The Florida State University
College of Arts and Science

Building An Algorithm for knowledge representation using LDAGs.

Saeed Tarokh M.D.

June, 2002

A Master's project submitted to the
Department of Computer Science
In partial fulfillment of the requirements for the
Degree of Master of Science

Major Professor: Dr. Daniel Schwartz

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Daniel G. Schwartz for his guidance and help. In addition, I would like to thank Dr. Hilbert Levitz and Dr. Sara Stoecklin for their willingness to serve on my committee. I have benefited very much from these professors' course offerings and advice.

To: My Dear Mother and The Great Memory of my Father

ABSTRACT

In this project, we are trying to build an algorithm for knowledge representation through the use of layered directed acyclic graphs (LDAGs). These LDAGs have been built using multiple-parent, multiple child capability. In another word, each node in such a graph has the capability of having multiple children as well as multiple parents. To show the effectiveness of our algorithm, we have used two different programs. Both programs have been written using java programming language and its awt package. The first program benefits the existence of multiple tables in the database. Such a structure of the database gives the program the ability to logically break the database to many parts and use each part to build a small graph. On the other hand, the second version of the program benefits the existence of only one table as its input and could build a large graph to represent the existing data in the database. A combination of such these two programs could let the system manager the ability to build different graphs using different logical partitions of the database. To access the information, which are needed to build the layered directed acyclic graphs; we have used an online database and the basic SQL commands. Since all interactions with the database employ standard SQL, the database system employed on the server side can be any SQL compliant database for which there exists a JDBC driver.

We should mention that this program would serve as the first draft of such a system. Having built the necessary algorithm for this program, we are certain that we can use the same algorithm with some minor changes for future works.

The current project is to read the data from an online database and to build the multiple- children multiple-parent layered directed acyclic graphs and eventually update the database. The program benefits from a server. The server side creates and maintains a MySQL relational database. All information is stored in this database. The output program is implemented as a Java frame that runs under any standard java environment. The server program is based on the Sun Microsystems Java Development Kit (JDK) 1.1.6 and uses Java Database Connectivity (JDBC) to perform SQL statement execution on the MySQL database. The output program is based on JDK 1.2.2, which includes the Java Swing package.

CHAPTER ONE: INTRODUCTION

1.1 Overview

The *layered directed acyclic graph* aims to act as an interface for the content of the remote databases. Therefore, our objective is to create a graphical layered directed acyclic graph, which represents the relationships among the current data in the remote database. This is used to show the classification of the data to the user through graphical representation. Since all interactions with the database employ standard SQL, the database system employed on the server side can be any SQL compliant database for which there exists a JDBC driver.

Dr. Daniel Schwartz and Saeed Tarokh, by using multiple tables in the database, created a first draft of such a system as a stand-alone application. In that draft, and in order to build a parent-child relationship, we represented each node of the graph by a separate table. With building a one-table database and adding the abilities to analyze the starting table and to update the database as well as adding the dummy nodes, we built the second (current) version of the system.

The present system extends our initial work further by adding the ability to create dummy nodes and adding these dummy nodes to the database. This can help us to get rid off some of the line overlapping that might happen when we build the graph. In addition, to avoid having multiple tables in the database, we have created a one-table database, which represents the parent- child relationships. The database system employed here is the well-known freeware MySQL.

The project started with defining what we had done to build the first version, and later we describe the second version of the software. The biggest challenge in such a system was to build an algorithm, which determines the x-derivate of each node on the graph. After addressing the second version of the software, we address an algorithm, which we have built to determine the x-derivates of the nodes in the graphs.

Later on, we will describe the opportunities for possible future developments and what we will need to do to make more steps towards the overall goal of a fully functional system.

CHAPTER TWO: THE DESIGN OF THE FIRST VERSION

2.1 The System Structure

The system has been designed using a MySQL database, a Java Database Connection (JDBC) program, and an output-producing program.

2.2 The Programming Language

Java was chosen as the programming language for this project, not only because it is popular, easy to use, easy to understand, but also because of its portability and its convenience for GUI design. Java's graphical capability and compatibility with the major web browsers make it the first choice to do the web-based application programming.

The output program is written with JDK 1.4. This is the latest version in our department system. It includes the Java Swing library package for doing graphical programming.

Since the database MySQL does not yet have a driver that works with JDK 1.4, we have used JDK 1.1.6 for the Java Database Connection (JDBC), which acts as the interface of server program, and the database, Java is an excellent language for server-side database applications.

2.3 The Database

The database is the central part, which is used to store and retrieve information. Currently, the database is located on the following server: dbsrv.cs.fsu.edu. However, we can use any kind of MySQL database. MySQL, a freeware SQL (Structured Query Language) database is used in this project. It is a very fast, truly multi-threaded, multi-user, and robust SQL database. But it is not as functional as some current commercial database products, such as, Oracle, MS SQL Server. Since it's free, it's good to be used in the school as educational purposes. But our database system could be easily changed to be one of the commercial products as mentioned above. The future project implementations could try to switch to Oracle database, as it is available now in our department system.

2.4 Connection between server and database

When the server is started, it establishes a connection with the MySQL database through Java Database Connectivity (JDBC) as described in Figure 2.1.

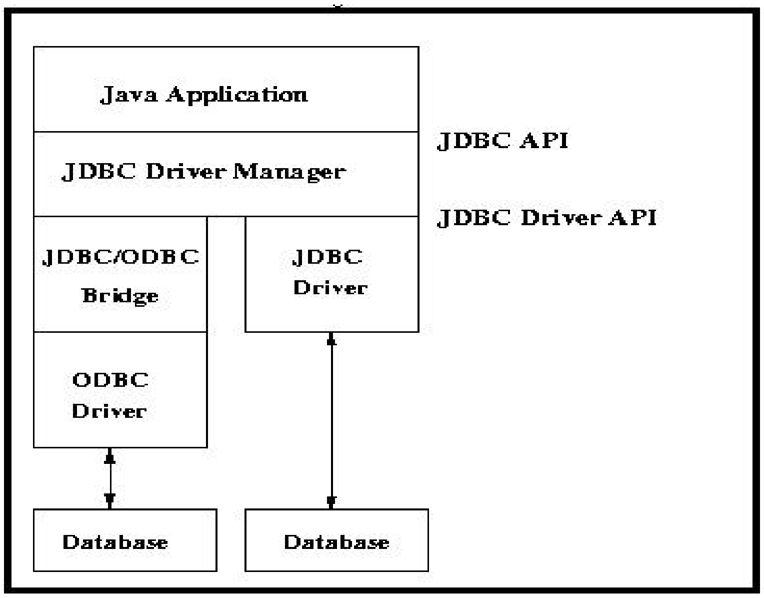


Figure 2.1 Java Database Connectivity

JDBC is Java API that allows Java application program to access relational databases using SQL. Each database system is accessed via a specific JDBC driver that implements the java.sql.Driver interface. For MySQL, we use the following statement to load and link the MySQL database driver,

```
Class.forName("org.gjt.mm.mysql.Driver" );
```

After loading the database driver, the JDBC DriverManager needs to open a connection to the database, where the database is specified by a specially formatted URL. The connection statements are as follows,

```
String dbUrl = "jdbc:mysql://dbsrv/tarokhdb";
Connection connection =
        DriverManager.getConnection(dbUrl,
        user, password);
```

After finishing the connection, we can directly implement SQL statement inside our applications, for instance,

```
Statement statement =
          connection.createStatement();


                  (“SELECT * FROM Links “);

while(resultSet.next())
{
   Association A2= new Association();

   A2.setName(resultSet.getString("Child")) ;
  store.addElement(A2);
   A2.parentName.addElement(resultSet.getString
                                         ("Parent"));
}// end While Loop
```

CHAPTER THREE: PROJECT IMPLEMENTATION

3.1 The Database

In the first version, the database contains the tables created by the user. These tables represent those nodes in the graph that have children. For the child-less nodes, regardless of what level they have in the graph, there is no need for a separate table. However, the user can create an empty table for the child-less nodes. Since the program tries to read a table for each node in the graph from the database, there is a need to catch the exception, when the table is not available. To do this, we use the following piece of code:

```
 catch ( SQLException sqlException )
{
                //System.out.println("Where is this table?");
}
```

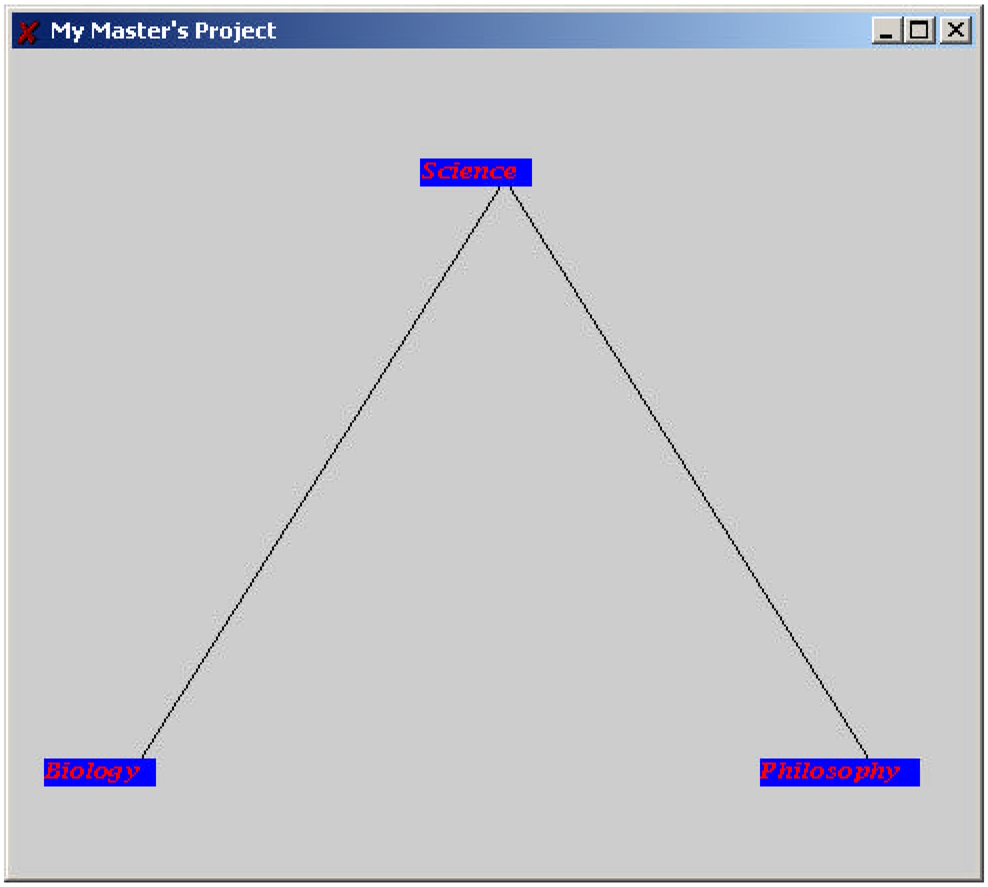
The various tables created for this purpose are:

Table Name	***Description***
Root	The only node that appears in the first level of the graph. The existence of this node and the corresponding table is a must for the program. There is a possibility to change the name of the root table. To do this, we have the following implementation. `String root_name;` This is the variable that we can assign the name of the root to, for example: `root_name ="Science";`

In this section, we provide an example for the root table and show the output of the program. Here is the example of a root table, which has been named as Science:

Table Science
Biology
Philosophy

And this an output which shows the relationship between the Science and its children:



In the second version of the program, we have made some changes to the structure of the database.

The various tables created for this purpose are:

Table Name	*Description*
Links	In this table, the user stores the relationships between the parent and children. Each record in this table has two attributes, parent and child.
Nodes	This table is invisible from the user. This is to store the name of all the nodes in the database. To do this, the program updates the content of the Nodes based on the content of the Links table. There is only one attribute for each record in this table, which is the name of the node.
NodeLevel	This table is used to store the name of the nodes and the level, they will appear on the graph in. In this table, we have two attributes

	for each record, which are the level and the name.
NodexLevel	This table is used to store the name, the x-coordinate and the level of each node. In this table, we have three attributes for each record, which are the level and the name, and the x-coordinate.

The detailed design of each of these tables can be found in the appendix.

3.2 The Client Program

3.2.1 First Version: In the first version of the program, we have used seven different classes. In this section, we are going to describe these classes shortly and explain their behaviors.

3.2.1.1 Association.java: This class is a very important part of our program. Since, we have the possibility that a node has multiple parents and/or multiple children, we need to somehow build an association among a node and its children and parents. To do so, we have implemented the class Association. To store the name of the parents of each node, we have added a vector to the class Association. Similarly, to store the name of the children of each node, we have added another vector to the class Association. Other important elements of this class are:

```
private int Level;
private double x;
private double y;
private String name;
```

Obviously, we have used an integer to store the level that each node will appear in. We have used similar implementations to keep the Y-Coordinates as well as X-Coordinates of the nodes. The reason for using the double instead of integer for X and Y coordinates is to have flexibility at the time that the program wants to determine these values.

3.2.1.2 Print_Data.java: This class is very important, in the case that we need to make any change to the implementation of the program. This class has been implemented to show the actual values that we have stored in each object of the class Association. Since each object of the class Association represents a node in this program, using the class Print_Data.java, we can actually see where each node will be put. This class is for the debugging purposes. Once we finish all the changes that we intend to make, we will not need to have this class to create the GUI output. Here is a short sample of the output created by the class Print_Data.java created for the simple program in section 3.1:

```
My Name is: Science
My Level is: 0
My Height is: 87
My width is: 250.0
Here are my children:
---------Biology

---------Philosophy

Here are my Parents:
My Name is: Biology
My Level is: 1
My Height is: 387
My width is: 62.0
Here are my children:
Here are my Parents:
---------Science

My Name is: Philosophy
My Level is: 1
My Height is: 387
My width is: 438.0
Here are my children:
Here are my Parents:
---------Science
```

3.2.1.3 Read_Data.java: This class is to give us the ability to read all the necessary data from the database, and store that in the program. To do this, we need to read multiple tables from the database and store the data in a vector that acts as our primary run-time storage. This vector has been declared in the class MyProject.java. To provide the ability to store the data, class Read_Data.java reads the content of each table and creates an instance of class Association.java and stores that in the vector.

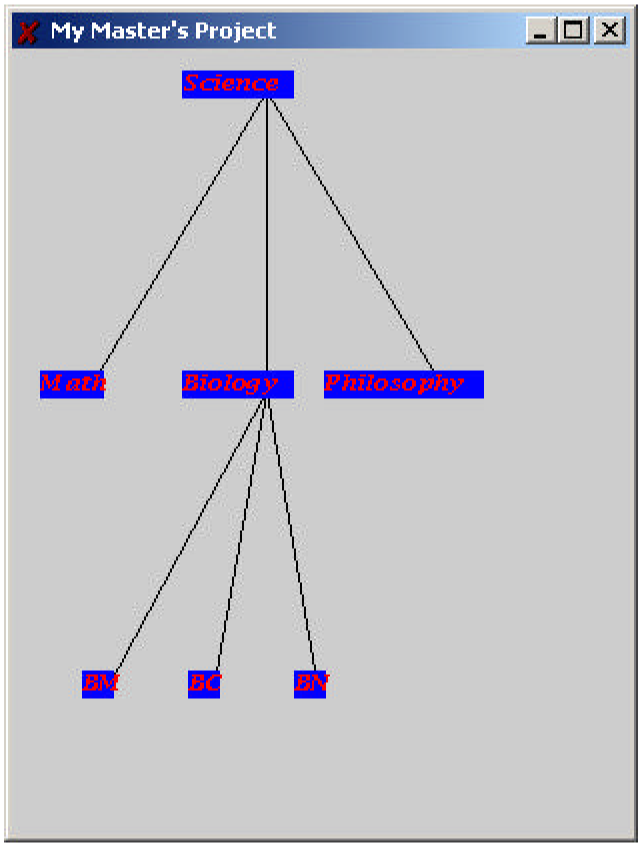
3.2.1.4 XCoordinate.java: This class is the location of the algorithm that we have created and implemented. Upon a call to this class, the program will determine the value of the x-coordinates for each node and store those in the corresponding fields of each node (instances of the class Association.java).

3.2.1.5 Dummy_Data.java: To provide a better output for the program, we need to consider that some nodes in the program might not have any children. Because we don't want the children of the other nodes to appear in the location of the children of these children-free nodes, we will need to create some dummy children to keep the shape of the output acceptable for the casual user. To do so, we have implemented a class to create this kind of dummy data. This class gives us the ability to recognize these children-free nodes, create the dummy children for them, and avoid the appearance of these dummy nodes on the final output.

The following is an Example of the program output and its corresponding database content when some of the nodes don't have any children.

Table Science
Biology
Philosophy
Math

Table Biology
BM
BC
BN



3.2.1.6 MyProject.java: This class is to work as the heart of the implementation of our program. In this class, not only we create objects of the above-mentioned classes, but also we give other abilities to the program to make it more and more powerful. In this class, we have created the GUI, which is the output of the program. To do so, we have used the capabilities of the class JFrame.java. Also we are using the package java.sql, to establish the connection to the remote database.

3.2.1.7 Main.java: This class is to create an instance of the class MyProject.java, and to launch the application.

3.2.2 Second Version: In the second version of the program, we have used sixteen different classes. Although the implementation of the classes Main.java, Association.java, XCoordinate.java, and Print_Data.java have been preserved, but other classes have gone through some modifications. Also, we have added more classes to provide new abilities. In the following section, we will review the new classes as well as the modifications that we have made to the old classes, which we have inherited from the first version.

3.2.2.1 Levels_Determination.java: In this version of the program, since we don't have as many tables as we had for the first version in the database, we are supposed to read the data from a single table. This single table only allows us to understand the parent-child relationships. Due to this, we need to somehow determine the levels of the nodes. To do this, we have implemented the class Levels_Determination.java. This class, with the help of the other classes in this program, creates a level zero. The level zero has only one node and that is what we have determined as the Root. After allocating level zero to the Root, and based on the parent-child relationships, the program determines other levels.

3.2.2.2 Assign_Children.java: After we read the data from the database, we need to determine all the children of each node and assign those to the parents. In another word, we need to add the names of children to the vector children in the object of class Association, which is actually our node. The class Assign_Children.java does this task for us.

3.2.2.3 Eliminate_Duplicates.java: One of the problems that we might encounter is to have some duplicate nodes. In another word, if we have multiple parent-child relationships for a node, then we will create multiple copies of those nodes. The problem gets more complicated when a node is represented in two different levels. This can also happen if a node is the child of two different parents, which are in two different levels. This can provide a completely confusing diagram for our end users. To avoid such a situation, we eliminate the duplicate nodes by just preserving one of the copies and adding the name of all parents to its parentName vector.

On the other hand, if we have the same node in two different levels of the diagram, we will eliminate the node in the higher level and keep the one in the lower level. Using this method, we avoid the existence of horizontal lines to represent the child-parent relationships.

3.2.2.4 Make_Distance_Nodes.java: To have a proper empty space under the children-less nodes, we have created this class. This class creates a proper number of the Distance nodes. These nodes act as the children for the children-less nodes and preserve the empty space right under their children-less parents. This is to help us to provide an acceptable and non-confusing output for our users.

3.2.2.5 MyProject.java: This class is an extension of the class MyProject.java from the first version. However, we can easily observe major modifications from the first version to the second version. In this version, we have given this class the ability to update the database. Once all the nodes and their attributes were determined, we update the content of the tables in the database. Similar to the older version, we have kept the graphical part of the program in this class.

3.2.2.6 Class Read_Links_Data.java: This class has been constructed by modifying the class Read_Data.java from the first version. Once the connection to the database was established, this class gives the program the ability to read the content of the database and create objects of the class Association.java for each node. Once the program creates these objects, it will store the objects in the main storage utility, in this case a vector, for the further use.

3.2.2.7 Root_Level_Initialization.java: Since the existence of the root is a must for this program, we need to determine the location of the root. The location of the root will act as a basis for the further location determination of the other nodes. This class will analyze the content of the database, and will determine the location of the root.

3.2.2.8 Set_Levels.java: After we read the data from the database Links table, we will need to determine the level of each node. As we mentioned earlier, this task has been done by class Levels_Determination.java. On the other hand, this class gives us the ability to avoid the horizontal lines in the graph. Since we don't want to have any horizontal line to represent the parent-child relationship, we will move the child to a lower level and change the values of the related attribute in the instances of the class Association.java.

3.2.2.9 Sort_Nodes.java: This class is to provide a simple sorting ability for the current existing nodes in our main vector. The sorting is done based on the values of the attribute level of the nodes. A bubble sort algorithm has been used to implement this sorting utility.

3.2.2.10 Sort_Nodes.java: Once the primary sorting is done, we will need to have a more sophisticated sorting ability. The reason is the order in which we read the data from the database may not correspond to what we may wish to have as our output. Therefore, we have implemented this class to gives the program an ability to sort the nodes based on the location of their parents in the picture.

3.2.2.11 YCoordinate.java: This is a simple class that calculates the y coordinate of each node in the graph.

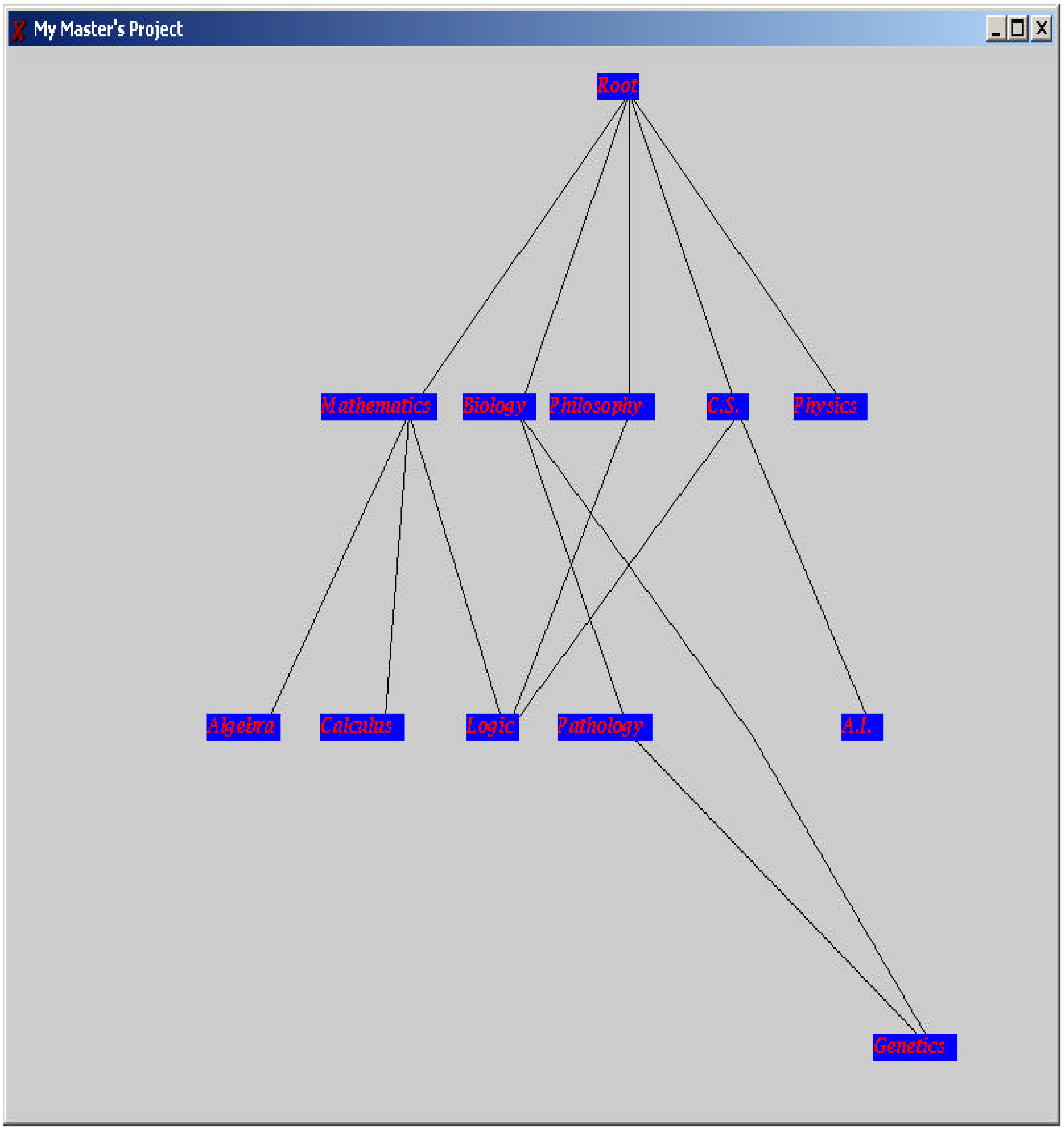
3.2.2.12 Make_Dummy_Nodes.java: In this version of the project, we decided to create dummy nodes whenever we have a parent- child relationship in which the difference of the level between parent and child is more than one. In this class, we have given the software the ability to recognize this kind of parent-child relationships and to create these dummy nodes. We have to mention that the user is not able to see such a dummy node in the output. The only thing that will appear in the output is a line that moves from some upper level to a lower level. However, this line is not going to appear as a straight line, but it will appear as a broken line. The breaking point will be in the intermediate levels. After we create these dummy nodes, we update the database by adding these dummy nodes to the tables. To do so, we remove the primary parent- child relationship, and we create some new relationship using the dummy nodes. For example; if we have a record like the following in the Links table:

Parent	***Child***
Biology	Genetics

in which Biology is in the level one and Genetics in level three, then we will create a dummy node and place the dummy node in the second level of the picture. Then we will update the database in the following manner:

Parent	***Child***
Biology	Dummy Node
Dummy Node	Genetics

In the following page, we can observe an example of the output in a similar program: Looking at the program output, we observe that genetics has two different parents, Biology and Pathology. However, Biology is in the level one and pathology in level two. In such a situation, we create a dummy node as we have already mentioned. The other point about this output is the presence of the Logic, which is the common child among Philosophy, Mathematics, and Computer Science. The third thing to pay attention about is the presence of the empty space right under some nodes, which don't have any children. The examples of this kind of node are AI, Physics, Logic, Calculus, and Algebra.



In addition, we should mention the content of the database before and after the execution of the program. The only table present in the database before the execution of the program is the Links table. Here is the content of the table Links before the execution of the above-mentioned program:

Parent	***Child***
Root	Biology
Mathematics	Algebra
Mathematics	Calculus
Root	Mathematics
Root	Philosophy
Root	C.S.
Biology	Genetics
Mathematics	Logic
C.S.	Logic
C.S.	A.I.
Biology	Pathology
Philosophy	Logic
Pathology	Genetics
Root	Physics

However, after the execution of the program, we will have some additional tables:

Table NodexLevel

Node	***X-Coordinate***	***Level***
Genetics	733	3
A.I.	685	2
DummyNode1	592	2
Pathology	500	2
Logic	407	2
Calculus	314	2
Algebra	222	2
Physics	667	1
C.S.	583	1
Philosophy	500	1
Biology	416	1
Mathematics	333	1
Root	500	0

Table NodeLevel

Node	*Level*
Genetics	3
A.I.	2
DummyNode1	2
Pathology	2
Logic	2
Calculus	2
Algebra	2
Physics	1
C.S.	1
Philosophy	1
Biology	1
Mathematics	1
Root	0

Table Nodes

Node
Genetics
A.I.
DummyNode1
Pathology
Logic
Calculus
Algebra
Physics
C.S.
Philosophy
Biology
Mathematics
Root

3.3 The Algorithm

In this section, we will describe the algorithm that we have created and implemented to make the output of the program.

```
//We have three primary parameters to pass to this algorithm. The first one is an integer
//which shows the width of the picture. The second one is the container, which has been
//used to store all the nodes. The third one is another integer, which is to show the highest
//level that has been assigned to the nodes.
```

```
X Coordinate Determination (integer Picture width, Vector storage, integer Level)
{
            //We have determined a constant integer, which is to decide how much
            //empty space we would like to leave between the top of the picture and
            //where the root appears.
            Constant integer const;
            //The next two doubles are used to show the maximum and minimum
            //amount of the values, which could be assigned to the x-coordinates.
            //In the beginning, we initialize these numbers, but later on and for each
            //level we will decrease the difference of these two values.
            Lowest Limit = picture width/ const;
            Highest Limit= picture width – (picture width/const);
            // Using a for loop, we can go through the Vector and determine the
            //number of the nodes that we have in each level. At first, we start with the
            //highest level.
            Integer my;
            Double temp22, temp33;
            For (integer count=Level; count>0; count--)
              {
                  // This is an integer to count the number of nodes in each level.
                  Integer NodeCounter=0;
                   For (int x = 0; x < storage . size ( ); x++)
                  {
                        if( storage. get(x).get Level( ) = = Count)
                        {
                              NodeCounter++;
                        }

                  }
                  // After we find the number of the nodes for each level, we
                  //determine the location of the first node with such a level
                  // in the vector.
                  For (my = 0;my <storage. Size ( );my ++)
                  {
                        if( storage .get( my ). get Level( ) = = count)
                              break;
                  }
                  // Here we are using a temporary variable to keep the value of the
                  // Lowest Limit before we overwrite its value.
                  temp22=Lowest Limit;
                  // Here we are to determine the distance between the nodes in each
                  // Level, and store the value in the variable temp33.
                  temp33=(Highest Limit – Lowest Limit)/(true_counter-1);
                  // After we determine the distance among the nodes, we start from
                  // the point in which we had found the first node in this level and
                  // and we assign the value of the x coordinate of each node.
```

```
                         For (int xxx = my; xxx < my + NodeCounter; xxx++)
                        {
                               (storage . get(xxx)).setX(temp22) ;
                               temp22 =temp22 +temp33;
                        }
                        // After we finish the task for each level, we need to re-initialize
                        //our temporary variables to zero.
                        temp22 =temp33=0;
                        //Now, we are moving to the upper levels, and we are making the
                        // area in which nodes appear smaller and smaller.
                         Lowest Limit = Lowest Limit  + picture width/const;
                         Highest Limit = Highest Limit - picture width/const;
                 }
         }
}
```

This algorithm could be used for both small and large graphs.

CHAPTER FOUR: CONCLUSION AND FUTURE WORK

Based on the work done in our first version, a more sophisticated version of the System was implemented and debugged. The current system takes care of generation and manipulation of tables and creation of more sophisticated outputs. Not only, it allows defining relations between tables, but also it is able to provide us a more sophisticated output from a minimum amount of input data.

This system, which we had started on experimental basis, has now taken shape to develop into a big system. It provides a kind of interface, which allows a casual or professional user to learn a lot about the content of an on-line database and the relationship among its tables. The design of the system is done so as to accommodate further work on it. Additional functionality can easily be added into the system so that the program can provide more sophisticated output. We believe the current algorithm could be used for both small and large graphs.

There are some potential future development work could be done.

1. Now that we have found the location of each node on the graph, we can swipe the location of these nodes to provide the least number of line crossings. Substantial work could be in this part.
2. In the Current version, the data is derived only from one table. However, in the first version, we have been reading the data from multiple tables. There will be an opportunity to work toward combining these two different versions, and give the program an ability to accept different kinds of outputs.
3. As relations can be defined between tables, the part to implement keeping check of referential integrity when rows are added into tables and also handling cascading updates need to be done.

More:

1. Currently both programs are using single-threaded client-server approach. Obviously, it is impractical to run just one client at one time. So these programs need to be extended to be multi-threaded programs, so that more clients can operate concurrently with one server.
2. Since Oracle database is more functional and popular than MySQL, and Oracle is available for use now in the department system, it would be better to switch to use Oracle database before the program evolves to be bigger and more complicated.

APPENDIX A: USER MANUAL

1. System Requirements

Package name	location on diablo
JDK 1.1.6	/usr/local/jdk1.1.6/
JDK 1.2.2	/usr/local/jdk1.2.2/
MySQL database server	/usr/local/mysql/
JDBC-MySQL driver	/usr/local/mysql/twz1/
Plug-in	Java 2 Runtime Environment Version 1.2.2, or higher

2. Installation

1) Set up:

- Set up server machine environment. Add the following paths to your .tcshrc file
  setenv CLASSPATH
  /usr/local/mysql/twz1/jdbc/mysql:/usr/local/mysql:/usr/local/java/lib/class.zip

- The program should have information about the current user name, database account password, and the name of the database. An example is as follows,
  user = tarokh
  password = project
  db = tarokhdb

2) Compile the server program and initialize the database. In the server directory, run the following commands,

```
/usr/local/jdk1.1.6/javac setupTable.java
/usr/local/jdk1.1.6/java setupTable
```

3) Determine the name of the root.

Root_name = “Science”;

1. Execution

   1) Run client. Start the client program with entering java Main to the command line.

   2) My URL is: http://www.cs.fsu.edu/~tarokh/project will have enough information to help the future users.

2. How to manually access the database

To access the database manually using SQL statements, the user can use the *mysql* command provided in the MySQL server. The following command connects to the MySQL database under the command line. Once the database is connected, you can use standard SQL statement to manually access your database tables.

```
dbsrv> /usr/local/mysql/bin/mysql –u tarokh -p tarokhdb
Enter password: project
```

APPENDIX B: DATABASE DESIGN

Links:

Column Name	*Type*	*Description*
Parent	VARCHAR(30)	Stores the name of the parent.
Child	VARCHAR(30)	Stores the name of the Child.

NodexLevel

Column Name	*Type*	*Description*
Node	VARCHAR(30)	Stores the name of the Node.
X	NUMBER	Stores the X-Coordinate of the Node.
Level	NUMBER	Stores the Level of the Node.

NodeLevel

Column Name	*Type*	*Description*
Node	VARCHAR(30)	Stores the name of the Node.
Level	NUMBER	Stores the Level of the Node.

Nodes

Column Name	*Type*	*Description*
Node	VARCHAR(30)	Stores the name of the Node.

REFERENCES

[1] Larry Stephens, Michael Huhns, “Consensus Ontologies Reconciling the semantics of webpages and agents”, October 2001, IEEE Internet Computing, Pages: 92 –95.

[2] Kevin Mukhar, David Shanes, James De Carli, Todd Lauinger, Ron Phillips, “Beginning Java Databases: JDBC, SQL, J2EE, EJB, JSP, XML”, August 2001, Wrox Press Inc, ISBN 1861004370

[3] Ivor Horton, “Beginning Java 2 SDK 1.4 Edition”, 2002, Wrox Press Inc, ISBN: 1861005695

[4] Cay S. Horstmann, Gary Cornell, “Core Java Vol. I, II”, Sun Microsystems Press, 1999, ISBN 0130819344.

[5] Bruce Eckel, “Thinking in Java”, 2nd Edition, Prentice Hall, 2000, ISBN 0130273635.

[6] Online java tutorial at http://java.sun.com/docs/books/tutorial/

[7] Paul DuBois, Michael Widenius, “MySQL”, New Riders Publishing; ISBN: 0735709211; 1st edition (December 28, 1999)

[8] Harvey M. Deitel, Sean E. Santry, Paul J. Deitel, “Advanced Java 2 How to Program”, Prentice Hall; ISBN: 0130895601; 1st edition (September 15, 2001)