International Journal of Computer Applications (0975 – 8887)
Volume 81 – No.4, November 2013

Enhanced JGraphEd Drawing Framework for Graph Drawing Application

Jitendra Sharma
M.Tech Scholar, Dept. of Computer Science
Swami Keshvanand Institute of Technology
Jaipur, India

Shubhra Saxena
Reader, Dept. of Computer Science
Swami Keshvanand Institute of Technology
Jaipur, India

ABSTRACT
The graph drawing and analyzing software JGraphEd is easily available and provides better environment, so that users are capable to study and review the algorithm, resolve a hard headed practical problem and study the functional process via graphical display environment. Although, there are few software's which are capable of solving these problems as the availability and compatibility with various environments has been quite a hard task. The newly proposed algorithms such as checking of 'Cycles' in a graph, checking of 'Bi-Partite' of a graph, and 'Isomorphism Test' provides much better recognition to the user who want to learn the algorithms efficiently and easily. This can be accessed via any Internet browser anytime, anywhere, without downloading and setting up any software.

Keywords
Graph, Algorithm, Environments, JGraphEd, Software.

1. INTRODUCTION
Graph theory is quickly forwarding into the mainstream of mathematics primarily because of its various applications in various fields which include algorithms, computations, operations research and scheduling [1][2]. There are different ways of storing graphs in a computer system. It depends on the data structure used and also depends on two things, i.e. the algorithm and the graph structure which is applied for manipulating the graph. It was designed to allow user to draw a graph step by step by adding, removing and modifying nodes and edges. It has a variety of independent algorithms provided for manipulating and visualizing graphs. This paper also suggest what more can be added to JGraphEd to make it better. Section 2 describes the problem statement and solution approach. Section 3 describes various existing technologies for the development of java graph algorithms. Section 4 describes the brief overview of different algorithms in JGraphEd. Section 5 gives fairly extensive description of the proposed algorithms that are implemented in JGraphEd. Section 6, describes the java implementation of the proposed algorithms of JGraphEd.

2. PROBLEM STATEMENT
In the present era, a wide range of graph editing and analyzing tools are available. The most important problem is to find the best tool which can solve the problems faced by the user, and also which is compatible with the runtime environment and can easily be modified as per need. The main problem that arises in the implementation of the algorithm is to use the algorithms by their tools. Some tools are implemented with very few algorithms. As per the different reviews obtained from researchers, a new algorithm is required that can simplify the problems. JGraphEd provides the best editing and analyzing software but can still be made better and simpler for easy hand application. A variety of algorithms are implemented in JGraphEd and the code structure is very neat and clean making one easy to extend. Also, the documentation and graph data structures used in JGraphEd are easy to understand.

3. DIFFERENT GRAPH ANALYZING TOOLS
In present scenario, a wide range of graph editing and analyzing tools are available. The problem is to find the best which solves the problem of the user, which is compatible with the runtime environment and can easily be modified as per user needs. The below Fig 1 describes various existing technologies for the development of java graph algorithms.

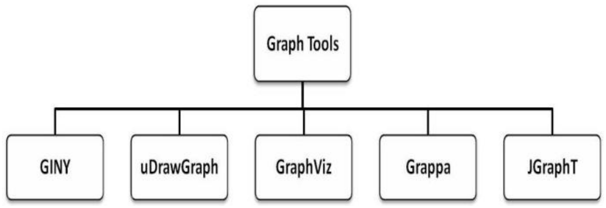


Fig 1: Different graph analyzing tools

3.1 GINY
GINY is an open source Java graphing library. The GINY application is implemented with the help of Piccolo. It is used for the creation of 2D structure graphic programs [3]. The main purpose behind using GINY is that it is simple and easy to use and it doesn't provide any kind of algorithms directly. The GINY is simply an interface layer that is useful for building graphing projects, but there are some more common algorithms that are available. The different algorithms include Embedded Layout, Shortest Path, Hierarchical Layout and many more.

3.2 uDrawGraph
The primary objective of uDrawGraph is that user can create their graph structure as easily as possible in order to get a clear layout. Everything is just a few more mouse clicks away which includes inserting a node, double clicking to change the attributes of the nodes like text area, shape, colour and linking can be done by simply dragging a line from one node to another. It creates various diagrams, structures, hierarchies, visualizations, flow charts using automatic layout which is much easier and faster than any other application drawing program.

The outstanding feature of uDrawGraph is the automatic graph layout, as present in the other graph drawing tools that users have to rearrange the nodes every time whenever they adds something to graph. But this doesn't happen with uDrawGraph, because the entire layout can be done by software [4]. One of the most amazing properties of uDrawGraph is the incremental graph layout. This feature is used when a large amount of nodes and edges have to be added to a given graph [5]. It also describes the graph structure, API, graph layout, its visualisation, multi graph, multi view and also drag and drop feature.

3.3 GraphViz

GraphViz is the most common graph drawing tool, and an open source graph visualization software. Graph visualization is means of presenting the required information as diagrams and also represents as networks. It takes programs details of the graph in an easy text language and creates diagrams in various utile formats. It is only helpful in drawing a graph because a user can't apply the basic graph algorithms for seeing its properties. In GraphViz the graphs that we will use can be either directed or undirected. It extends with both graphical as well as command line tools [5][6].

3.4 Grappa

Grappa is an extensive graph drawing package written in Java language. It consists of different classes that implement the graph representation with the help of API. The main disadvantage with Grappa is that it was built in Java language with JDK version 1.2 or its previous one, but now a day's users are mostly using Java Development Kit (JDK) version 1.5 or the later version. With the development of web based technology the Grappa provides different classes and packages for different remote services. It provides client and server based application program in which one machine acting as client can request "n" number of graph drawing services from server side on anywhere on the network [7]. Grappa was built in Java because its applications can be created using applets and can be executed with any Java enabled Web browser. Further, Grappa provides three useful features Extensibility, Portability, and Customizability.

3.5 JGraphT

JGraphT is used to allow graph theory and algorithms and is designed in such a way that it is easy to understand and safe. JGraphT Java graph library is free and provides usage in wide range of applications. Users can create graphs based on URL, XML, and other extensions. JGraphT supports various types of graphs including weighted graphs, unweighted graphs, or any other kind of user defined graphs. It is also used for finding whether the graph is directed or undirected, simple graphs and graphs having multiplicity properties.

3.6 JGraphEd

JGraphEd is a Java graph redaction software and graph drawing model. It is contrived for users to create graphs stepwise by adding, removing or modifying nodes or edges. There are many reasons for the question that why JGraphEd was chosen. Some of the reasons are listed below:

- Most important reason for choosing JGraphEd is that it can be executed online which makes the software platform independent.
- It is concerned with Java and is compatible with JDK 1.5 or later version of it.
- A variety of algorithms are implemented in JGraphEd.
- The code structure is very neat and clean which makes it extensible.
- Last but not the least**,** the documentation and graph data structures used in JGraphEd is easy to understand.

JGraphEd operates for simple graphs. It has various features which were found more prominent than other graph drawing tools which include modifying graphs in any way. Graphs can be rotated, resized, even one node can be selected and shifted to some other place, nodes can be labeled, edges can be selected and edges can be curved for all the algorithms to implement on them.

4. EXISTING ALGORITHM IN JGRAPHED

There are many algorithms implemented on JGraphEd which makes it one of the most useful graph algorithm tools. The algorithms which are implemented on JGraphEd are creating a random graph, depth first search on a graph, checking connectivity of a graph, checking the Biconnectivity of a graph, making a graph Maximal Planar, checking the planarity of the graph, performing embedding of a planar graph, canonical ordering of a graph, normal labelling of a graph, straight line grid embedding of a graph, making tree of a graph, calculating Dijkstra's shortest path between a pair of vertices of a graph, displaying minimum spanning tree of a graph. There are three types of applications of JGraphEd. [8]

- Test Application - It tests various properties of a graph and gives a Boolean result.
- Operation Application - This applies some algorithm to a Graph.
- Display Application - This displays the graph after applying some algorithm and giving the desired result.

5. PROPOSED ALGORITHM IN JGRAPHED

This section describes about the proposed applications that is added to JGraphEd, The first three sections describe about the additional application that has been made and make JGraphEd better.

5.1 Checking for Cycles in a graph

There is a difference between property of a graph having cycles and a graph having no cycles. It is always important to know whether the graph contains a cycle or not, although it is obvious for graphs having less size, the problem may occur for huge size graphs. The algorithm for checking whether a graph contains a cycle or not is followed by [9]:

```
Input: A Graph G
Output: Whether Graph G is having cycle or not.
Method:
For every connected subgraph g of the graph G
  {
   Apply Depth first search on g
     If ∃ an edge e ∈ E where E = set of edges of g such that
     e is a backedge, the graph G contains one or more cycle
   Else
     G doesn't have any cycle.
  }
```

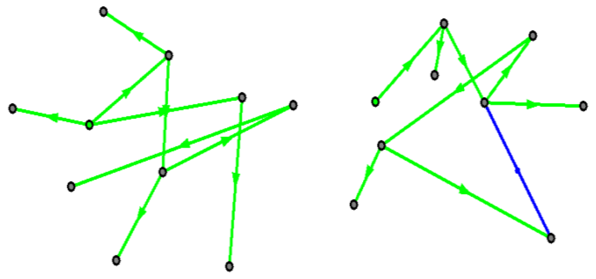


Fig 2: Cyclic graph showing the back edge in dfs search

The CycleCheckOperation.java file in operation package checks for every connected graph of a graph G, and applies depth search first and eventually checks for any back edges giving desired result. Fig 2 shows two graphs, the first one doesn't contain any cycle where as the second one does. So there exists one back edge in second graph.

5.2 Checking of Bi-partiteness of graph

A graph is said to be bipartite graph only when it does not contain any odd-length cycles in a graph. Because if effort is made to put one node of the cycle of odd length in one set and then on adding the next node in the other set, the final node will appear in the same set, which is not allowed as per the definition of Bi-partiteness, which says there can't be any node between the elements of same set. Fig 3 shows two graphs, one is a cycle of odd length and the other one is cycle of even length.

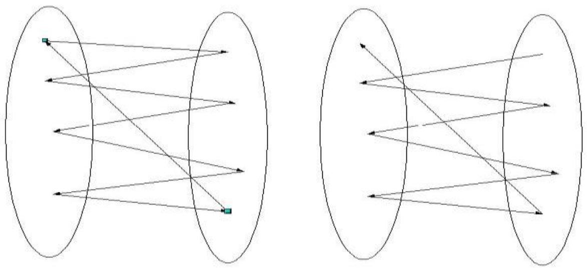


Fig 3: Graphs of odd and even length subjected to Bipartite Test

The first figure tries to partition a circle of length eight into two subsets without keeping any edge in one set which can be done. But the second figure can't be completed because the starting point and ending point lies on the same set. The algorithm for finding whether a graph is bi-partite or not is exactly as finding whether a graph contains a graph of odd length or not. Algorithm for predicting whether the graph is bipartite or not is followed by [10]:

```
Input: A Graph G
Output: Whether Graph G is bipartite or not.
Method:
Apply depth first search operation on Graph G
  {
    For each backedge, there is a circle, calculate the length of
    the circle.
       {
         If length is odd then the graph is not bipartite,
         Else
         The Graph is bipartite.
       }
  }
```

5.3 Checking for Isomorphism of two graphs

It is more difficult to identify the isomorphism of two graphs because there are n! different ways to find out one-to-one correspondence relation between the vertex sets of the given two graphs with ‘n’ vertices. The checking of one-to-one correspondence relation between the graphs is more difficult when the value of ‘n’ is large. The proof that the two simple graphs are not isomorphic is by giving the detail description about the common property that is not shared by both the simple two graphs, if they do not share any property between them, the graph is not isomorphic. As seen in Fig 4, the above two graphs are having the same number of vertices and edges, but they are not isomorphic because the first graph has a vertex ‘e’ of degree one and the second one doesn't. If the total number of vertices, total number of edges and the degree of the vertices all have same characteristics then the graph is isomorphic, However, when these measures are same, it doesn't necessarily mean that the given two graphs are isomorphic. There are no useful sets of invariants currently known which can be used to find whether simple graphs are isomorphic.

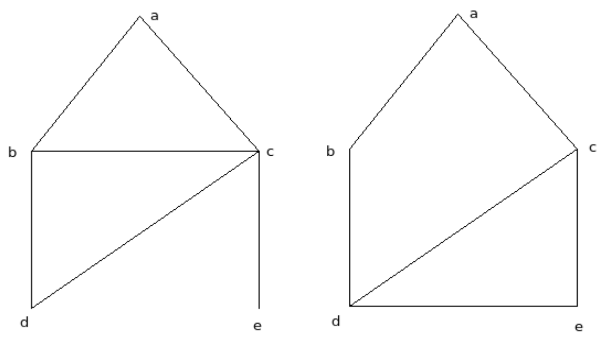


Fig 4: Simple non-isomorphic graphs satisfying first two properties of a graph

Input: Two graphs G_1 and G_2
Output: Whether G_1 and G_2 are isomers or not.
Method: The heuristics approach to check whether two graphs are isomorphic or not are:

1. The number of nodes of the graphs must be same.
2. Then number of edges of the graphs must be same.
3. The degrees of the nodes must be same, i.e. if we sort the degrees of Graph G_1 in a vector and compare it with sorted sequence of the degrees on Graph G_2, they must be equal for the graph to be isomorphic.

But as it is known that predicting the isomorphism of two graphs are NP-hard, therefore there can be some counter example for beating a heuristic and improving it. For example consider the following Fig 5.

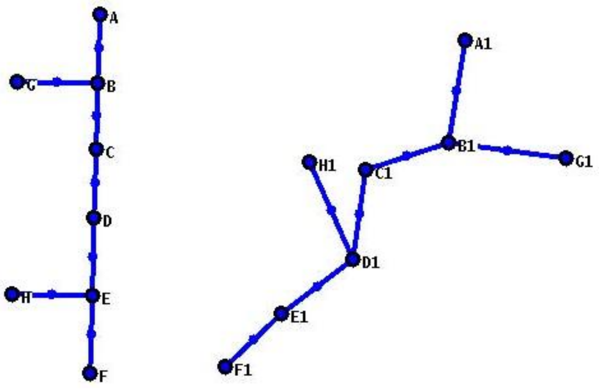


Fig 5: Two non-isomorphic graphs that satisfy the above heuristic

1. Both the graphs have eight nodes.
2. Both have seven edges.
3. The sorted sequence of degrees of first graph is {1, 1, 1, 1, 2, 2, 3, 3}
4. The sorted sequence of degrees of second graph is {1, 1, 1, 1, 2, 2, 3, 3}

But the two graphs in above Fig.5 are not isomorphic, because in first graph there exist a edge CD where `C' and `D' are nodes of degree two, where as in second graph there is no edge between the nodes having degree two , which are `C1' and `E1'.

6. JAVA IMPLEMENTATION OF PROPOSED ALGORITHM

The proposed JGraphEd tool interface is same as existing one but there are minor changes. It is different in the following ways such as:

- User Interface of JGraphEd (Proposed Menu Icons)
- Test Menu (Proposed Test Operations)

It consists of many proposed algorithms, which includes the following testing algorithm:

- Checking whether graph contain cycle
- Checking of Bi-Partiteness of a graph
- Checking for Isomorphism of two graphs

6.1 Implementation of Cyclic test

Step 1: Input graph for checking whether the graph contains cycle or not: The complex input graph has shown in Fig 6 in the new graph editor window, the input graph consist of 6 nodes and 10 edges. Now, cyclic test algorithm is applied on the given input graph.

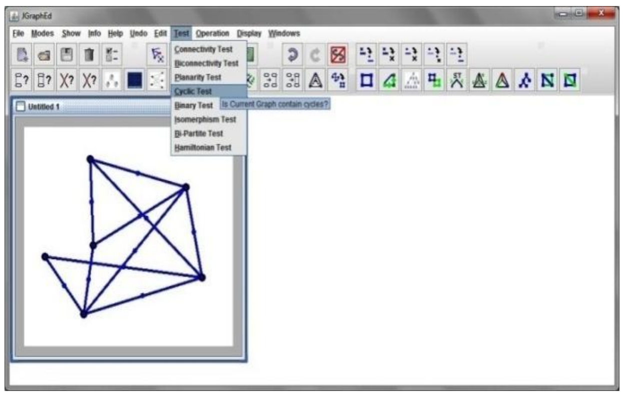
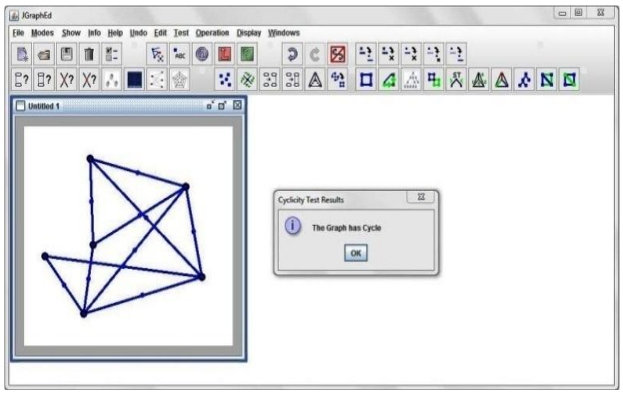


Fig 6: Input graph for checking cycle

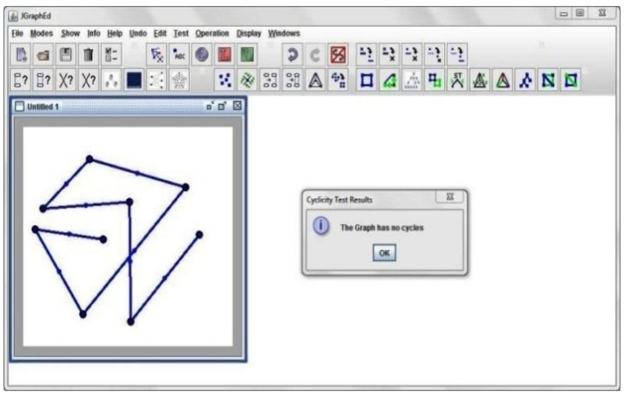
Step 2: Cyclic output is displayed showing whether the graph has cycles or not: The user clicks onto the cyclic test option, it will show two types of output whether the current graph contains cycle or not. In this case, the graph which has been selected so far shows an output message window as "The Graph has Cycle" which means there exists one cycle or more than one cycle which depends upon the input graph.


JGraphEd
File Modes Show Info Help Undo Edit Test Operation Display Windows
Untitled 1
Cyclicity Test Results
OK

The Graph has Cycle

Fig 7: Result showing the graph contains cycle

Step 3: Checking of cyclic test on another graph, output graph contains no cycles: Now, in the second case of cyclic test algorithm, another graph is drawn in a new graph editor window i.e., new graph with 8 edges and 9 nodes. When cyclic test algorithm is applied it will show an output message window as "The Graph has no Cycles" this is because the selected graph contains no cycles, the starting and ending of graph has different node so it doesn't create any cycle. For a graph to be cyclic the edges must be equal or greater than nodes, but in our example value of edges is less than nodes.


JGraphEd
File Modes Show Info Help Undo Edit Test Operation Display Windows
Untitled 1
Cyclicity Test Results
OK

The Graph has no cycles

Fig 8: Result showing the graph contains no cycle

6.2 Implementation of Bi-partite test

Step 1: Input graph for checking whether graph is Bi-Partite or not: The complex input graph has been shown in Fig 9 in the new graph editor window, the input graph consist of 6 nodes and 6 edges. Now Bi-Partite Test algorithm is applied on the given input graph.

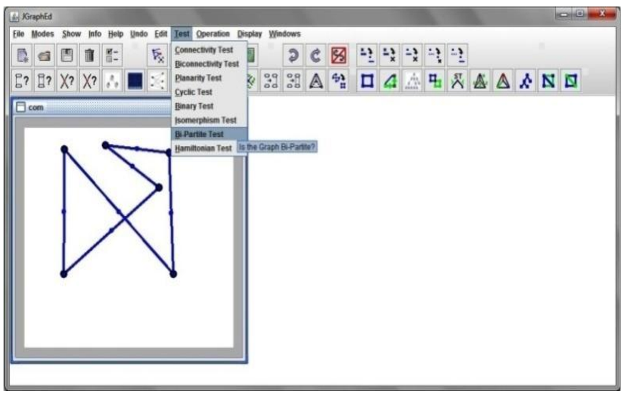
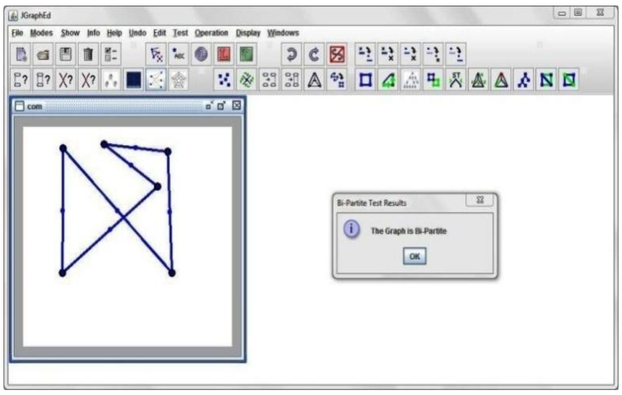


Fig 9: Applying Bi-Partite test on input graph

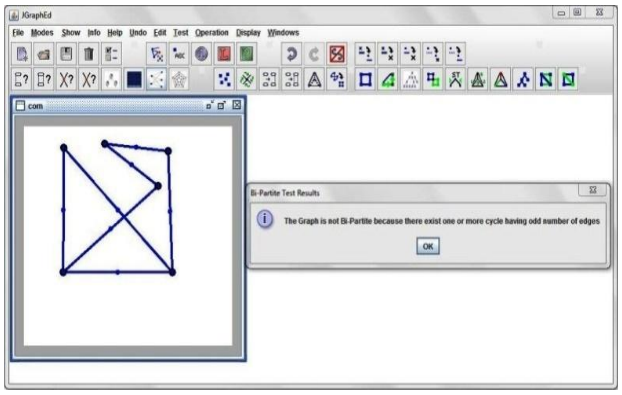
Step 2: Bi-Partite output is displayed showing whether the graph is Bi-Partite or not: The user clicks onto the Bi-Partite test, it will show two types of output whether the current graph is Bi-Partite or not. In this case the graph which has been selected so far shows an output message window as "The Graph is Bi-Partite" which means there exists cycles with even number of edges.


JGraphEd
File Modes Show Info Help Undo Edit Test Operation Display Windows
com
Bi-Partite Test Results
OK

The Graph is Bi-Partite

Fig 10: Result showing the graph is Bi-Partite

Step 3: On adding another edge in a graph, it displays that the graph is not Bi-Partite: Now, in the second case of Bi-Partite test algorithm, another edge is added in a current graph i.e., the current input graph has been modified by adding the edge between the two bottoms most node in a graph. When Bi-Partite test algorithm is applied it will show an output message window as "The Graph is not Bi-Partite" because there exist one or more cycle having odd number of edges. This is because the selected graph contains odd number of edges cycles. In this graph it contain two cycles having odd number of edges, the first cycle contains 3 edges and other cycle contains 5 edges respectively.


JGraphEd
File Modes Show Info Help Undo Edit Test Operation Display Windows
com

Bi-Partite Test Results

The Graph is not Bi-Partite because there exist one or more cycle having odd number of edges

OK

Fig 11: Result showing the graph is not Bi-Partite

6.3 Implementation of isomorphism test

Step 1: Input graph for checking whether the graph is isomers graph or not: The complex two input graph has been shown in Fig 12 in the new graph editor window, the first input graph consist of 4 nodes and 4 edges and second input graph consist of 4 nodes and 5 edges. Now, isomorphism test algorithm is applied on the given input graphs.

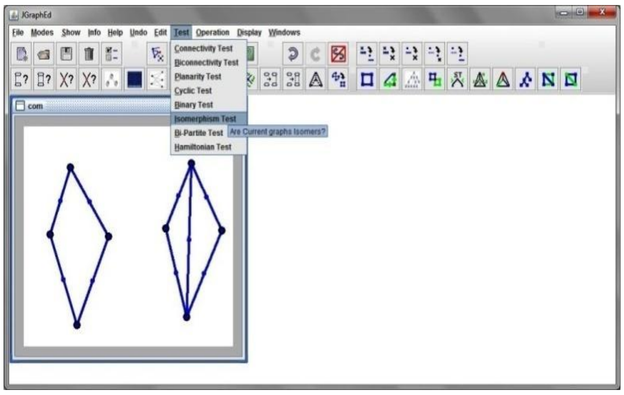
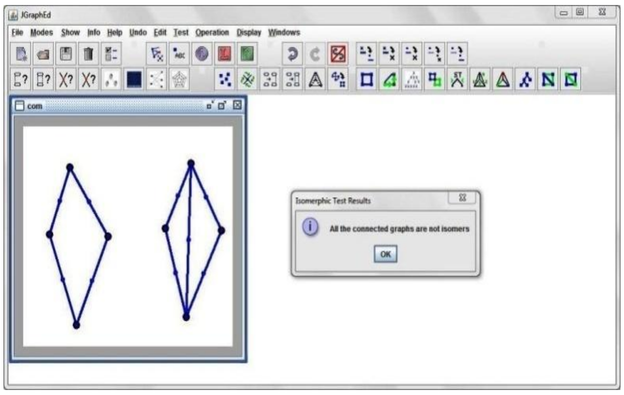


Fig 12: Applying isomorphism test on input graph

Step 2: Isomorphism test output is displayed showing whether the connected graphs are all isomers or not: The user clicks onto the isomorphism test option it will show two types of output, whether the current connected graphs is isomers or not. In this case, the graphs which has been selected so far show an output message window as "All the connected graphs are not isomers" which means there exists an extra edge in the graph which is not present in other, so both the graphs are not equal and not isomers to each other.


JGraphEd
File Modes Show Info Help Undo Edit Test Operation Display Windows
com

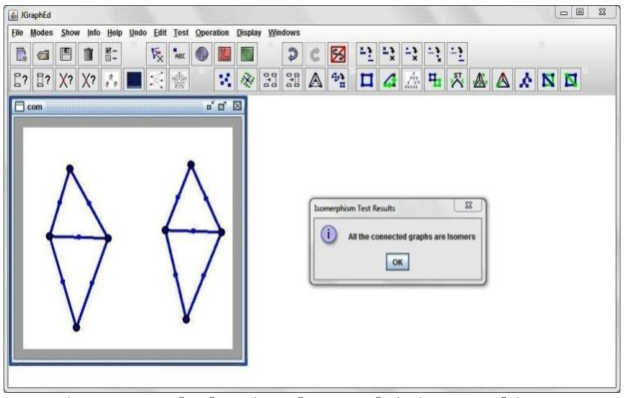
Isomorphic Test Results

All the connected graphs are not isomers

OK

Fig 13: Result showing the graph is not isomorphism

Step 3: On modified the current connected graphs, it displays graphs are isomers: Now, in the second case of isomorphism test algorithm, another edge is added in a current graph i.e., the current input graph has been modified by adding the edge inside the graph. When Isomorphism test algorithm is applied it will show an output message window as "All the connected graphs are Isomers" this is because the selected graphs may contain each and every edge common to both the graphs that's why both the connected graphs are isomers.


JGraphEd
File Modes Show Info Help Undo Edit Test Operation Display Windows
com

Isomorphism Test Results

All the connected graphs are Isomers

OK

Fig 14: Result showing the graph is isomorphism

7. CONCLUSION

The user interface of JGraphEd has been enhanced by adding proposed icons on the JGraphEd toolbar. All proposed icons on the toolbar are properly working without any problem. In proposed test menu, it successfully added the proposed algorithm to the drop down list of test menu. They are 'Cyclic Test', 'Bi-Partite Test', and 'Isomorphism Test' and have shown successfully while clicking on the test menu of JGraphEd menu bar. This paper has depicted the entire designed views and characteristics of JGraphEd. It also describes the structure and framework for its redaction and drawing potentialities, the execution of algorithms or operations, the different data structures which are furnished with JGraphEd. JGraphEd also assist its use in different graphs and structure and also provides detailing of various algorithms.

8. FUTURE SCOPE

In the future work some more test and analysis algorithms can be added, for example making a 'non-planar graph' planar by deleting some selected edges, implementation of 'Minimum spanning tree' using krushal's algorithm, converting a 'Graph' into orthogonal, calculation of 'Breadth first search' of a graph could be included in present JGraphEd to provide better understanding of graph algorithm.

9. REFERENCES

[1] Sokhom Pheng, Clark Verbrugge, “Dynamic Data Structure Analysis for Java Programs”, 14th IEEE International Conference on Program Comprehension, pp. 191-201, 2006

[2] Jeremy Singer, Chris Kirkham, “Dynamic Analysis of Java Program Concepts for Visualization and Profiling”, Journal of Science of Computer Programming, vol.-70, no.-2, pp. 111-126, 2008

[3] Benjamin B. Bederson, Jesse Grosjean, and Jon Meyer, “Toolkit Design for Interactive Structured Graphics”, IEEE Transactions on Software Engineering, vol.-30, no.-8, August 2004.

[4] Sergei Gorlatch, Marco Danelutto, “Integrated Research in GRID Computing”, ISBN 13:978-0-387-47656-3, Springer Science Media Ltd., 2007

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, “Pattern-Oriented Software Architecture - A System of Patterns”, ISBN 978-81-265-1611-7, John Wiley India Ltd., 2008.

[6] Ivan Herman, Guy Melançon, M. Scott Marshall, “Graph Visualization and Navigation in Information Visualization: A Survey”, IEEE Transactions on Visualization and Computer Graphics, vol.-6, no.-1, pp. 24-43, 2000

[7] Naser S. Barghouti, John M. Mocenigo, Wenke Lee, “Grappa: A Graph Package in Java”, IEEE Transactions on Graph theory and AT & T Laboratories, vol.-13, pp. 336-343, 2007

[8] Jon Harris: JGraphEd – “A Java Graph Editor and Graph Drawing Framework”, Carleton University , Comp 5901 Directed Studies, April 2004.

[9] Hongbo Liu, Jiaxin Wang, “A new way to enumerate cycles in graph”, IEEE Computer Society, Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services.2006.

[10] Luo Shiguang “Application of Bi-partite Optimal Matching in Color-Base Image Retrieval”, IEEE Computer Society, International Conference on Machine Vision and Human-machine Interface, 2010.