  
Computer Science and Engineering Department  
Thapar University  
Patiala

  
(Dr. S. K. Mohapatra)  
Dean (Academic Affairs)  
Thapar University  
Patiala

## ACKNOWLEDGMENT

---

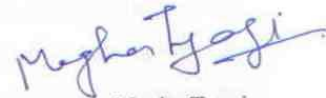
First of all I am thankful to God for blessings and showing me the right decision. With his mercy, it has been possible for me to reach so far.

I would like to express sincerest thanks to my thesis supervisor Dr. Deepak Garg, for his inspiration, guidance, stimulating suggestions, immense help and support throughout the period of this research work. He has provided me with all the necessary resources including motivation and research environment without which it would not have been possible to complete this work. It was a great opportunity for me to do this work under his supervision.

I am thankful to the authors whose work I have consulted and quoted in this work.

I lack words to express my cordial thanks to all my friends for their useful comments and constructive suggestions during all the phases of my life.

Finally, I convey deep sense of gratitude towards my family members for their moral and financial support and encouragement without which it would not have been possible to bring out this thesis.



Megha Tyagi

(801032014)

## ABSTRACT

---

---

Dynamically changing graphs are used in many applications of graph algorithms. The scope of these graphs are in graphics, communication networks and in VLSI designs where graphs are subjected to change, such as addition and deletion of edges and vertices. There is a rich body of the algorithms and data structures used for dynamic graphs. The thesis discussed the techniques and data structures used in various dynamic algorithms. The effort is tried to find out the comparison in these techniques namely the hierarchical decomposition of graphs and highlighting the ingenuity used in designing these algorithms. The thesis provides the comparative analysis of dynamic graph techniques over the various graph properties like planarity , spanning forest, edge connectivity and bipartion.It analyze that which techniques works better for which properties and minimizes the time and space usage.

The thesis propose the algorithm for one of the online dynamic graph application,i.e deadlock detection. A deadlock detection and avoidance technique is based on various techniques of representing the directed acyclic graph. The various graph model is developed based on the resource allocation graph and detecting the cycle in the graph using wait for conditions. These techniques are much discussed in past and has the algorithm for detecting the cycle for centralized system or distributed system for multiprocessing environment. This newer approach described in this thesis defines the online algorithm approach to detect the deadlock when the new edge is created and maintain the topological order of the graph using the two way search method for cycle detection.

# Table of Contents

---

---

Certificate.....	i
Acknowledgement .....	ii
Abstract .....	iii
Table of Contents.....	iv
List of Figures .....	vii
List of Tables .....	viii
1 Introduction.....	1
1.1 Character Recognition.....	2
1.1.1 Online character recognition.....	2
1.1.2 Offline character recognition.....	3
1.2 Handwriting style.....	4
1.3 Commercial Products Based on Pen Computing.....	6
1.4. Introduction to Devnagari Script.....	7
1.4.1 Devnagari Writing System .....	7
2 Online handwritten recognition system.....	10
2.1 Issues in Online Handwritten Character Recognition for Indian Writing ....	10
2.1.1 Liable to vary.....	10
2.1.2 Number of stroke classes.....	11
2.1.3 Presence of vertical appendages of modifiers.....	11
2.1.4 Directionality of writing.....	11
2.1.5 Variation of the number and writing order of multi stroke.....	12
2.1.6 Arbitrary pen-lifts in the course of writing a stroke.....	12
2.1.7 Character insertions to the left of an already written character.....	12
2.2 Steps for Online Handwriting Recognition.....	12
2.2.1 Data collection.....	13
2.2.2 Preprocessing.....	13
2.2.3 Segmentation.....	16
2.2.4 Feature Extraction.....	16

2.2.5 Recognition.....	18
2.2.6 Postprocessing.....	18
3 Literature Review.....	19
3.1 Preprocessing.....	19
3.1.1 Normalization.....	19
3.1.2 Smoothing.....	20
3.1.3 Resampling.....	20
3.1.4 De-hooking.....	21
3.1.5 Interpolation.....	21
3.2 Feature Extraction.....	22
3.2.1 Structure based feature extraction.....	22
3.2.2 Dominant point.....	30
3.3 Recognition.....	38
4 Problem Statement.....	43
4.1 Introduction.....	37
5 Implementation.....	44
5.1 Recognition of matras of Devanagari script.....	44
6 Conclusions and Future Scope .....	47
6.1 Conclusion.....	47
6.2 Future Scope.....	47
References.....	49
List of Publications.....	53

## List of Figures

---

Figure 1.1 Units of online handwriting.....	2
Figure 1.2 Types of character recognition.....	3
Figure 1.3 Boxed discrete handwriting.....	5
Figure 1.4 Various handwriting styles.....	5
Figure 1.5 Pen based input devices.....	7
Figure 1.6 Hindi matras.....	8
Figure 1.7 Different forms of Devanagari characters.....	8
Figure 1.8 Different representations of character.....	9
Figure 2.1 Shape variations.....	11
Figure 2.2 Preprocessing steps.....	14
Figure 2.3 Preprocessing.....	15
Figure 2.4 Normalizing size.....	15
Figure 2.5 Normalizing rotation.....	15
Figure 2.6 Sampling.....	15
Figure 2.7 Resampling.....	16
Figure 2.8 Segmentation.....	16
Figure 2.9 Online handwriting recognition phase.....	18
Figure 3.1 Thinning.....	19
Figure 3.2 Skeletonization.....	20
Figure 3.3 Missing points due to speed of writing.....	21
Figure 3.4 Interpolation of missing points.....	21
Figure 3.5 Character features.....	23
Figure 3.6 Shadow features.....	24
Figure 3.7 Chain coding.....	24
Figure 3.8 Intersection of end points.....	24
Figure 3.9 Diagram of proposed technique.....	25
Figure 3.10 Bounding box problem.....	25
Figure 3.11 Directional coding.....	28

Figure 3.12 Intersection with points.....	28
Figure 3.13 Distance from particular point to other point.....	29
Figure 3.14 SC features at the considered point.....	30
Figure 3.15 Polygon with the imagined rectangle.....	32
Figure 3.16 Dominant point detection of online script.....	32
Figure 3.17 Feature extraction for sample of character of class 2.....	33
Figure 3.18 Chromosome shaped curve.....	36
Figure 3.19 Teh-chin algorithm.....	37
Figure 3.20 Teh-chin algorithm(k curvature).....	37
Figure 3.21 Teh-chin algorithm(1 curvature).....	37
Figure 3.22 Binary classification.....	38
Figure 3.23 Training of MSTDNN.....	40
Figure 3.24 Recognition phase of freeman’s direction procedure.....	42
Figure 5.1 Snapshot of excel sheet having dominant point and their distance from centre.....	43
Figure 5.2 Preprocessing of stroke.....	43



## List of Tables

---

---

Table 1.1	Comparison between online and offline handwritten characters.....	3
Table 2.1	Comparison between Global and Local features.....	17
Table 3.1	Local curvature assignment.....	35
Table 5.1	Matras representation.....	44
Table 5.2	Recognition accuracy with 900 sampled data.....	46

# CHAPTER 1

## INTRODUCTION

---

---

### 1.1 Dynamic Graphs

A Graph is a collection of nodes and edges which represent the network of entities and association between these nodes. Dynamic graphs are not fixed wrt time, but can evolve through local changes of the graph. Any problem associated with dynamic graph should be solved quickly as the new changes arrive after each modification. In the current scenario, no problem is truly static, so each problem may have some dynamic changes which make the problem dynamically ridden than absolutely static. In communication networks for instance, a network changes its routes as nodes and links go down due to failures and repairs. The dynamic graphs[1] are used in almost every application ,which is dynamically changing. The World Wide Web is the biggest example of dynamic graphs as the new servers and host are keeps on adding in the graph and makes the graph dynamic. While the updation takes place in the graph, the dynamic graph maintain the various properties of graph like, graph connectivity, planerarity, spanning forest of the graph and bipartions[2][3].

**Definition:** A dynamic graph  $G=\{G_0, G_1, \dots, G_m\}$  is a sequence of graphs, where  $G_s = (X_s, Y_s)$  represent the instance of graph at any time  $t$ .

The Dynamic graph can be visualized as a world wide web where the graph vertexes represent the nodes and edges represent the links in the graph. The web graph changes dynamically as various nodes and links losses functionality as the network becomes larger.

## 1.2 Dynamic Graph Algorithm

Dynamic graph algorithm handle the graph problems, where the graph undergoes the series of updates including the insertion of an edge and deletion of an edge and answers the various queries like, Whether the graph is connected or not. The algorithm finds out the solution for the various updates and queries and performs better than the static algorithm which answers to the solution computing from the Scratch. Hence the dynamic graph algorithm does not require the whole previously computed information about the graph and improves the lower bound as comparable to their static counterparts. A dynamic graph algorithm is a data structure which is operated on a graph and they support two types of operations [4]:

- Updates and
- Queries.

An update is a local change of the graph like insertion and deletion of an edge and a query is a question about a certain property of the current graph like, the two vertex are connected or not in the graph. The purpose of dynamic graph algorithms using the different data structure tools is to find out the faster update operation by using the structural information of the graph. The dynamic graph update operations are faster than any static counterpart of the algorithm. Usually, queries take less time than updates, and the sequence of operations (updates and queries) is not known in advance. There are various dynamic graph algorithmic techniques which are used to perform the updation and query operation on the dynamic graphs. The techniques perform better for different properties of graph and improve the lower time bound than their static counterparts.

**Definition :** The Dynamic Algorithm compute some function  $X$  on the initial input  $Y$  and maintain the detail about  $X(Y)$  where  $Y$  is a initial input ,and work without re-evaluating  $X(Y)$  from starting as does in static algorithms.

A dynamic graph algorithm maintains a given property  $P$  on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property  $P$  quickly, and perform update

operations faster than recomputing from scratch, as carried out by the fastest static algorithm.

According to the functionality of dynamic graph algorithms they can be categorized in two groups[5]. The fully dynamic graph algorithms perform both the insertion and deletion in the graph and also answer about the query. Partially dynamic algorithms are those which perform either the insertion in graph or deletion in graph but do not perform the both operations together in the same algorithm.

- The graph is incremental if it supports insertions only
- And decremental if it supports deletions only.

The dynamic graph algorithms maintain the updates operation on directed and undirected graph. For undirected graph, the dynamic graph algorithm uses the different techniques like clustering, sparsification and randomization by using different data structure tools. These algorithms reserve the properties of dynamic graphs like vertex and edge connectivity, minimum spanning tree[2][3].

For directed graph, the dynamic graph algorithm deals with the two problems for maintaining the updates and query operation.

- In the fully dynamic transitive closure problem a directed graph  $G = (V, E)$  is maintained under an intermixed sequence of the following operations[6]:

**Insert**(x, y): insert an edge from x to y.

**Delete**(x, y): delete the edge from x to y.

**Query**(x, y): return yes if y is reachable from x, and return no otherwise.

- In the fully dynamic All Pairs Shortest Path (APSP) problem([7],[8]) a directed graph  $G = (V, E)$  is maintained with real-valued edge weights under an intermixed sequence of the following operations:

**Update**(x, y,w): update the weight of edge (x, y) to the real value w, this will includes as a special case both edge insertion (if the weight is set from  $+\infty$  to  $w < +\infty$ ) and edge deletion (if the weight is set to  $w = +\infty$ ).

**Distance**(x, y): output the shortest distance from x to y.

**Path**(x, y): report a shortest path from x to y, if any.[2]

Throughout the thesis, 'm' and by 'n' denote by number of edges and vertices in G, Respectively.

There has been a lot of research in the area of directed graph where these two problems are widely discussed and provided with some solution. The main goal in case of directed graph is to minimize the running time of the algorithms. The undirected graph in dynamic graph algorithm techniques achieves the better time bound as compared to directed graph. Undirected graphs provide the algorithmic solution in polylogarithmic time bounds as compared to polynomial time bound of directed graphs. The goal of this thesis is to study the algorithmic techniques that have been used in the literature and the tools for building the data structure used in dynamic graph algorithms. The thesis also provides the comparison of various dynamic graph algorithmic techniques, which are used in undirected graph and compares the time bound for update and query operation for various dynamic graph properties.

In particular, some common data-structural tools[10] those are at the base of these techniques are also discussed in the coming chapter. This will present all the latest results in a unifying framework so that they can be better understood and deployed also by non-specialists for various dynamic graph applications.

Graph algorithms are fundamental in computer science, and much work has been done into the study of dynamic graph algorithms. That is, algorithms that maintain some property of a changing graph more efficiently than recomputation from scratch after each change.

The algorithms designed for various application uses similar techniques and hence a brief description about the techniques is discussed. These techniques use graph decomposition and one of the very important reasons for getting the better bound is how well the hierarchy in this partition is used by the data structure. The data structures used are those that maintain the properties of dynamically changing trees.

The dynamic graph algorithm provides answer to the following operation.

- Whether the two nodes are connected or not?
- Retain the various graph properties like minimum spanning forest, vertex connectivity and bipartiteness etc.
- Various update taking place in dynamic graph when the new links are added and deleted from the graph.

According to the operation supported, the algorithm can be divided into two categories[4]:

- **A fully dynamic graph algorithm:**

This algorithm supports both the insertion and deletion of edges.

- **Partially dynamic graph algorithms:**

This type of algorithm support only edge insertion or deletion but not both.

The dynamic graph algorithms are the prominent area of research for last few decades and various algorithms has been developed to maintain the graph properties like minimum spanning tree, planarity, 2-edge connectivity and bipartiteness. As dynamic graph algorithms perform better than their static counterparts, they are more difficult to design and analyze.

There are lots of applications for dynamic graph algorithm where the better time bound is required to solve the complex problems like,

- Assembling planning
- Chip design
- Graphics
- And communication networks

So, as the internet expands and the user on the web increases, there is lot of requirement for the efficient dynamic graph algorithm.

The main contribution of this thesis is to provide the comparative analysis of a new and general technique for designing dynamic graph algorithms, which are clustering sparsification and randomization and also providing the solution to deadlock detection using incremental cycle detection. These techniques are used to speed up many fully

dynamic graph algorithms[4]. Roughly speaking, when the sparsification is applicable, it speeds up a  $T(n, m)$  time bound for a graph with 'n' vertices and 'm' edges to  $T(n, O(n))$ , that is, almost to the time needed if the graph were sparse[11]. Sparsification applies to a wide variety of dynamic graph problems, including minimum spanning forests, edge and vertex connectivity.

## 1.3 Dynamic graph algorithmic Techniques and tools

### 1.3.1 Clustering

The clustering is the subdivision of graph node set into groups. This technique firstly introduced by Fredricson[10]. It partitions the graph into a smaller subdivision of connected sub graph called **clusters**. The techniques use the tree data structure to store the information about the graph edges and nodes.

**Definition:** A clustering  $\mathcal{C}$  (G) of a graph  $G=(X, Y)$  is a subdivision of vertices X into disarranges, nonempty subset of  $\{C_1, C_2, C_3, \dots, C_k\}$  where  $C_i \in \mathcal{C}$ .

The technique work as follow:

- It is based upon the decomposition of vertex set V into the sub graph called clusters and the decomposition applied recursively to the higher level. And the information about the sub graph is combined with topology tree [3].
- The clustering technique improves further in which the edges can be in multiple groups, and only one edge will be selected depending upon the topology of the spanning tree.

Dynamic clustering can be defined as:

**Definition:** A dynamic clustering  $\mathcal{C}$  (G) =  $\{C_1, C_2, C_3, \dots, C_L\}$  of a dynamic graph G, with length L, consist of a set Clustering  $C_1, C_2, C_3, \dots, C_L$  where  $C_i$  is a clustering of a graph  $G_i$ .

### 1.3.2 Sparsification

This technique was introduced by Eppstein et al [11] and it is a general technique which can be used as a black box in designing algorithms. The technique reduces the number of

edges in the graph and speedup the dynamic algorithm. Due to this technique the time bound of the algorithm improves and become analogous to the sparse graphs. The technique works on the top of the given algorithm and does not demand the structural detail of the graph.

### **1.3.3 Randomization**

The third technique introduced by Henzinger and King [12] for dynamic graph algorithm uses the power of randomization for improving the faster update time. In this technique the graph decomposition takes place with randomization.

The technique advances the lower time bound for fully dynamic graph algorithm for properties like connectivity bipartiteness and minimum spanning forest of a graph. The result of this technique achieves the faster fully and partially dynamic algorithm.

## **1.4 Application of Dynamic Graph Algorithms**

In many situations, graphs are likely to change and need to be updated to maintain the graph properties. According to the changes the dynamic graph algorithms can be classified into three categories. The algorithm is incremental when it deals with the insertion of edges in the graph; and the graph is decremental when it deals with the deletion of edges in the graph. The algorithm is fully dynamic when the edges can be inserted and deleted at some time and can be handled by the same algorithm. These algorithms have a wide area of research for various applications some of them are mentioned below:

- Incremental cycle detection or topological ordering occurring in circuit evaluation
- pointer analysis
- management of compilation dependencies
- and deadlock detection
- In some applications cycles are not fatal, strong components, and possibly a topological order of them, must be maintained. An example is speeding up of pointer analysis by finding cyclic relationship



## 1.5 Structure of the Thesis

The rest of thesis is organized in the following order:

**Chapter-2:** This chapter describes the entire three dynamic graph algorithms techniques. The analysis provides the advantages and disadvantages of each technique and defines in which application the techniques work efficiently. Also the data structure and tools are defined and discussed in this chapter providing the profound analysis of data structure tools used with each technique. The chapter discusses the major area of research in dynamic graph that is deadlock detection. It also gives the detail description of previous deadlock detection techniques.

**Chapter-3:** This chapter provides the problem statement of the thesis and the methodology used to solve the problem. The clear insight about the problem statement is defined. The new approach is discussed to solve the deadlock detection with incremental cycle detection technique.

**Chapter-4:** This chapter provides the comparative analysis of dynamic graph algorithm techniques in table format. The comparison is done on the various dynamic graph properties like connectivity, minimum spanning tree and on space usage by each technique. The update and query time is compared for undirected graph algorithmic techniques.

**Chapter-5:** This chapter provides the complete solution to the problem defined in chapter 3. The new method is defined which is used in the deadlock detection technique. The new approach for the deadlock detection is analyzed and the new algorithm is given for the problem solution. Also the running time of the algorithm is compared from the previous algorithm lower bound.

**Chapter-6:** This chapter provides the conclusion for the new technique introduced and gives the insight of usefulness of the technique for the various applications for future scope.

## CHAPTER 2

### LITRATURE REVIEW

---

#### 2.1. General Techniques for Dynamic Graph Algorithm

The basic update operations in dynamic graph algorithms are edge deletion and insertion. Typically, most of these techniques use some sort of graph decomposition, and partition where either the vertices or the edges of the graph to be maintained. Moreover, data structures that maintain properties of dynamically changing trees, such as Topology trees and ET trees are often used as building blocks by many dynamic graph algorithms.

The techniques considered are:

- 1) Clustering
- 2) Sparsification and
- 3) Randomization.

##### 2.1.1. Clustering

This technique was introduced by Frederickson [10]. It partitions graph into clusters (as connected sub graphs). The partition is done in such a way that each update involves only a constant number of such clusters. For example in the Topology Tree, the partition ensures that only local adjustments are needed in case of edge insertion or deletion update. The partition defined by the clusters is applied recursively. This can improve bounds as in Frederickson [10] from  $O(m^{0.67})$  to  $O(\sqrt{m})$ .

##### Use of Clustering for maintaining Minimum Spanning Forest:

Using the FindCluster [10] procedure the vertex set  $V$  is partitioned into sub trees with few neighbors(in fact  $\leq 3$ ). For a recursive partition a Topology Tree is used and better results are obtained by the usage of two-dimensional Topolnogy Tree. Here the updates

are confined to changes in a sub tree isomorphic to the one-dimensional Topology Tree. So, for a Topological partition of order  $z$ , it contains  $O(m/z)$  vertices. Therefore the bound is  $O(\sqrt{m})$ .

**Theorem:** The minimum spanning forest of an undirected graph can be maintained in  $O(\sqrt{m})$  time per update where  $m$  is the current number of edges. The Clustering technique is problem-dependent to a large extent and therefore is difficult to use as a black box so it needs to be combined with more general techniques to produce a more efficient algorithms.

The clustering is the subdivision of graph node set into groups. It partition the graph into a smaller subdivision of connected sub graph called **clusters**. The techniques use the tree data structure to store the information about the graph edges and nodes.

**Definition:** A clustering  $\zeta(G)$  of a graph  $G=(X, Y)$  is a subdivision of vertices  $X$  into disarranges, nonempty subset of  $\{C_1, C_2, C_3, \dots, C_k\}$  where  $C_i \in \zeta$ .

The technique work as follow:

- It is based upon the decomposition of vertex set  $V$  into the sub graph called clusters and the decomposition applied recursively to the higher level. The information about the sub graph is combined with topology tree [4].
- The clustering technique improves further in which the edges can be in multiple groups, and only one edge will be selected depending upon the topology of the spanning tree.

Dynamic clustering can be defined as:

**Definition:** A dynamic clustering  $\zeta(G) = \{C_1, C_2, C_3, \dots, C_L\}$  of a dynamic graph  $G$ , with length  $L$ , consist of a set Clustering  $C_1, C_2, C_3, \dots, C_L$  where  $C_i$  is a clustering of a graph  $G_i$ .

Clustering when used for a single level in dynamic graph algorithms obtain the lower bound of  $O(m^{2/3})$  but when the partition is applied recursively to the higher level using the two dimensional topology tree the lower time bound improved to  $O(m^{1/2})$ .

**Advantages:** the technique work faster for the Dynamic graph algorithm and is suitable for the deterministic algorithm. Clustering can be in cooperated with other graph technique to produce the efficient results. The algorithm has lower search space in dynamic approach and has a quick response to the clustering events.

**Drawback:** To the large extent the technique is problem dependent and can be applied as a black box for dynamic algorithms.

### 2.1.2. Sparsification

This technique was introduced by Eppstein et al [11] and it is a general technique which can be used as a black box in designing algorithms. The technique reduces the number of edges in the graph and speedup the dynamic algorithm. Due to this technique the time bound of the algorithm improves and become analogous to the sparse graphs. The technique works on the top of the given algorithm and does not demand the structural detail of the graph.

The technique makes use of certificate to be applied on the graphs. The definition is as follows:

#### Definitions

**Certificate:** For any graph property  $P$ , and graph  $G$ , a certificate for  $G$  is a graph  $G'$  such that  $G$  has property if and only if  $G'$  has the property  $P$ .

**Strong Certificate:** For any graph property  $P$ , and graph  $G$ , a strong certificate for  $G$  is a graph  $G'$  on the same vertex set such that, for any  $H$ ,  $G \cup H$  has property if and only if  $G' \cup H'$  has the property  $P$ .

**Sparse Certificate:** A strong certificate with at most  $cn$  edges on a graph  $G$  which has  $n$  vertices for some constant  $c$ [11].

The technique work as follow:

- The graph with  $E$  edges and  $V$  nodes partition the edge of graph  $G$  into a assembly of  $O(E/V)$  sparse sub graphs where each sub graph is a order of  $O(V)$ .

The graph is then decomposed into sparser sub graph incorporating the meaningful information for each sub graph and having the sparse certificate. Hence each node in the tree is represented by the sparse certificate.

- Now when any insertion or deletion take place for edges, the  $O(E/V)$  graphs with  $O(V)$  links each would be required for updates.

Let  $A$  be the algorithm that maintain some number of properties on the dynamic graph  $G$  with time bound  $F(E, V)$ , where  $E$  is a number of edges and  $V$  is a number of vertex set. So the sparsification advance the decomposition of  $G$  into smaller sub graph with  $O(V)$  edges each.

Hence the technique uses the dynamic algorithm to only some small sub graph of  $G$ , resulting into advanced time bound of  $F(n, O(n))$ .

So, the techniques improves the time bound to  $O(n^{1/2})$  where the previously known time bound were  $O(m^{1/2})$  for the update operation.

**Theorem** : Let  $P$  be a property for which we can find sparse certificates in time  $f(n,m)$  for some well-behaved  $f$ , and such that ,a data structure is constructed for testing property  $P$  in time  $g(n,m)$  which can answer queries in time  $q(n,m)$ .

Then there is a fully dynamic data structure for testing whether a graph has property  $P$ , for which edge insertions and deletions can be performed in time  $O(f(n,O(n))) + g(n,O(n))$ , and for which the query time is  $q(n,O(n))$ [11].

**Theorem** : Let  $P$  be a property for which stable sparse certificates can be maintained in time  $f(n, m)$  per update, where  $f$  is well-behaved, and for which there is a data structure for property  $P$  with update time  $g(n,m)$  and query time  $q(n,m)$ . Then  $P$  can be maintained in time  $O(f(n,O(n))) + g(n,O(n))$  per update, with query time  $q(n,O(n))$ [11].

Sparsification applies to a wide variety of dynamic graph problems, including minimum spanning forests, edge and vertex connectivity. As an example, for the fully dynamic minimum spanning tree problem, it reduces the update time from  $O(pm)$  [16, 17] to  $O(pn)$ .

## Advantages

The technique has the following advantages:

- The techniques applied to the wide variety of graph problems comprising vertex and edge connectivity, minimum spanning forest and bipartite graphs. As an example, for the fully dynamic minimum spanning tree problem, it reduces the update time from  $O(PE)$  to  $O(PV)$ .
- The technique speed up the dynamic graph algorithm and work superior for small update sequences.
- It provides the improved space usage of  $O(E \log V)$  compared to other graph techniques.

### 2.1.3. Randomization

The third technique introduced by Henzinger and King [12] for dynamic graph algorithm uses the power of randomization for improving the faster update time. In this technique the graph decomposition take place with randomization.

The technique advance the lower time bound for fully dynamic graph algorithm for properties like connectivity bipartiteness and minimum spanning forest of a graph. The result of this technique achieves the faster fully and partially dynamic algorithm.

#### Random Sampling

The random sampling is the key idea behind this technique. When any of the edge  $e$  is removed from the tree then the edges (non tree edges) which are incident on the tree  $T$  will be randomly selected for the replacement of the deleted edge.

The main idea of the technique:

- The graph is decomposed to the  $O(\log n)$  level. Where the dense part (highly connected) of the graph is connected to the lower level than those where the graph is sparse (weakly connected).
- When the tree edges are deleted at level  $i$ ,  
There exit the two cases:

**Case 1:** then the high probability edge is randomly selected to recombine two disjoint sub trees using random sampling.

**Case 2:** when by deletion of the edge the graph become sparse, and random sampling fails then These edges are moved to level (i+1) and the same procedure is applied recursively on level (i+1).

Hence, the technique maintains the spanning forests for the graph, for each level i, whose edges are in level i and are below it. The technique implements the algorithm for various properties of Dynamic algorithms using the eulerian tour implementation of the spanning tree.

**Theorem:** Let G be a graph with 'm' edges and 'n' vertices subject to edge deletions only. A spanning forest of G can be maintained in  $O(\log^3 n)$  expected amortized time per deletion, if there are at least (m) deletions. The time per query is  $O(\log n)$ [13].

## 2.2. Data Structure tools for implementing dynamic graph techniques

There are many fully dynamic data structure for the dynamic graph problems. Many use the concept of partitioning of vertex into a disjoint set of paths. Some of the data structures are:

- Topology tree
- ET trees
- Dynamic Trees
- Top Trees

### 2.2.1 Topology Trees

The tree represents the hierarchy of the tree T. These trees were introduced by Fredricson ([10], [15]) to maintain the updates for dynamic trees. The tree uses the following terminology:

- Vertex cluster: connected sub graph of the Tree T.
- Cardinality: number of vertices in cluster.



**Definition:** The tree  $T$  defines the systematic division of the other tree, according to the topology.

For the restrained multilevel division, topology tree define the following properties:

- The tree define the node at level one as a cluster at level 1 which is the root node containing the single vertex.
- All the nodes at level  $l \geq 1$  has at most two children, which shows the node clustering at level  $l - 1$  union define the vertex cluster that node represents.

The tree uses the partition of vertices into clusters at each level where restricted partition is defined as:

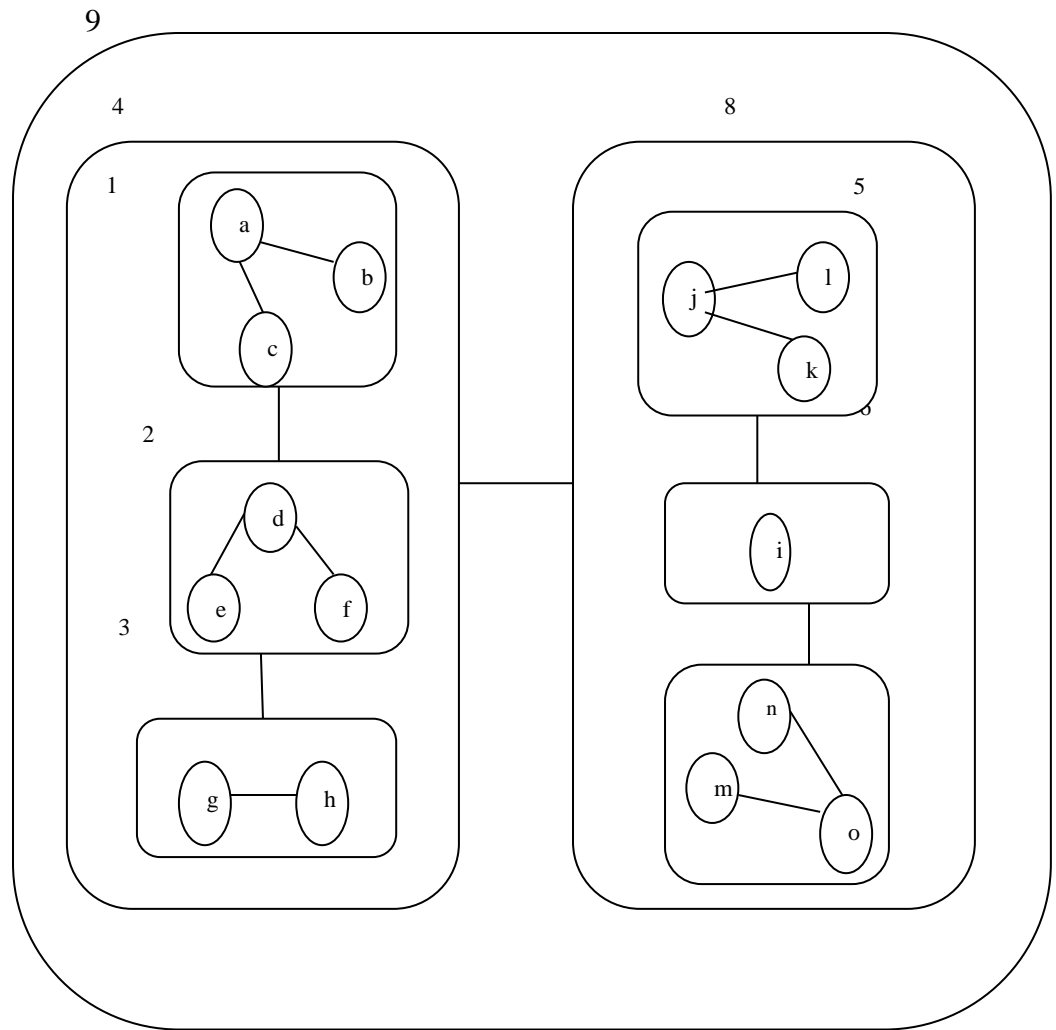
- Every Cluster with outer degree three must have a cardinality of 1, else if outer degree for every cluster is less than three than cardinality must be of 2.
- No two neighboring cluster can be incorporated and which are still satisfying the above condition.

### **Insertion and Deletion in Tree**

When deletion of the edge 'e' take place in the tree  $T$ , then the deletion makes a tree divided into two trees  $T_1$  and  $T_2$ . The union is performed on the adjacent cluster preserving the topology tree properties defined above.

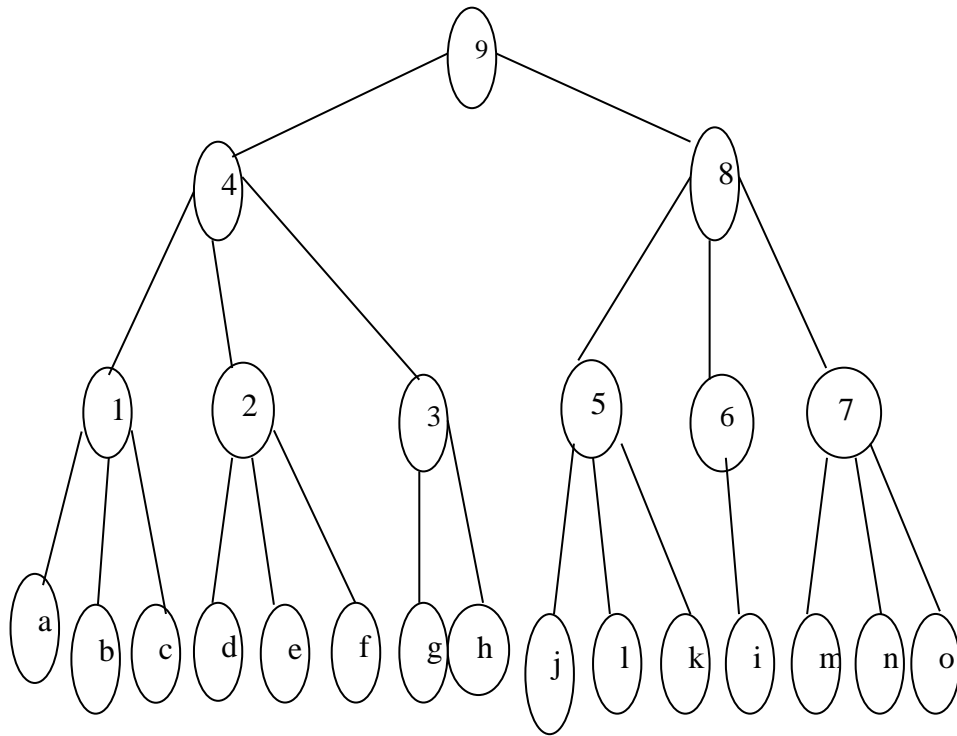
When new edge is added, the two separated tree is combined to form a single tree. If the degree of the vertex after union exceed from 3 then the deeply nested cluster is then split till root to preserve the topology tree properties.

The Figure 2.1 next page defines the hierarchical topological partitioning and the analogous topology tree in Figure 2.2.



**Figure 2.1 the hierarchical Topological partition**

The analogous topology tree for the above Figure 2.1 is shown on the next page:



**Figure 2.2 Topology Tree.**

The height of the topology tree is  $O(\log n)$ . Hence any update in tree involves some level and require the local adjustments, so time required to update the topology tree is  $O(\log n)$ .

### 2.1.2 Euler Tour Tree Data Structure

The Euler tour is one of the data structure used to implements the randomized algorithmic techniques. The tree  $T$  is encrypted with the 'n' nodes in the tree and any random node can be chosen as a root node.

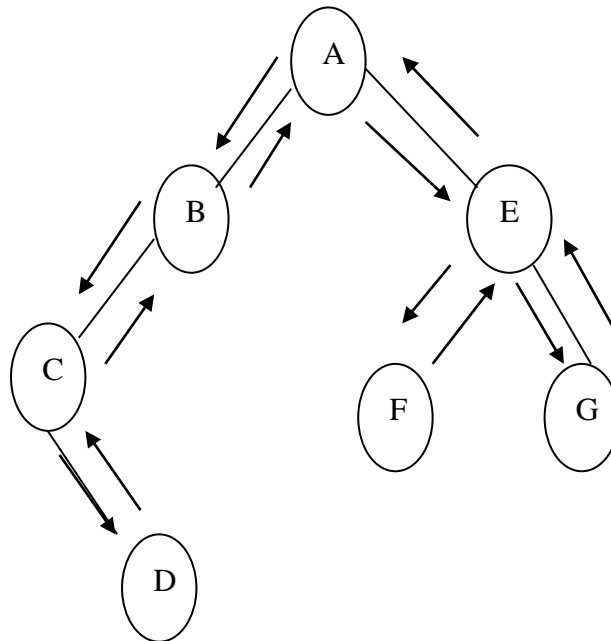
The **Euler tour** visits the every edge exactly once and if represented as a tree then visiting every edge exactly twice, once entering into the vertex and once leaving that vertex[16].

**Definition:** The Euler tour tree reserve the Euler tour of the tree and represent the Euler tour in the balance binary search tree.

Euler tour tree are the substitution for the link-cut trees. These trees are apparent and easier to evaluate than the link cut trees for dynamic graphs.

The tree does not store the path information about the trees but store the procure information on the sub trees. The tour is the depth first traversal of the tree which return to the root node at the end.

The Figure 2.3 shows the Euler tour of the tree, directed edges show the sequence of visitation.



**Figure 2.3 Euler tour of tree**

The sequence of visitation here is A , B , C ,D ,C,B,A ,E ,F ,E ,G ,E ,A for the tree T.

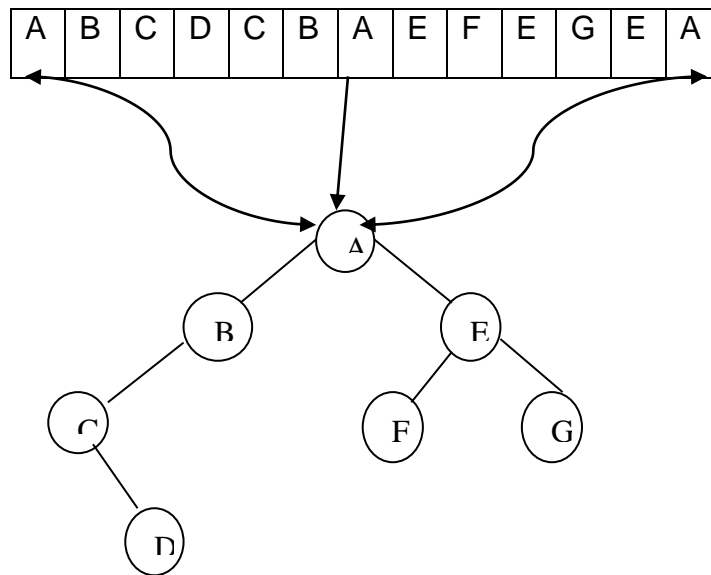
The tour started and ended with the root node A.

The Euler tour function is called for the visitation where the function E(s) is defined as:

- E(r)
- Visit the vertex r.
- While every children x of r
- Do call E(x)
- Visit r.

Here, the d-degree node is visited 'd' times and edges twice except the root node which is visited d+1 times. The function  $E(T)$  represents the sequence of Tree T. Adjacency list with arrays and pointers are used to store the vertex occurrence.

The node holds the pointer to the visited sequence in the BST presenting the first and last time it was visited. The Figure 2.4 shows it below:



**Figure 2.4 pointer related to root.**

### Operations

The Euler tour tree supports the following operations:

- **Cut(x):** cut the sub tree rooted at x while splitting the BST before the first visit and last visit to x and concatenating the both.
- **Find root(x):** returning the root of the node x, where root is visited first and last.
- **Link(x, y):** it insert x sub tree as a child of node y.

Every operation in Euler tour can preserves the property by splitting, merging and searching in the Euler tour tree. The operation involves the  $O(\log n)$  per operation for the update in the Euler tour tree.

ET-trees are significantly simpler than the other data structures discussed so far, which at first glance would make them the preferred choice in dynamic tree applications. The

trouble with them is that they cannot handle path queries efficiently. Each edge in the forest appears as two nodes in the representation, and they may be arbitrarily far apart. This makes it hard to aggregate information about a specific path. Aggregating information over the entire tree, however, is relatively simple.

### 2.1.3 Dynamic Trees

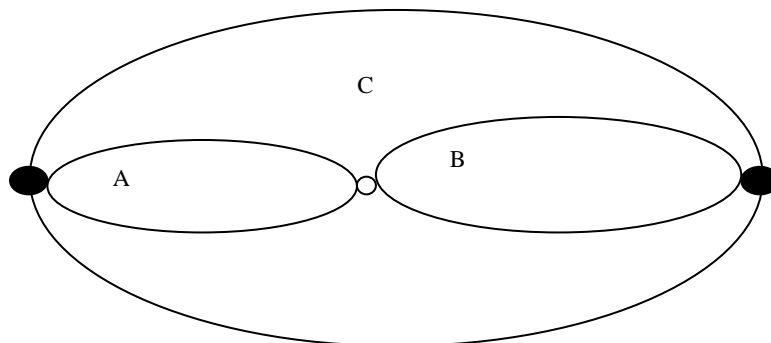
There are number of dynamic trees which are used for dynamic graph algorithms to maintain the graph properties. One of them is **link-cut trees** which have the application in the area of Network problems and dynamic connectivity problem. These trees were introduced by Sleator and Tarjan ([5],[17]). The tree maintain the logarithmic amortized time per update operation.

### 2.1.4 Top Trees

The tree was introduced by alstrup[19]. The top tree support the path oriented updates and Queries basically for the problem of Divide and conquer algorithm. The working of top tree depends on boundary nodes and clusters.

Maintaining top trees of height  $O(\log n)$  and with  $O(m^{1/2})$  cluster nodes supporting Link, Cut, and Expose with a sequence of  $O(\log n)$  Merge and Split and  $O(1)$  create and Destroy operations per update. The sequence itself is computed in  $O(\log n)$  time[18].

The Figure 2.5 shows the case of two Clusters A and B and the parent node C in the top tree.



**Figure 2.5 cluster with boundary nodes**

### 2.3. Application of Dynamic Graph algorithm: Deadlock Detection

Deadlock is a situation where multiple events at the same site or different sites are simultaneously waiting for the other process to complete but neither ever does. This problem is common in centralized system or in distributed environment .in parallel computing the software and hardware locks are used for handling the shared resources while maintaining the process synchronization and hence leads to deadlock[20].

There are three methods for dealing with this problem: deadlock prevention, deadlock avoidance, and deadlock detection combined with recovery. The two first methods ensure that the system will never enter a deadlock state while the third method allows the system to enter a deadlock state and then recover from it. Deadlock prevention prevents deadlocks by restraining how requests can be made. This implies restrictions of concurrency while deadlock avoidance and deadlock detection combined with recovery provide full concurrency.

Both these last methods represent the system by a resource allocation graph and use an algorithm for checking whether the resource allocation graph is cyclic or not (a cycle in the graph corresponds to a deadlock situation). The commonly recommended cycle detection algorithm is Topological Sorting having a running time of  $O(n + e)$  where  $n$  is the number of nodes and  $e$  is the number of edges in the graph. Topological Sorting gets cheaper as the graph gets sparser. However, with an efficient use of resources, i.e., when the process uses almost all the resources in the system, the graph will be dense.

While creating a resource allocation graph, some of the resources are allowed to be released or allocated at the time. So the graph adds the edges to the graph when the new request for the resource is arrived and deletion take place from the graph when the resource is released by the process. The next section describe the online approach as the new request for the edge  $(u, v)$  come and check that if the insertion of edge create any cycle or not.

The section 2.3.1 discusses the some of the previous techniques to detect the deadlock and their time bound. The section 5 defines the new proposed technique for detecting the

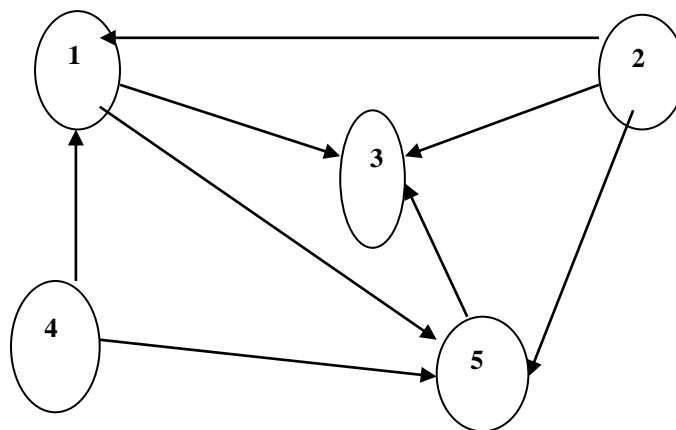
two way searching algorithm to report the cycle if exists in the graph and analyzing the time bound for the algorithm. Section 5.2 analyzes the performance comparison of the proposed deadlock detection algorithm with some of the existing ones. And chapter 6 provides some discussion and open problem in the area of deadlock detection.

### 2.3.1 Previous Techniques for Cycle Detection

Some of the previous work done in deadlock detection and avoidance is by using the path matrix. As the insertion and deletion of the edges only change the part of the resource allocation graph , path matrix technique does not scan the whole graph and rely on the recompilation of the path matrix to answer whether the cycle exist by addition of the new edge (u, v). The unsuccessful allocation of the resource to the process can (that is detecting the cycle) can be found by it in  $O(1)$  amortized time and keeping the path matrix representation of the resource allocation graph acyclic.

#### Deadlock Detection Using Path Matrix

The resource allocation graph has the three operations to perform, the unsuccessful allocation of the resources means the edge (v, w) will create a cycle, or the correct allotment and release of edges will keep the graph acyclic. The Figure 2.6 shows the wait for graph and the corresponding path matrix[21].



**Figure 2.6 Wait for graph and path matrix.**



The corresponding path matrix is shown here:

$$\begin{pmatrix} 0 & 0 & 2 & 0 & 1 \\ 1 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

The operation supported in this path matrix is to find out that the insertion of edge will create any cycle and successful insertion of the edge will update the path matrix and keep the resource allocation graph acyclic. The graph G shown above represent the path matrix P where P [2, 3] shows the total 3 number of different path exist from node 2 to node 3.

Here the path matrix representation of graph G is defined over  $|V| \times |V|$  matrix  $P = [I, J]$   $1 \leq i, j \leq n$  where

$$[i, j] = \begin{cases} p & \text{if there are } p \geq 1 \text{ different path from node } i \\ & \text{To node } j \\ 0 & \text{if there is no path from } i \text{ to node } j \end{cases}$$

The path matrix represents the unique way of representing the direct acyclic graph. And the solution is unambiguous.

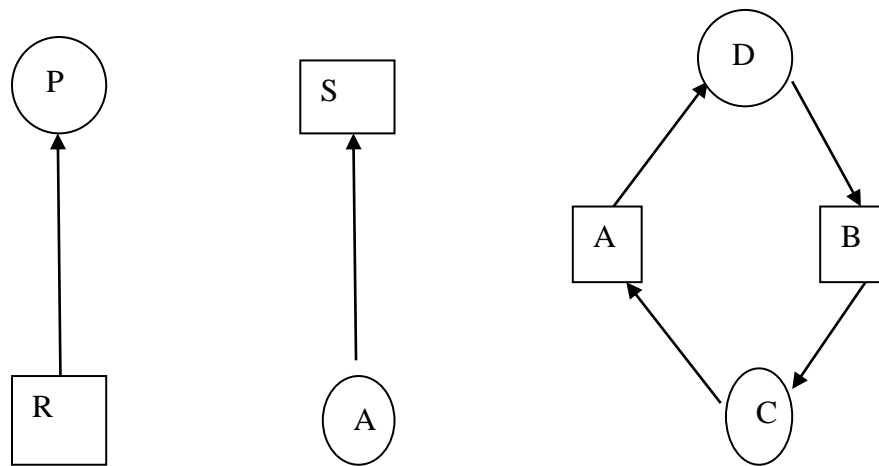
The operation of the graph model can be performed As follow:

- 1) the insertion of an edge can be checked whether it create a cycle: if the new edge (v, w) is added then check whether  $[w, v] \neq 0$ , if this is the case that means at least one path exist from node w to node v and that will lead to create a cycle in the resource allocation graph .the operation can be detected in  $O(1)$  time bound [21].
- 2) Successful insertion and deletion of edge: These operations can be done by successful insertion and deletion of path matrix  $P_{(v, w)}$  from the path matrix P.

### Deadlock Detection by Exploring Strongly Connected Components

Some of the previous techniques detecting the deadlock use the strongly connected components to indentify the cycle from the resource allocation graph. Here the resource allocation graph of the system is the simple directed graph which defines the current state of allocation of resources at some time. The technique detects more than one point in the strongly connected component of resource allocation graph, where it indentifies the deadlock and handles it [26].

The graph has two types of nodes, processes shown by circle and a resource defines by square. The graph holds the three kinds of arcs showing the requesting resources and the resources granted to process and currently held by the process. The figure 2.7 shows the three conditions:



**Figure 2.7 : resource allocation graph**

- i) holding a resource
- ii) requesting resource
- iii) deadlock

The Problem and technique illustrates the power of the adjacency matrix and path matrix to compute the deadlock detection via strongly connected component.

**Definition:** the graph  $G = (V, E)$  is a directed graph and contain the vertices  $v_1, v_2, v_3, \dots, v_m$ . Then the adjacency matrix is defines a nonzero matrix  $(m \times m)$  containing 1 at entry  $[j, k]$  when there is a path form node  $v_j$  to  $v_k$  and contain 0 when the nodes are not adjacent.

$$a_{jk} = \begin{cases} 1 & \text{when vertices are adjacent} \\ 0 & \text{when no path exists} \end{cases}$$

The power of adjacency matrix are used to compute the path matrix for  $|V|=m$  where  $m$  are total no of vertices.

The techniques work as follow:

For the simple directed graph  $G = (V, E)$  the path matrix  $P$  is computed and  $P^t$  is the transpose of the path matrix  $P$ , and then define  $P \times P^t$ . Where the row  $j$  define the strongly connected component of node  $v_j$ .

Applying the method on resource allocation graph, computes the matrix  $P \times P^t$  and it detects the strongly connected components and hence finds out the process and resources involved in the deadlock [23].this technique provide the better way to develop the operating system algorithms for detecting the deadlock.

### **The Depth first Search for Cycle Detection**

Deadlock is the most studied problem in the area of operating system and databases. The deadlock in data bases held due to locks as some transactions are waiting for the other item which is currently held by the another process and the situation lead to deadlock. The problem occurs in static environment can be easily handled by various algorithm and one of the simplest is using Depth first search. In dynamic environment where the new edges and vertices are added simultaneously is the most studied area of recent research. The path diffusion technique is used for communication deadlock for handling the deadlock in dynamic environment.

The algorithm rules for handling the deadlock in static environment are defined below.  
 The algorithm return true if it detect the cycle else return false.

### Cycle detection (G)

- For all vertex  $v$  in  $G (V, E)$ ,  $v \in V$  and all edge  $(v, w) \in E$ .  
     Mark all  $v = \text{WHITE}$ .  
     End.
- While  $(V \neq \text{NULL})$   
     If  $(v.\text{mark} == \text{WHITE})$  then:  
         Visit  $(G, v)$ , return TRUE.  
     End
- Return FALSE.

### Boolean Visit (G, v):

- $v.\text{mark} = \text{GRAY};$
- while for each edge  $(v, w)$  in  $G$  do:  
     If  $w.\text{mark} == \text{GRAY}$  then,  
         Return TRUE.  
     Else if  $w.\text{mark} == \text{WHITE}$  then,  
         If Visit  $(G, w)$  then  
             Return TRUE.  
     End
- $v.\text{mark} = \text{BLACK}$  (visited).
- Return FALSE.

Here the given algorithm for cycle detection work well for tree data structure but need to be slightly modified for graph. If the vertex is encountered again after all of its descendent nodes earlier visited then there exist no cycle, else if the node is reachable from any of its descendent, then node is involved in cycle and reports it. The color technique is used to mark the visited node and if GREY node is encountered then there exist a cycle and need the mechanism to break it and maintain the topological order. This

is the simplest approach for the deadlock detection( [19],[20]) but there has been the immense research on the deadlock issue in computer world.

**Theorem:** the complexity of the algorithm is  $O(E + V)$  For a given graph where  $E$  is the number of edges and  $V$  is the arc in the graph the link representation of graph will take  $O(E+V)$  time to detect the cycle where as using the adjacency list representation, the time bound is  $O(V^2)$  and additional time for reordering the topological order. The improved algorithm for deadlock detection in dynamic environment will provide the better time bound.

## 2.4 Method for incremental cycle detection

There has been number of algorithm which finds out the cycle in the graph and provides the polynomial time bound for deadlock detection. The incremental cycle detection technology provides the better time bound as compared to older algorithms for online cycle detection. The incremental cycle detection has a wide area of research in recent years. The technique is being used for many applications like pointer analysis, circuit evaluation and deadlock detection. The next subsection describes one of the methods used for incremental cycle detection.

### 2.4.1 One-Way Search

This is one of the methods for incremental cycle detection. The incremental cycle detection trigger the search when the new edge is added hence finds out the cycle for the dynamic graph. The techniques help to find out the cycle from the current topological order and do not start from the scratch to find out the cycle. The algorithm of one way search for the cycle detection problems and rely on the maintenance of the topological order. If after the edge insertion the graph triggers a cycle then search stops and reorders the topological order[29]. If there is no cycle the graph only reorder the new inserted edge vertices.

When the new edge  $(v, w)$  is added,

- the testing is done for cycle detection by searching forward from the vertex  $w$ , and visiting all the vertex till finding  $v$ (there is a cycle) .
- Or finding out all the vertices reachable from  $w$  without finding  $v$  (there is no cycle).

The one way search method takes  $O(m)$  time bound per arc addition in worst case , as searching for the forward direction.

The method maintains the topological order which help in improving the time bound of the one way search.

- For the newly added edge  $(v, w)$ , if  $v < w$  then by topological order definition [33], the graph will remain acyclic.
- If after searching the node 'v' is not encountered in forward search, then there exist no cycle and the topological order is to be maintain because the node 'w' and 'v' must be out of order.

The topological order is maintained for those vertices which are visited after the forward search and placing them in front of other vertices in the graph.

To maintain the topological order the one way search method uses the counter values to retain the topological order .All the vertices are arbitrary numbered from 1 to 'n' and each vertex is initialized with the counter value 'c' to 'n'.

When a forward search encountered, the visited vertices are renumber by the search consecutively from  $c + 1$ , in a topological order with respect to the subgraph induced by the set of visited vertices, and increment 'c' to be the new maximum vertex number.

One way to reorder the visited vertices is by

- making the search depth-first and
- Order the vertices in reverse postorder.

With this scheme, all vertex numbers are positive integers no greater than  $nm$ .

Shmueli introduced this one way search as a heuristic technique for detecting a cycle but the author does not mention whether the topological order is to be maintain hence, In the worst case, every new arc can invalidate the current topological order and trigger a search

that visits a large part of the graph, so the method does not improve the  $O(m^2)$  worst-case time bound for cycle detection. But after this the time bound has been improved asymptotically.

For the better time bound for cycle detection the one way search uses the topological ordering of vertices so that the time for searching the whole graph can be minimized. The search for 'v' from 'w' need not visit vertices larger than 'v' in the current order, since no such vertex, nor any vertex reachable from such a vertex, can be 'v'. The method is described in detail here.

When the new arc (v, w) is added where  $v > w$ , then the LIMITED-SEARCH procedure is called for the forward search. This is defined as follow. The minus sign in the function denotes the subtraction.

### **Implementation of limited search**

Arc **function** LIMITED-SEARCH (node v, node w)

$F = \{w\}; A = \{(w, x) \mid (w, x) \text{ is an arc}\}$

**While**  $A \neq \{\}$  **do**,

Choose  $(x, y) \in A; A = A - \{(x, y)\}$

**If**  $y = v$  **then** return (x, y)

**Else if**  $y < v$  and y does not F **then**

$F = F \cup \{y\}, A = A \cup \{(y, z) \mid (y, z) \text{ is an arc}\}$

**End**

**End.**

**Return null**

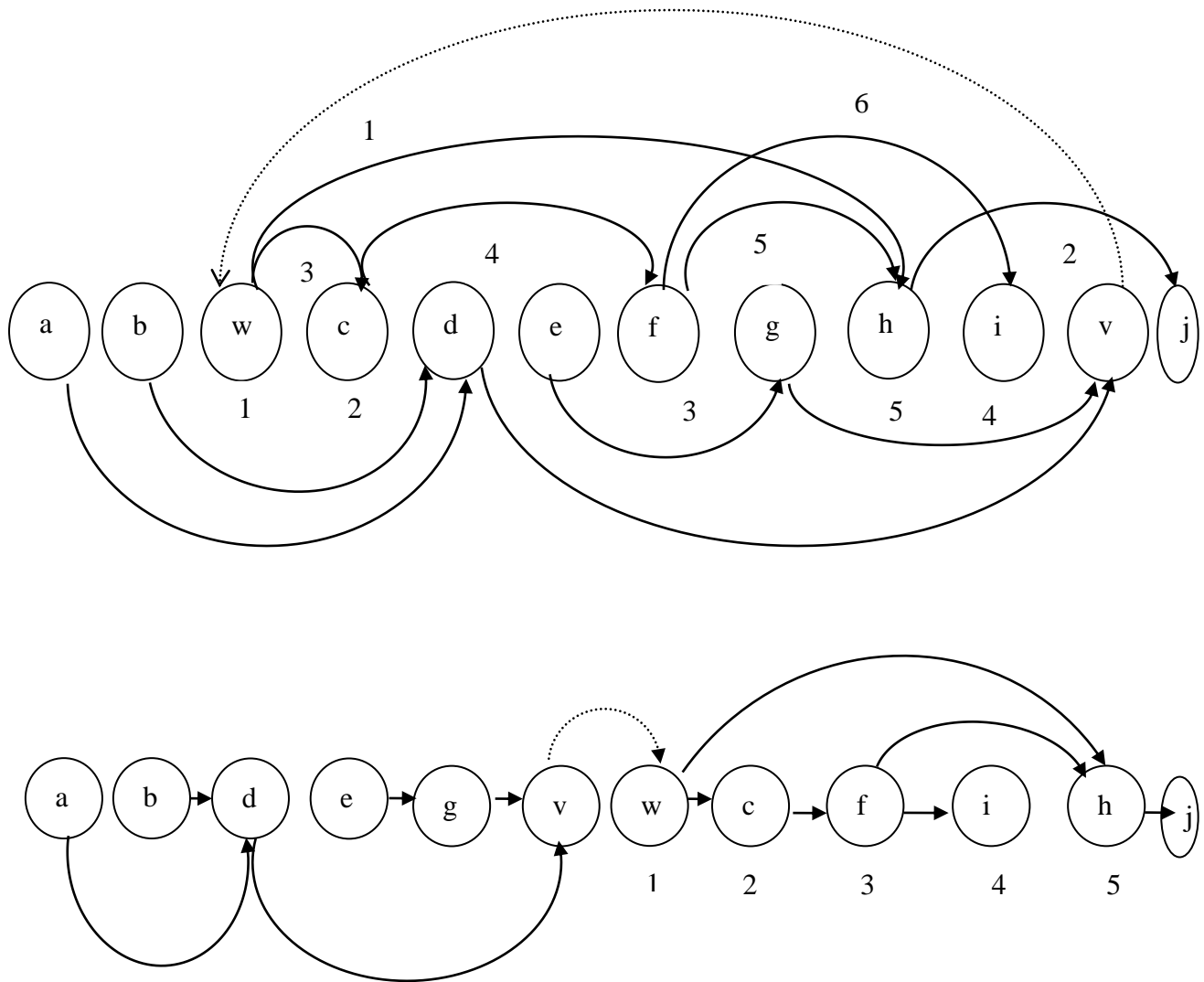
In the algorithm of LIMITED-SEARCH the set F denotes the number of nodes to be traversed by the search and the set A denotes the number of edges to be visited during search.

The search work as follow:

- An iteration of the while loop that deletes an arc  $(x, y)$  from  $A$  does a traversal of  $(x, y)$ . The edge to be selected from the set  $A$  is arbitrary.
- If the addition of  $(v, w)$  creates a cycle,  $\text{LIMITED-SEARCH}(v, w)$  returns an arc  $(x, y) \neq (v, w)$  on such a cycle; otherwise, it returns null.
- If it returns null, restore topological order by moving all vertices in  $F$  just after  $v$  (and before the first vertex following  $v$ , if any). Order the vertices within  $F$  topologically, for example by making the search depth-first and ordering the vertices in  $F$  in reverse postorder with respect to the search. Figure 2 shows an example of limited search and reordering.

The Figure 2.8 defines the limited search and reordering the topological order after the arc addition:





**Figure 2.8: Limited search followed by vertex reordering**

Initial topological order is left-to-right. Arcs are numbered in order of traversal where the search performed is depth first. Visited vertices are **w, c, f, h, i, j**. They are numbered in reverse postorder with respect to the search and reordered correspondingly.

#### 2.4.2 Analysis of time bound of one way search

The topological ordering is reorder in one way search and minimizes the time bound on search. Here set F and A are represented using the link list and mark the vertices as they added to the search. The total time for search is  $O(1)$  plus the  $O(1)$  per arc traversal. Only the last search, which does at most 'm' arc traversals, can detect a cycle.

To bound the total number of arc traversals, the notion of relatedness was introduced. A vertex and an arc to be related if some path contains both the vertex and the arc, and unrelated otherwise. This definition does not depend on whether the vertex or the arc occurs first on the path; they are related in either case. If the graph is acyclic, only one order is possible, but in a cyclic graph, a vertex can occur before an arc on one path and after the arc on a different path. If either case occurs, or both, the vertex and the arc are related.

If the addition of  $(v, w)$  does not create a cycle but does trigger a search then the path and vertices are unrelated. If  $(x, y)$  be an arc traversed during the (unsuccessful) search for  $v$  from  $w$ . Then  $v$  and  $(x, y)$  are unrelated before the addition but related after it.

The vertex arc pair related in searching method can be at most  $nm$ . And the total searching time for cycle including the last added arc will be  $mn+m$ . So the total running time of the search method is  $O(mn)$ .

Shmueli suggested the one way search method but did not give the detail description that how to accomplish the topological order. He merely give the hint about the numbering scheme which can be used to accomplish the reordering of topological order.

To do the reordering efficiently, the representation is more complicated than a simple numbering scheme. Hence the solution to the dynamic ordered list problem is defined: represent a list of distinct elements so that order queries (does  $x$  occur before  $y$  in the list?), deletions, and insertions (insert a given non-list element just before, or just after, a given list element) are fast. Solving this problem is tantamount to addressing the precision question that Shmueli overlooked and did not provided the solution.. Dietz and Sleator[30] gave two related solutions. Each takes  $O(1)$  time worst-case for an order query or a deletion. For an insertion, the first takes  $O(1)$  amortized time; the second,  $O(1)$  time worst-case. Bender et al. [34] simplified the Dietz-Sleator methods. With any of these methods, the time for reordering after an arc addition is bounded by a constant factor times the search time, so ' $m$ ' arc additions take  $O(nm)$  time.

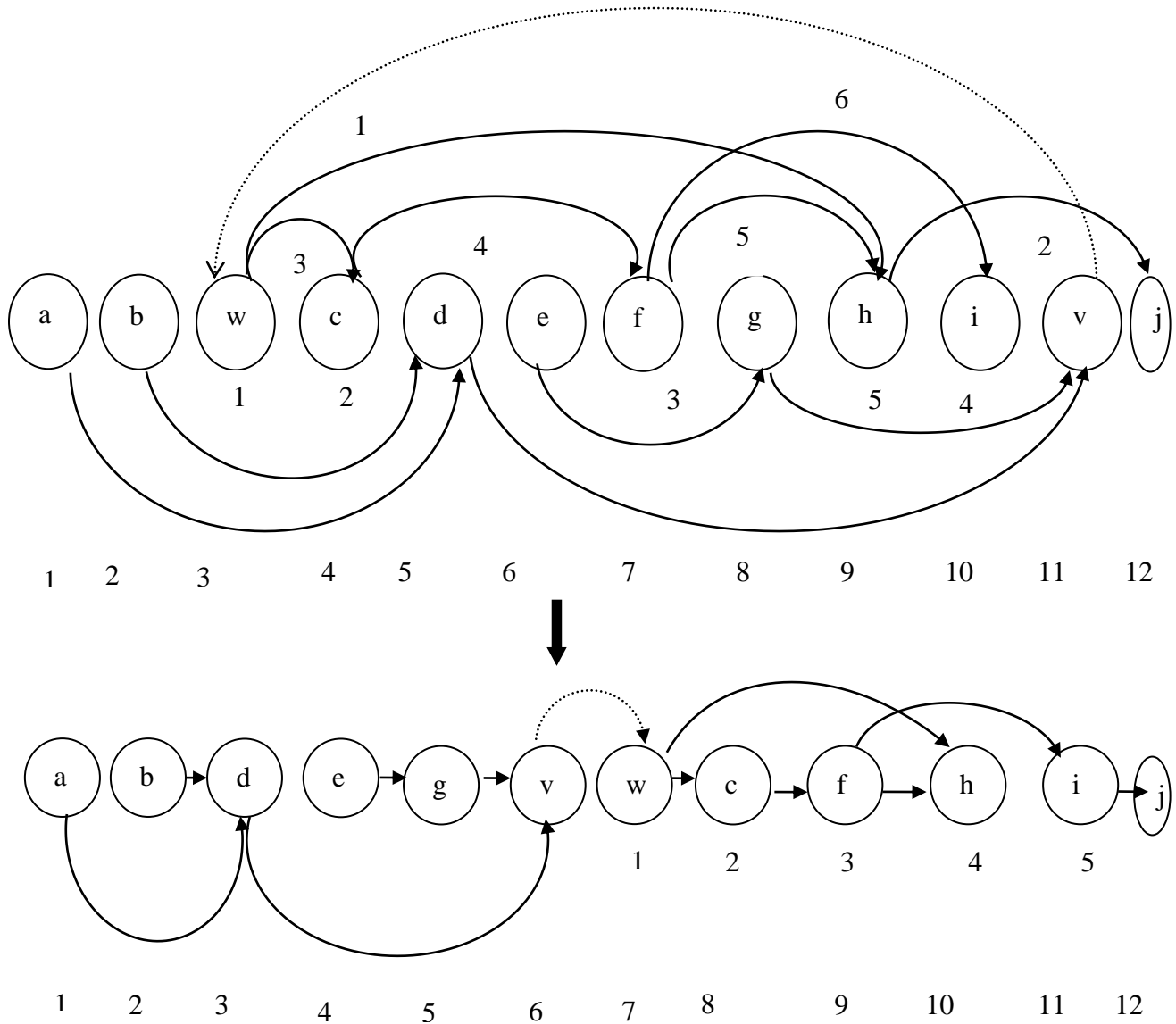
The one way search method suggests the simpler way to reordering the vertices which are between the vertexes ' $v$ ' and ' $w$ ' (both are inclusive) is defined as follow:

- Move all vertices visited by the search after all other affected vertices, preserving the original order within each of these two sets.

Figure 2.9 illustrates this alternative reordering method. The topological ordering algorithm is called local if it reorders only affected vertices. Except for Shmueli's unlimited search algorithm and the recent algorithm of Bender et al. [34], all the algorithms discussed are local.

The reordering can be efficiently even if the topological order is explicitly represented by a one-to-one mapping between the vertices and the integers from 1 through 'n'. The reordering time is  $O(n)$  per arc addition; the total time for  $m$  arc additions is  $O(nm)$ . The

Figure 2.9 next page shows the reordering example for a graph:



**Figure 2.9** the method of restoring topological order after a limited search of the graph in the previous Figure 2.8.

The number given to the vertices are according to the topological order. The vertices which are affected is **w,c,d,e,f,g,h,i,v**. Arcs are numbered in order of traversal. The affected vertices are reordered by moving the visited vertices **w,c,f,h,i** after the unvisited vertices **d,e,g,v**[36].

## CHAPTER 3

### PROBLEM STATEMENT

---

---

#### 3.1 Problem Definition

The previous chapter discussed the various dynamic graph techniques and provides the detail description of merits and demerits of each technique used for dynamic graph algorithm for undirected graph. The techniques are suited for the various applications and provide the better time bound on various graph properties according to the conditions. The thesis provide the comparative analysis of dynamic graph algorithmic techniques on various graph properties like,

- Minimum spanning forest
- 2-edge connectivity
- Bipartition of the graph
- And space utilization

The analysis compares the query time and updates time for all graph techniques for different graph properties. It provides the answer that which techniques performs better for different graph properties for update and query time.

The graph techniques have a large area of research. There is profound research on various techniques for different applications. One of the applications for dynamic graph algorithm is deadlock detection. The deadlock detection has undergone with a large amount of research and many algorithms has been provided in last decade that how to minimize the search time for cycle detection. But there are many issues related to the running time of deadlock detection. The issues related to the online recognition of cycles are as follow:

- Searching for whole graph for detection of cycle.
- The running time for dynamic graph take the poly logarithmic time bound, which take a lot of time and space.

- The algorithm search from the scratch for searching the cycle in the graph.
- The one way search method described in chapter 2 search for a cycle in forward direction for any added edge hence the minimum search time is  $O(nm)$ .

### **3.2 Proposed Objective**

The main objective of the thesis is to provide the solution for the above mention issues for cycle detection in dynamic graphs. The thesis answers to the following issues:

- The new two way search method is analyzed and discussed.
- The method is applied to the deadlock detection for searching of cycle and provides the full analysis of topological ordering benefits in two way search method.
- The new algorithm is proposed for deadlock detection using the two way search method.
- The proposed algorithm is compared and analyzed with the other deadlock detection algorithm.
- The algorithm is verified with the proof of correctness.
- The examples are given to provide the explanation for the proposed algorithm.

# COMPARITIVE ANALYSIS OF DYNAMIC GRAPH TECHNIQUES

---

---

### 4.1 Comparative analysis of dynamic graph techniques for undirected graph

The following comparative analysis is performed on undirected graph  $G(m, n)$  by using various dynamic graph algorithmic techniques. These properties are needed to be intact whenever there are changes in the graph when the updation and query operation take place. The brief detailing of these properties are defined as follow:

#### 4.1.1 Minimum Spanning Forest

The spanning forest is a sub graph which represents the spanning tree of connected component of the graph containing no cycle and covers each vertex in the graph. Maintaining the spanning tree of the graph is a major requirement to answer about the queries whether the graph is connected or not. The property answers the minimum path to the weighted link that, which path is shortest in the communication network. The minimum spanning forest algorithm starts with the 'n' vertices and with no edges, maintain the minimum spanning forest in polynomial time for edge insertion and deletion. The lower bound varies for various dynamic graph algorithmic techniques as shown in the table 4.1.

#### 4.1.2 2-Edge connectivity

The graph is known to be  $k$  - edge connected when by deletion of fewer edges than  $k$  is removed from the graph, and the graph remains connected. The any tree edge is said to be covered if it is a part of the non-tree edge  $(v, w)$  and is a part of cycle induced by the non-

tree edge  $(v, w)$ . So  $e$  is treated as a non tree edge until it is not used by any non tree edge. Since 2-edge connectivity is a transitive relation on vertices, it follows that two vertices  $x$  and  $y$  are 2-edge connected if and only if they are connected in set  $A$  and all edges in  $x \dots y$  are covered [Frederickson 1997]. Hence 2 edge connectivity is checked for any set of vertices whether they are 2- edge connected or not.

### **4.1.3 Bipartiteness**

When insertion and deletion of edges take place in the graph the various graph properties are checked. The bipartiteness is also checked during the updated of the graph. The graph is checked whether it remain bipartite after each graph updates[. This property of graph gives the better time bound in randomized technique. The other techniques give larger time bound on graph.

### **4.1.4 Space**

The techniques used with different graph tool and data structure take different time bound for space usage. The space usage depends upon the data structure used with the graph techniques. Space requirement are different for each graph technique.

The following comparative analysis if performed on undirected graph  $G(m, n)$  is using various dynamic graph algorithmic techniques. Here  $k$  is the logarithmic of  $n$ . The table gives the insight about the time taken by various algorithmic techniques to preserve the graph properties. Based on this comparative analysis, the Table 4.1 is made on the next page which is as follow:



**Table 4.1 Comparative Analysis of Update time**

<b>TEECNIQUES</b>	<b>CLUSTERING</b>	<b>SPARSIFICATION</b>	<b>RANDOMIZATION</b>
<b>UPDATE TIME</b>			
<b>MINIMUM SPANNING FOREST</b>	$O(k^3)^*$	$O(n^{1/2})$	$O(\log^3 n)$
<b>2-EDGE CONNECTIVITY</b>	$O(mk^3)$	$O(n^{1/2})$	$O(\log^3 n)$
<b>BIPARTITION OF THE GRAPH</b>	$O(m^{1/2})$	$O(n^{1/2})$	$O(\log^3 n)$
<b>SPACE</b>	$O(m \log n)$	$O(m \log n)$	$O(m+n \log n)$

The Table 4.2 next page compares the query time for various dynamic graph algorithmic techniques on different graph properties like minimum spanning forest, 2- edge connectivity and on bipartitions. It gives the answer to various queries whether the graph is connected or not and answer the query whether there is path from node i to j.

**Table 4.2 Comparative Analysis of Query time**

<b>TEECNIQUES</b>	<b>CLUSTERING</b>	<b>SPARSIFICATION</b>	<b>RANDOMIZATION</b>
<b>QUERY TIME</b>			
<b>Minimum Spanning forest</b>	$O(1)$	$O(1)$	$O(\text{Log}^3 n)$
<b>2-edge connectivity</b>	$O(\log n)$	$O(\log n)$	$O(\log n / \log \log n)$
<b>Bipartition Query</b>	$O(\log n)$	$O(n^{1/2})$	$O(1)$

Based on the above analysis this has been analyzed that dynamic graph algorithm work well on different graph algorithm techniques using different data structure tools.

The clustering technique has the following advantages over other techniques:

- The technique work better with space and can be used as a black box for many application.

The sparsification technique has been researched for a long time and provides the better time bound as compared to clustering:

- The technique provides the answer to queries about connectivity in less amount of time.
- The techniques answer well for the small updates sequences.
- And has a same space usage as clustering techniques but depend upon the mount of update.

The randomized algorithm works well for all types of update sequence and answer the query about the connectivity. The dynamic algorithm work with randomize technique perform well on the following properties:

- The dynamic graph algorithm with randomized technique performs better time bound as compared to other techniques.
- The technique answers that weather the graph is bipartite in better lower bound than sparsification and clustering technique.
- The query time is also minimized using this technique in various applications.

### PROPOSED DEADLOCK DETECTION ALGORITHM

---

---

The proposed online algorithm for deadlock detection maintains the acyclic property of the  $n$ -vertex directed graph when the new edge is added in the graph. The time bound for the incremental cycle algorithm for deadlock detection take  $O(\min\{m^{1/2}, n^{2/3}\}m)$  time bound for the 'm' edge insertion in the directed graph. It report the cycle when the algorithm detects for edge  $(v,w)$  that there exist a path form vertex 'w' to 'v'.

The algorithm defined here maintains the topological order also if the cycle is not detected and the graph is acyclic. The topological order of the direct acyclic graph is a sequence of vertices such that for every edge  $(v, w)$ ,  $v < w$ . In the previous topological ordering algorithm the cycle can be detected in  $O(m+n)$  time by either using the depth first search method [40] or by recursively deletion of the vertices.

As for some deadlock problem the graph is not fixed and the new edge added in the graph, the incremental cycle detection detect the cycle when the new arc is added and maintains the topological order of the graph.

The algorithm define here work better for cycle detection and finding the topological ordering for the arc addition than running the algorithm from scratch for the static graph. The assumption is made that the vertex set is fixed and initially the edge set is empty. Here 'n' denotes the number of vertices and 'm' denotes the number of arcs added. The idea uses in the algorithm is the two way search or compatible search.

## 5.1 Two way Method for Cycle Detection

The vertex order defines the topological order of the vertices. The vertex order can be maintain using heap or dynamic list data structure[38]. The vertex order testing can be performed whether  $v < w$  using these data structure in  $O(1)$  time. Deleting the vertex and reinserting it in proper order take the same bound.

The two way search work as follow when the new arc  $(v, w)$  is added,

- when new arc is added then add  $(v, w)$  to the set of edges going out of  $v$ , and add to the set of edges going into  $w$ . if  $v > w$  then the searching will work forward from  $w$  and start searching backward from  $v$  until the cycle is detected.
- If cycle is not found then restore the topological ordering of vertices.

A set of vertices is forward if there exists a path from  $w$  to  $y$  of the arcs that has to be traversed forward.

The set of vertices are backward if there exists a path from  $y$  to  $v$  of arcs traversed backward.

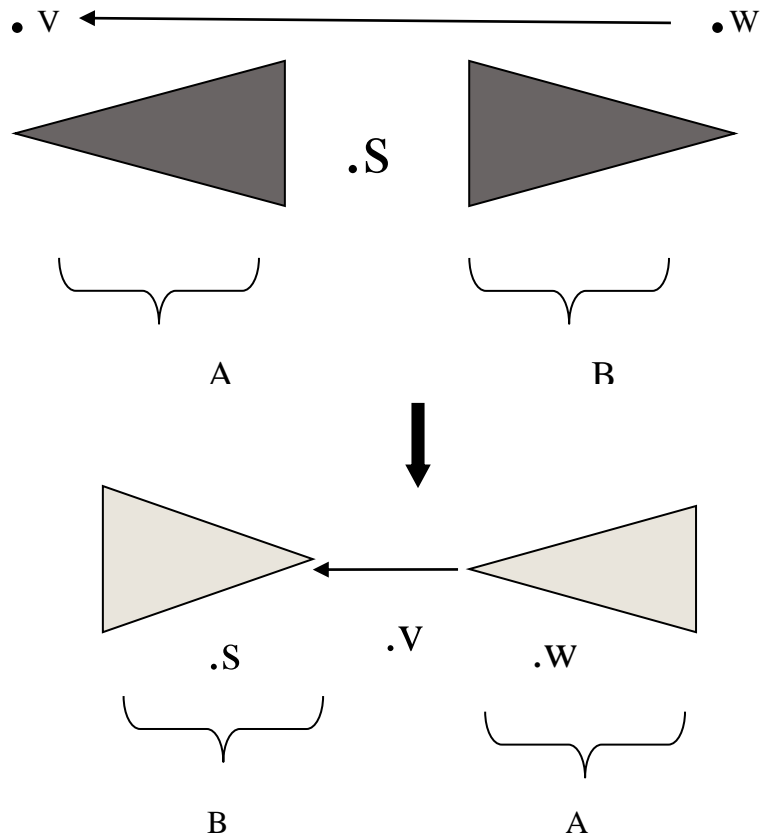
To start the search,

- All the arcs are visited from the forward vertex set.
- And all the arcs are traversed form the backward vertex until the forward visited arc reaches the backward vertex  $y$  or a backward traversal reaches a forward vertex  $y$ .

If any of the above case is true, then there exist a cycle or there exist a vertex  $x$ , such that all forward vertices which are smaller than  $s$  and all of the backward vertices which are greater than  $x$  are scanned.

If in the cases above there exist a path then report it, consisting of a traversed path from vertex  $w$  to  $y$  and from vertex  $y$  to  $v$  visited backward and followed by the newly created edge  $(v, w)$ .

The Figure 5.1 next page defines the two way search:

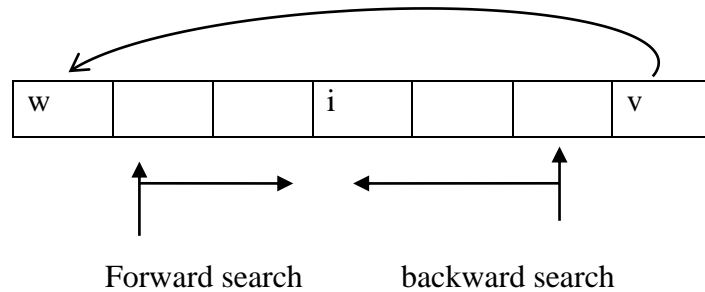


**Figure 5.1 topological order reordering**

In the above figure the set A is a collection of forward vertices which are smaller than  $y$  and set B is a collection of backward vertices which are greater than 's'. The method find out the topological order of the set A and B respectively and delete the vertices in  $A \cup B$  from the current topological ordering and reordering then after 's', in topological order of B followed by topological order of A. The method required  $O(1)$  time bound per arc visited[39].

In incremental cycle detection only the last edge added will create the cycle hence the last search take  $O(m)$  time overhead.

The next Figure 5.2 shows that cycle will be detected if the forward and backward search meets at any index  $i$ .



**Figure 5.2 forward and backward searches meet at index  $i$ .**

Before defining the algorithm the two way search for cycle detection has the following assumptions:

- The two way search involves the forward traversal arc  $(w, x)$  and backward traversal of arc  $(y, v)$ .
- The search work with the compatibility means forward and backward vertices should be in topological order before starting the search. (Before the addition of the vertex  $(v, w)$ ).
- Every vertex should be in the following three states before applying the algorithm for cycle detection:
  - 1) Unvisited
  - 2) Forward (visited by the forward search)
  - 3) Backward (visited by the backward search)
- Before any arc is added, the entire vertex set must be in unvisited state.
- $A$  and  $B$  is the set of forward and backward vertices.
- If the search does not detect the cycle, then some vertices  $A \cup B$  must not be in topological order, and need to be reordered.
- The search maintain the set of vertices  $F_A$  that are to be traverse forward, and the set of vertices  $B_B$  that are to be traversed backward.

- If the search result in some arc other than (v, w) then there exist a cycle, else algorithm will result null.

The rules for the algorithm are as follow:

**DeadlockDetectionfunction (v, w)**

- $A=\{w\}$  ,  $B=\{v\}$ ;  $F_A=\{(w ,y)/ \text{ where } (w, x) \text{ is the edge}\}$ ,  $B_B= \{(x ,v) / \text{ where } (x ,v) \text{ is an edge in the graph}\}$
- **For** there exist (u ,y)  $\notin A$  and (x ,z)  $\notin B$  where (u<z) then **do**  
 Select (u, y) and (x, z) from set  $F_A$  and  $B_B$  where (u<z)  
 Remove the edge (u, y) and (x, z) from set  $F_A$  and  $B_B$ .  
     **If** y  $\notin B$  **then return (u, y)**  
     **Else if** x  $\notin A$  **then return (x, z)**  
 (Report cycle including edge (u, y) or (x, z)  
     **If** y does not belong to A **then**  
      $A= A \cup \{y\}$ ;  $F_A= F_A \cup \{(y, s)/ \text{ where } (y, s) \text{ is an arc in the graph}\}$ .  
     **End**  
     **If** x does not belong to B **then**  
      $B= B \cup \{x\}$ ;  $B_B= B_B \cup \{(r, x)/ \text{ where } (r, x) \text{ is an arc}\}$   
     **End**  
**End**
- **Return NULL.** ( does not create any cycle)

The algorithm defines the two way compatible search for cycle detection. The choice to which pair the arc to be traverse is arbitrary until the search is compatible. The loop searches for the loop creating edge when the new arc (v, w) is added to the directed acyclic graph [37].

If the function returns the null means there exist no loop in the graph and restore the topological order of the graph as some of vertex in  $A \cup B$  to be placed in their vertex order as defined in one way search.

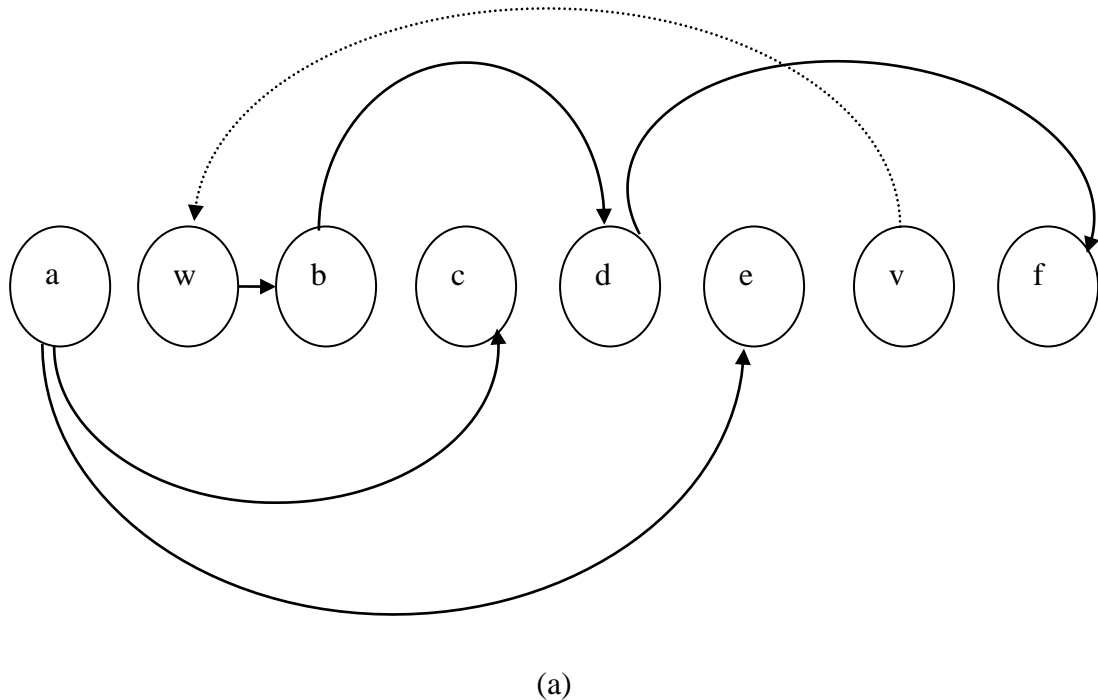


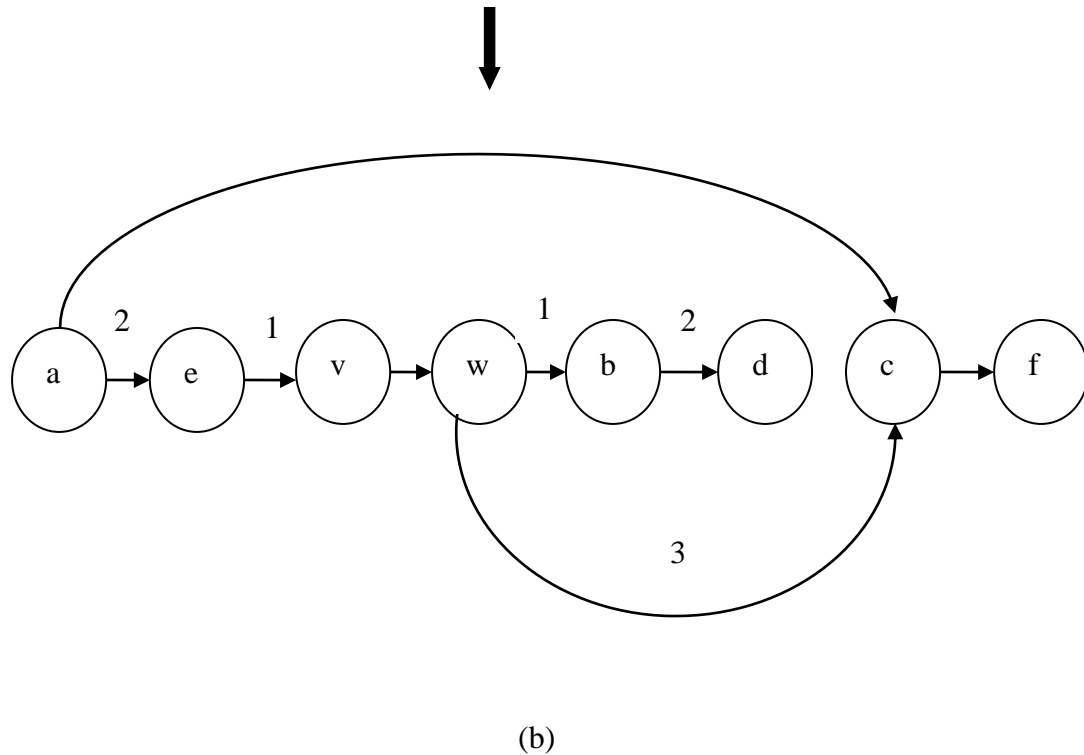
The two way search find out the cycles and reorder the vertices in topological order if cycle does not exist. The efficiency of two ways search depends upon the number of arc visited in the cycle detection. For m arc addition the time bound is  $O(m^{3/2})$ . The implementation can be filled in either using the priority queue data structure or dynamic order list.

Though the addition of the edge (v, w) start the search step but not necessarily detect the cycle, in this case the forward edge (w, y) and backward edge (x, z) are unrelated before the edge addition but related after the addition.

### 5.1.1 Example of two way search

The example Figure 5.3 below defines the working of this function on the acyclic graph:





**Figure 5.3 Two way function example**

The figure 5.3 above shows the example of search method when the new edge (v, w) is added, showing the two way search method and restoring the topological order (b) of the vertices so that only last edge added will detect the cycle.

Here initially set A contain vertex {w}, and set B contain the vertex {v}. The traversed edge pair are for forward direction is {w, b, d} and for backward traversal is {v, e, a}.

After the search the edge set of forward vertices will contain {(w, c), (b, d), (d, f)} in the set  $A_F$ . The index chosen is  $t=d$  which can be any arbitrary node. The forward vertices less than the index is {w, b} and backward vertices greater than {v, e}. To retain the topological order of the vertices all forward vertices less than t will be moved before t and all the backward vertices greater than t will be moved before the forward vertices smaller than t. The topological is implemented using dynamic ordered list.

### 5.1.2 Example graph detecting Cycle

Another example elaborates the algorithm defined above and reports the cycle. The edge  $(v, w)$  is added in the direct acyclic graph and only the last added edge will report to the cycle if exist. The figure 5.4 below has a topological order of  $a, w, b, c, d, e, v, f$  before adding the edge  $(v, w)$  in the graph.

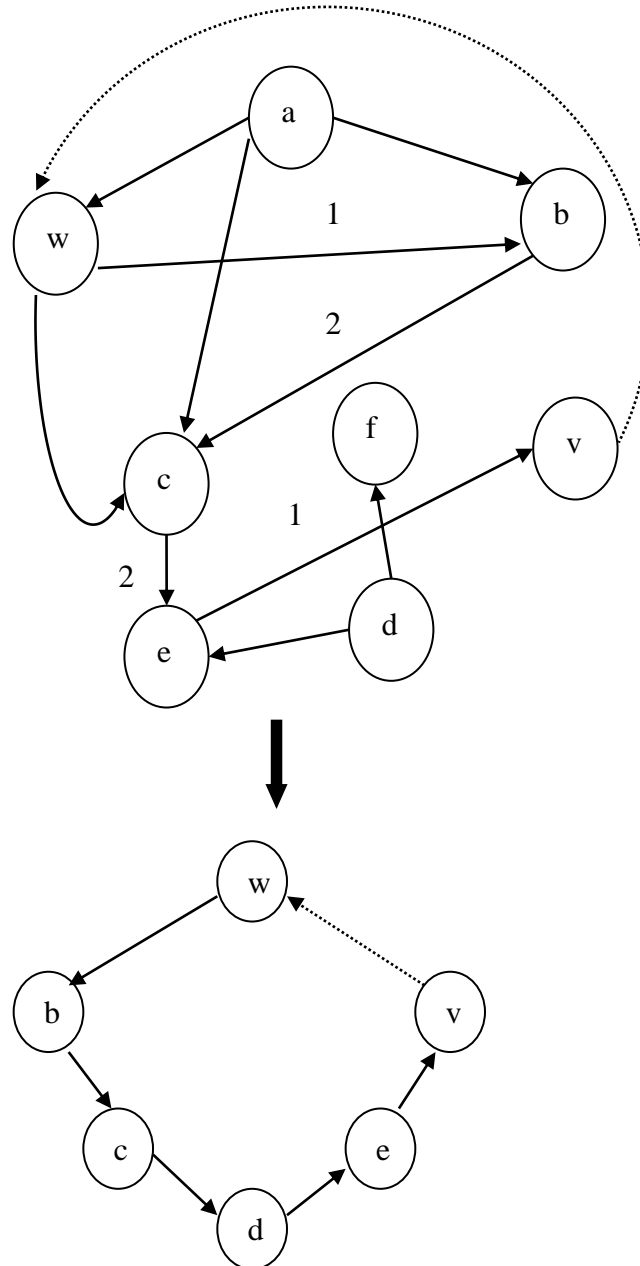


Figure 5.4 report the cycle.

The above example works on the algorithm and detects the cycle. The initial values of the forward and backward sets are as follow at each step:

**Step 1:**  $A = \{w\}$ ,  $B = \{v\}$  are the forward and backward set of vertices for the newly added edge  $(v, w)$ . and the set of forward edges will contain  $F_A = \{(w, b), (w, c)\}$  and the backward set of edges will contain  $B_B = \{(e, v)\}$ . Now the forward and backward edges will be traversed together if they are compatible. The search is **compatible** if the forward edge  $(u, y)$  and the backward edge  $(x, z)$  are traverse in the same step if  $u < z$ . The forward edge  $(w, b)$  and  $(e, v)$  are traversed in the same step and the order of compatible edges are arbitrary.

**Step 2:** After first step the forward vertex set A and backward vertex set B will contain:

$$A = \{w, b\} \text{ and}$$

$$B = \{v, e\}$$

And the forward edge set and the backward set of edge will contain the following edges:

$$F_A = \{(w, c), (b, c)\} \text{ and}$$

$$B_B = \{(d, e), (c, e)\}$$

The forward and backward vertex set does not contain any common vertex in the sets ,so the search continue until the edge set is null or the cycle is detected and reported.

**Step 3:** Now the forward edge  $(w, c)$  and the backward edge  $(d, e)$  are traversed as the two edges are compatible because  $(w < e)$  in the previously restored topological order.

The new forward vertex set and backward vertex set will contain

$$A = \{w, b, c\} \text{ and}$$

$$B = \{v, e, d\}.$$

The newly added forward edge and backward edges in the sets are:

$$F_A = \{(b, c), (c, e)\}$$

$$B_B = \{(c, e)\}.$$

The forward and backward vertex set still not contain the common vertex in both the sets so the search continues until there exist the untraverse edges in the forward and backward vertex edge sets.

The next edges select are  $(b, c)$  from forward edge set and  $(c, e)$  from backward edge set.

**Step 4:** The new vertex set are:

$$A = \{w, b, c\} \text{ and}$$

$$B = \{v, e, d, c\}$$

And the forward and backward edge set contain

$$F_A = \{(c, e)\} \text{ and}$$

$$B_B = \{\text{null}\}.$$

In this step the backward vertex set is null and edge is reported for creating the cycle. The vertex set A and B both contain the vertex 'c' .and hence the edge (b, c) is reported from the algorithm for creating the cycle. The algorithm will be used to update the topological order.

**Theorem:** The algorithm correctly detect the cycle in  $O(m^{1/2})$  arc traversal as the edge (v, w) is added to the graph.

Proof: The addition of the edge will take  $O(1)$  time bound and  $O(1)$  time bound to traverse any arc. When the new edge is added the initiation of the search will take the additional overhead. When the graph changes due to insertion of edge the pair of related elements is counted. Initial value to related pairs is zero and increases as the search starts for the cycle. For related elements there are three possibilities vertex – vertex pair, arc-vertex pair and arc- arc pairs. The maximum number of vertex to vertex pair will be less than  $n^2/2$  and the arc to vertex pair can be  $nm$  and arc to arc pair will be less than  $m^2/2$ . The arc to arc pairs is used for searching the graph. The last search takes  $O(m)$  time overhead.

The time bound used by two ways searching is better than one way search but space requirement can be minimized if used with balanced approach. The space usage is  $O(m)$  for searching algorithm for cycle using the link list implementation. The search picks up the two edges concurrently one forward and one backward so time is minimized but with the additional condition i.e if (u, y) is the forward edge to traverse and (x, z) is the backward edge traverse then  $(u < z)$  condition must be satisfied ,only then the search will be **compatible** [20].

The count is made on arc to arc pairs search. Let the total number of arc traversed during the search method for cycle is  $2i$ , where  $i$  edges are traverse forward and  $i$  edges are traverse backward. If  $(u, y)$  is the  $(i/2)^{\text{th}}$  edge that is being traverse then there is a arc must be related to newly traverse backward edge and the edge preceding it  $(u, y)$ , hence the total number of minimum  $(i/2)^2$  arc –arc related pair in search.

Two searches are possible:

- 1) First takes minimum of  $m^{1/2}$  edge searching for the cycle detection. The search is small.
- 2) Second one, the big search take more edge traversal in search method restricted to the affected region, which is in between the vertex  $v$  and  $w$ . Where  $(v, w)$  is the newly added edge. If  $2i_k$  is the total number of arc traverse in the  $k^{\text{th}}$  search method then ,

$$2i_k > m^{1/2}$$

The total number of related arc pair to single edge is  $(i/2)^2$  then the sum of all arc pair related to each other in the total search is,

$$\sum_k (i_k/2)^2 < m^2/2$$

The big search involves the total of  $2i$  edge traversal and by adding the newly created edge the total number of related arc pair will be grown up by,

$$i(i+1)/2 > i m^{1/2}/2$$

Where the  $m^2/2$  is the total number of related arc pairs hence the total edge traverse during the second search is  $O(m^{1/2})$ .

Previously there are heap structure to use the compatible search, one priority heap to store the forward arcs and one priority heap to store the backward arcs. The total time bound then increased by  $O(m^{1/2} \log n)$ . Where  $(\log n)$  is the heap overhead and can be minimized by using the dynamic ordered list or linked list data structure.

## 5.2 IMPLEMENTATION

The most deadlock detection algorithms find out the deadlock using the cycle detection method. The previous work on these methods [16] uses the dynamic ordered list, but the complexity for implementing the dynamic ordered list is high. The search techniques describe above uses the random sampling for selection the forward and backward vertices. The implementation here define does not uses the dynamic ordered list but uses the tree terminology for simplicity. The pair is here order lexicographically: if  $a, b < c, d$  then it must be  $a < b$  or  $a = b$  and  $c < d$  or  $c = d$ .

The vertices and arc are related to each other if they belong to the common path, or mutually related if they are the part of same cycle else the vertices and arcs are not related to each other if they are not on the common path. The static algorithm takes  $O(E+V)$  time bound to detect the cycle, but in dynamic environment it takes  $O(m^{3/2})$  time for sparse graph for detecting the cycle. It takes  $O(n^{5/2})$  time for detecting the same in dense graph due to [15].

The simple method to detect the cycle is to maintain the vertices in tree levels. Each level contains the vertices and maintains the indices of vertices to represent the topological order. When the search method initiated the backward edges is traverse within the level and if the backward edges traverse the maximum number of vertices then the level is incremented. The forward search traverses the edges that belong to the lower level. Each time the level of visited vertex is increased. The tree maintains the level and indices of each vertex. The vertex  $v$  level is denoted by  $L(v)$  and indices denoted by  $k(v)$ . The indices define the topological order of the vertices if no cycle is detected. For an edge  $(v, w)$  the **out(v)** shows the degree of outgoing edges from vertex  $v$ . Here the use of singly link list is taken to store the information for outgoing edges. The **in(w)** represent the incoming degree of the vertex  $w$ . If the forward and backward search does not find out the cycle then the indices on the vertices will represent the topological order.

The steps involves for inserting the new edge  $(v, w)$  is first checking the order whether the edge is in topological or not. The steps are as follow:

**Step 1:** If  $L(v), k(v) < L(w), k(w)$ , then the vertices are already in the topological order, or no search is needed.

**Step 2:** Initially backward set  $B = \text{NULL}$  and forward set  $F = \text{NULL}$  and the edges set is empty.

The maximum number of arcs visited is  $\text{max} = \{\text{min}(\mathbf{m}^{1/2}, \mathbf{n}^{2/3})\}$ . Now visit backward vertices on the same level until maximum edges are visited. DO visit Backwardvisit (v).

**Backwardvisit (x)**

- Mark the vertex x.
- While  $(s, x) \in \text{in}(x)$ 
  - DO Backwardtraverse(s, x)
    - 1) **If**  $s = w$  **Then** report a cycle and stop the search.
    - 2) Increment the arc visited
    - 3) **If** arc visited  $\geq \text{max}$  **then** stop the search as maximum number of arcs is visited.
      - $L(w) == L(v) + 1,$
      - $\text{In}(w) == \text{NULL}$  unmark all the vertices and go to step 3.
    - 4) If s is unvisited
      - Then backwardvisit(s)
- $B = \text{BU}[x].$

If cycle is reported in the step 2 then test  $L(v) == L(w)$  if so then visit the step 4.

**Step 3: (forward search):** now visit forwardvisit (w) and traverse the forward edges.

Forwardvisit(x)

**While**  $(x, y) \in \text{out}(x)$

**Do** ForwardTRAVERSE(x, y)

- 1) **If**  $y = v$  or y is in B
  - Then** stop the algorithm and report the detection of a cycle
- 2) **If**  $L(y) < L(w)$ 
  - Then**  $L(y) = L(w)$
  - $\text{In}(y) = \text{NULL}$
  - Forwardvisit(y)
- 3) **If**  $L(y) == L(w)$  then add (x, y) to  $\text{in}(y)$



**Step 4:** Reindex the values on the nodes in case there is no cycle.

**Step 5 (insert arc):** Add  $(v, w)$  to out  $(v)$ . If  $L(v) = L(w)$ , add  $(v, w)$  to in  $(w)$ .

The steps defined above search for the newly added edge and report the cycle if it exists. If there is no cycle in the search procedure then at each level the indices of each vertex define the topological order of the vertices. If level of vertices  $v$  and  $w$  are  $L(v) < L(w)$  and indices  $k(v) < k(w)$  then the addition of edge  $(v, w)$  will not create any cycle and all the vertices will remain in their topological order.

If by addition of the edge trigger the search and there exist a pre existing path from  $w$  to  $v$  then addition of the edge will create the cycle. In this case Step 2 will start the backward search and report the cycle or any different cycle containing the path  $w$  to  $v$ . If the level of vertex ' $v$ ' is same as the level of vertex ' $w$ ' then the node exist between the vertices will lie in the same level. And to break the cycle the level of vertex  $w$  will be increase and the step 3 will be followed. If the added edge has a vertex  $v$  in lower level than vertex  $w$  then the vertex  $w$  will increase its level in step 2 and will do the forward search in step 3 and report the cycle if one exists. The forward searching in the tree level will check for every edges such that if there exists a arc  $(y, x)$  and  $x=v$  and the vertex is the part of set  $B$ . Hence the algorithm in this tree format will search for the edges and report the cycle if exist in the forward and backward search. Else the lexicological order at vertices in each level represents the topological order. The space needed by the search algorithm is  $O(m)$ .

**Theorem:** When new edge is added the time required for insertion is  $O\{\min(m^{1/2}, n^{2/3})m\}$ .

The time required by the tree data structure uses the polynomial time  $n$  for level and indices. The addition and deletion of edges take  $O(1)$  time bound to traverse the arc. When forward search is traversed the increase of level take palce for one of the vertices in  $O(1)$  time and the backward search take place in minimum of  $(m^{1/2}, n^{2/3})$  time only. The forward search takes the  $O(1)$  time for edge traversal to initiate the search and to increase the vertex level to the next level.

For vertex set F for storing the forward visited vertices and B for storing the backward visited vertices the use the doubly link list is done to store the information. The counter bit can be used to find out whether the vertex belongs to the set or not. The time taken by this will be equivalent to  $O(1)$ .

### **5.3 DISCUSSIONS AND CONCLUSION**

The deadlock detection is the most open problem and a lot of research has been done to find out the algorithm to avoid deadlock in distributed and centralized environment. The thesis presented the new deadlock detection techniques using the compatible search method and provides the good time bound for finding out the cycles in the graph. The algorithm presented in section 5.1 works better for the sparse graph where there is less number of arc addition and search method take less time as compare to a denser graph. The algorithm presented in section will takes more space compare to linear search in which search only follow in the forward direction. The method takes  $O(m)$  space for the graph search for cycle detection.

The incremental cycle detection method has a number of advantages as the method work for dynamic environment as the arc added in the graph. In static environment the graph has to recompute the whole function from the scratch. The technique can work better in communication environment where the path diffusion techniques are used to find out the cycle in the graph. The usage of link list provide the better time bound on arc traversal as compared to heap which take the extra time of  $O(\log n)$  for  $m$  arc traversal. The technique used here maintain the arc – arc pair to obtain the search method , here the work can be done on vertex- vertex pair where the vertex guided search can perform the task for searching method. Here the algorithm shown has this time bound only for the adding the graph edges but does not has this time bound for deletion of edges at the same time Because maintaining the topological order needs the vertex reordering which takes the whole graph traversal to reorder the vertices. The algorithm can be modified where the search can be ordered and the arc selection is not arbitrary.

## CHAPTER 6

# CONCLUSION AND FUTURE SCOPE

---

---

Dynamic graph algorithm is the vast area of research for directed and undirected graphs. Where there are fully dynamic graph algorithms and partially dynamic graph algorithms for various applications. The undirected graph maintains the various properties on graph like spanning forest, bipartiteness and planarity. The directed graph algorithm maintains the transitive closure and spanning forest on the graph.

Throughout the literature survey, an attempt is made to present all the algorithmic techniques within a unifying framework by abstracting the algebraic and combinatorial properties and the data structural tools that lie at their foundations. Section 2.1 discussed the various dynamic graph techniques and provided with the merits and demerits of the techniques. Section 2.2 discussed the tools and data structure which lays at the foundation of these dynamic graph techniques. These techniques have an immense scope and are used in one of dynamic graph applications. Section 2.3 discusses the previous deadlock detection techniques and provide with the running time of algorithms. Deadlock detection can be solved using the one way search method which is examined in section 2.4 retaining the topological order of the vertices. The method has a higher time bound because the possible search is forward only.

Section 4.1 provide the comparative analysis of all the dynamic graph techniques and provide the profound insight of update and query time for the various dynamic graph properties like spanning tree, bipartition and edge connectivity.

In section 5.1 the new method is proposed for the cycle detection for deadlock application. The example is being presented for providing the elaborate usage of the new two way method for cycle detection. The new compatible search method provides the better time bound for online dynamic graph search for deadlock.

## Future Scope

The algorithm work better for the sparse graph where search method take less time when  $m = O(n)$  and report the cycle when detected.

- The new techniques can be developing with incremental addition of edges and with the deletion of edges from the graph. As the process leave the resource the edge is deleted, and when the process demand for the new resource the edge is inserted in the graph. Hence there are scope for the algorithms and techniques which perform both the task simultaneously.
- The reordering of vertices take the additional time in incremental cycle detection so there are scope where reordering can be done with weights.
- The work presented in the section 5.2 can be improved using the better data structure and improving the algorithm with arc deletion also.
- There is scope in cycle detection algorithm for amortized time bound for finding out the topological order of the graph if the graph is acyclic and maintain the strongly connected component for the graphs.

This bulk of recent work has raised some new and perhaps intriguing questions.

Are there any general techniques for making increase-only algorithms fully dynamic. Furthermore, no randomized algorithm is known for fully dynamic maintenance for shortest path. Future work can be done for finding out the randomized algorithm for faster solution. On the practical side, it would be interesting to push these designs into real systems and real deployments. The sybil attack in distributed systems refers to individual malicious users joining the system multiple times under multiple fake identities. Beyond sybil attacks and beyond social networks, the insights on attack edges, cuts, and mixing time may find applications elsewhere. For example, these insights might apply to PageRank and help it to be robust against sybil webpages. One could also imagine detecting email spams based on the email graph and its connectivity property.

## References

---

- [1] Giuseppe F. Italiano and Irene Finocchi. “Dynamic graphs” DISP, Università di Roma “Tor Vergata”. IST-1999-14186 (ALCOM-FT).
- [2] M R Henzinger and v.King. “Fully Dynamic Biconnectivity and transitive closure”. operation. In proc. Proc. 36<sup>th</sup> IEEE Foundation of Computer Science, 1995.
- [3] Jacob Holm and Mikkel Throup. “ Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity” AT&T Labs—Research, Florham Park, New Jersey.
- [4] Alberts, D., Cattaneo, G., and Italiano, G. F. 1997. An empirical study of dynamic graph algorithms. ACM J. Exper. Algorithmics.
- [5] C. Demetrescu. “Fully Dynamic Algorithms for Path Problems on Directed Graphs”. PhD thesis, Department of Computer and Systems Science, University of Rome “La Sapienza”, February 2001
- [6] Giuseppe F. Italiano and Camil Demetrescu “Dynamic Shortest Paths and Transitive Closure: Algorithmic Techniques and Data Structures “, Dipartimento di Informatica e Sistemistica Università di Roma “La Sapienza”, Roma, Italy.
- [7] S. Baswana, R. Hariharan, and S. Sen. “Improved decremental algorithms for transitive closure and all-pairs shortest paths”. 34th ACM Symposium on Theory of Computing (STOC’02), pages 117–123, 2002.
- [8] U. Zwick. “All pairs shortest paths in weighted directed graphs—exact and almost exact algorithms”. In 39th Symposium on Foundations of Computer Science (FOCS), pages 310–319, 1998.

- [9] Valerie King. “Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs” Department of Computer Science University of Victoria.
- [10] Greg N. Frederickson. “Data structures for on-line updating of minimum spanning trees, with applications”. SIAM J. Comput. 14(4) (1985), 781–798.
- [11] David Eppstien and Zvi Galil. “Sparsification–A Technique for Speeding Up Dynamic Graph Algorithms”, University of California, Irvine, Irvine, California .july 1997
- [12] Monika R. Henzinger and Valerie King. “Randomized fully dynamic graph algorithms with polylogarithmic time per operation”. J. ACM 46(4) (1999), 502–516.
- [13] Kuldeep Sharma, Deepak Garg "Randomized algorithms methods and techniques" International Journal of Computer Applications IJCA pp. 29-32 Vol 28 Number 11 August 2011.
- [14] C. Demetrescu. Fully Dynamic Algorithms for Path Problems on Directed Graphs. PhD thesis, Department of Computer and Systems Science, University of Rome “La Sapienza”, February 2001.
- [15] T. Reps and G. Ramalingam. “On the computational complexity of dynamic graph problems”. Theoretical Computer Science A, 158:233–277, 1996.
- [16] Fredman, M., and Henzinger, M. R. 1998. “Lower bounds for fully dynamic connectivity problems in graphs”. Algorithmica 22, 3, 351–362.

- [17] Hezinnger, M. R., And king, V. 1997b. "Maintaining minimum spanning trees in dynamic graphs". Lecture Notes in Computer Science, vol. 1256. Springer-Verlag, New York, pp. 594–604.
- [18] G.N. Frederickson. "A data structure for dynamically maintaining rooted trees. Journal of Algorithms", 24(1):37–65, 1997. See also SODA '93.
- [19] Jacob Holm and Kristian de Lichtenberg "Top-trees and dynamic graph algorithms" Master's Thesis , Department of Computer Science, University of Copenhagen, August 14, 1998.
- [20] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. "Minimizing diameters of dynamic trees". 24th International Colloquium on Automata, Languages, and Programming (ICALP), pages 270–280, 1997.
- [21] Resource deadlocks in distributed systems," ACM SIGACTSIGOPS Symp. Principles of Distributed Computing, Ottawa, Ont., Canada, Aug. 1982. New York: ACM, 1983, pp. 157-164.
- [22] Ference Belik. "An Efficient Deadlock Avoidance Technique" 882 IEEE Transaction on Computers vol. 39, NO. 7, JULY 1990.
- [23] L. M. Ham, C. Mohan, "A Distributed Deadlock Detection Algorithm for a Resource-Based System," ZBM Research Report, RJ 3765, IBM San Jose Research Laboratory, 1983.
- [24] M. Singhal, "Deadlock Detection in Distributed Systems," Computer, Vol. 22, No. 11, pp. 3748, Nov. 1989.

- [25] I. Cidon, "An Efficient Distributed Knot Detection Algorithm," IEEE Transactions on Software Engineering, Vol. 15, No. 5, pp. 644-649, May 1989.
- [26] G.S. Ho and C.V. Ramamoorthy, "Protocols for Deadlock Detection in Distributed Database Systems," IEEE Trans. Software Engineering, Nov. 1982, pp. 554-557.
- [27] Qinqin Ni et al. "Deadlock detection Based on resource allocation graph" 2009 Fifth International Conference on Information Assurance and security.
- [28] JOHNSON, D.B. Finding all the elementary cycles of a directed graph. SIAM Comput. 4, 1 (March 1975), 77-84.
- [29] Deepak Ajwani and Tobias Friedrich. "Average-case analysis of online topological ordering". In Takeshi Tokuyama, editor, *ISAAC*, volume 4835 of Lecture Notes in Computer Science, pages 464–475. Springer, 2007.
- [30] Paul F. Dietz and Daniel D. Sleator. "Two algorithms for maintaining order in a list". ACM Symposium on the Theory of Computing, pages 365–372, May 1987.
- [31] David J. Pearce and R. Obermarck, "Distributed deadlock detection algorithm," ACM Trans. Database Syst., vol. 7, no. 2, pp. 187-208, June 1982.
- [32] K. M. Chandy, J. Misra, and L. M. Haas, "Distributed deadlock detection," ACM Trans. Comput. Syst., vol. 1, no. 2, pp. 144-156, May 1983.
- [33] K. M. Chandy and J. Misra, "A distributed algorithm for detecting Paul H. J. Kelly. "Online algorithms for topological order and strongly connected components". Technical report, Imperial College, London, 2003.



- [34] B. Haeupler, S. Sen, and R. E. Tarjan. “Incremental topological ordering and strong component Maintenance”, 2008.
- [35] H.-F. Liu and K.-M. Chao. “A tight analysis of the Katriel-Bodlaender algorithm for online topological ordering”. *Theor. Comput. Sci.*, 389(1-2):182–189, 2007.
- [36] M. Thorup. “Integer priority queues with decrease key in constant time and the single source Shortest paths problem”. *J. of Comput. Syst. Sci.*, 69(3):330–353, 2004.
- [37] PEARCE, D. J. AND KELLY, P. H. J. 2003. “Online algorithms for topological order and strongly connected components”.
- [38] 12. T. Kavitha and R. Mathew. “Faster algorithms for online topological ordering, 2007”.
- [39] PEARCE, D. J. AND KELLY, P. H. J. 2007. “A dynamic batch algorithm for maintaining a topological order”. Tech. rep., Victoria University, Wellington, New Zealand.
- [40] Navneet Kaur and Deepak Garg. “Analysis of the Depth First Search Algorithms CiiT”. *International Journal of Data Mining Knowledge Engineering Print*: ISSN 0974 – 9683 & Online: ISSN 0974 – 9578 DOI: DMKE012012007 January,2012

## LIST OF PUBLICATION

---

- [1] Megha Tyagi & Deepak Garg “Comparative Analysis of Dynamic Graph Techniques and Data Structure”. International Journal of Computer Application (0975-8887), Vol. 45-No.5, May 2012, Page Number 41-46. [Published].
- [2] Megha Tyagi & Deepak Garg “An Improved Method for Deadlock Avoidance using Incremental Cycle Detection”. [To be communicated].