ABSTRACT

A VISUALIZATION SYSTEM FOR INFORMATION RETRIEVAL AND MINING IN HIGH DIMENSIONAL DATABASES

by

Xinhuan Zheng

In this thesis, we present a search engine capable of giving good heuristic answers to the queries on a structural database. A structural database holds structural objects, e.g., protein secondary and tertiary structures, 3D molecules, phylogenetic trees, neuroanatomical networks, parse trees, CAD/CAM parts, and XML documents. Answering queries on such databases often requires solving variants of the graph isomorphism or subisomorphism problems. We also describe a graphic user interface which interacts with users to facilitate visualizing query results. We use 3D molecules (graphs) as illustrating examples, though our prototype is able to handle many other different types of structural data.

A VISUALIZATION SYSTEM FOR INFORMATION RETRIEVAL AND MINING IN HIGH DIMENSIONAL DATABASES

by
Xinhuan Zheng

A Master's Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 2000

APPROVAL PAGE

A VISUALIZATION SYSTEM FOR INFORMATION RETRIEVAL AND MINING IN HIGH DIMENSIONAL DATABASES

Xinhuan Zheng

Dr. Jason T. L. Wang, Master's Thesis Advisor — Date
Associate Professor of Computer and Information Science, NJIT

Dr. D. C. Douglas Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Xiong Wang, Committee Member — Date
Special Lecturer of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Xinhuan Zheng

Degree: Master of Science in Computer Science

Date: January 2000

Education:

- Master of Science in Computer and Information Science
  New Jersey Institute of Technology
  Newark, New Jersey, January 2000.

- Bachelor of Science in Information Science
  College of Arts and Sciences of Beijing Union University
  Beijing, P.R. China, July 1995.

Publications:

"An Approximate Search Engine for Structural Databases," with J. T. L. Wang, Xiong Wang, Dennis Shasha, Bruce A. Shapiro, Kaizhong Zhang, and Zasha Weinberg. Submitted to *SIGMOD 2000*, Dallas, Texas.

ACKNOWLEDGMENT

I would like to take this opportunity to thank Dr. Jason T. L. Wang for his guidance for my master thesis. I want to thank Dr. D. C. Douglas Hung for being my committee member. I would like to give my thanks to Dr. Xiong Wang who is one of my committee members and gives me many good ideas regarding the design and implementation of the graphic interface. I would like to thank George J. S. Chang who gives me his technical supports. I also want to thank my husband, Qicheng Ma, for his help and support in the implementation of the graphic interface and the writing of my thesis. Without their help, it is impossible to finish my thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

A structural database is one whose data objects include trees, graphs, or sets of interrelated labeled points in two, three, or higher dimensional space. Examples include databases containing (i) protein secondary and tertiary structures, (ii) phylogenetic trees, (iii) neuroanatomical networks, (iv) parse trees, (v) CAD/CAM parts, and (vi) XML documents. Comparison queries on such databases require solving variants of graph isomorphism or subisomorphism problems, for which all known algorithms are exponential, so we have explored a large heuristic space.

This thesis presents a search engine capable of giving good heuristic answers to the following queries on a structural database $\mathcal{D}$:

(1) [object-to-object query] Given a query object o, which items o' in $\mathcal{D}$ are within a certain distance d of o or which are closest to o?

(2) [object-to-subobject query] Given a query object o, which items o' in $\mathcal{D}$ have substructures within a certain distance d of o or which are closest to o? We will show how to locate the substructures.

(3) [discovery query] (slower than search) Which substructures approximately occur in all the items in $\mathcal{D}$? We will show the alignment between the substructures and each item of $\mathcal{D}$.

Structural databases occur in various fields such as those dealing with phylogenetic trees [14], 3D molecules [12, 15, 16], and Web-based documents. The object-to-object query arises in information retrieval [2], multimedia [6], molecular biology [17], and elsewhere. The object-to-subobject query is motivated by the study of chemical informational retrieval where one is interested in substructure search in a

database of graphs [10]. The discovery query finds many applications in knowledge discovery and data mining [1, 3, 4, 5, 8, 9, 13, 18, 20]. In what follows, we use 3D molecules (graphs) as illustrating examples, though our prototype is able to handle many other different types of structural data.

1.2 Examples of Applications

In the thesis, we give examples from the search through molecules in the drug database maintained in the National Cancer Institute [12]. In drug discovery, several approaches have been put forth to identify lead compounds. One approach is to create a diverse set of unique structures that spans the entire structural space using the smallest representative set of compounds. By employing the queries described here, one is able to determine those structures. The underlying graph matching algorithms can help chemists in the combinatorial chemistry field to design diverse sets within structural families. Once a suitable lead compound is found, either through database scanning or biological screening, the graph matching algorithms can be used in database searches to find similar structures, substructures, or superstructures of the lead compounds that would have similar biological properties.

The rest of the thesis is organized as follows. Chapter 2 describes the system architecture and structural matching algorithms. Chapter 3 outlines the system design and the control flow chart of each component of the system. Chapter 4 describes the class hierarchy and the techniques for visualization of 3D objects. We conclude this thesis in Chapter 5. A user manual is listed in Appendix A and the source code in Appendix B.

CHAPTER 2

SYSTEM ARCHITECTURE

2.1 Overview of System Architecture

Figure 2.1 shows the architecture of our system. The Graphical Interface module accepts a query, parses it and sends it to the query processor. The Query Processor module analyzes and processes the query. It invokes a collection of filters and structural matching algorithms. The filters include data structures that give sublinear search ability (*MetricMap* [18]). At the end, the structural matching algorithms perform an exact calculation if possible (when comparing trees) or an inexact calculation (when comparing graphs).

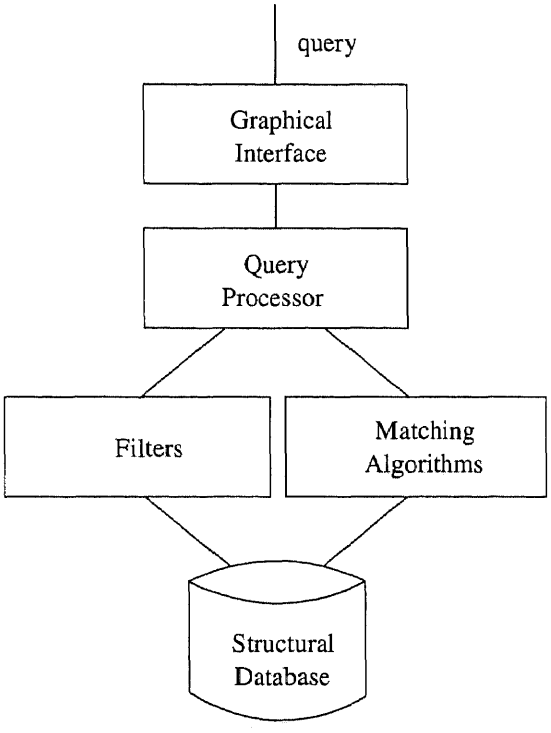

Figure 2.1 System Architecture

2.2 Structural Matching Algorithms

The structural matching algorithms calculate the "distance" or the "similarity" of two trees or graphs. One distance measure we use is a generalized notion of the string edit (Levenshtein) distance [17] for comparing two sequences. The definition of the tree edit distance is given in [19]. For two 3D graphs, we define their distance to be the minimal number of edit operations needed to transform one graph to the other. The edit operations include relabeling a node, deleting a node and inserting a node. Relabeling a node v means to change the label of v to any valid label that differs from its original label. Deleting a node v from a graph means to remove v from the 3D Euclidean space and make the edges touching v connect with one of its neighbors v' (this amounts to contraction of the edge between v and v'). Inserting a node v into a graph means to add v to the 3D Euclidean space and make a node v' and a subset of its neighbors become the neighbors of v. Notice that when a node v is inserted or deleted, the nodes surrounding v do not move, i.e., their coordinates remain the same. Since finding the distance between two graphs amounts to approximate subgraph isomorphism for which only exponential algorithms are known, we have developed a heuristic package involving a large family of filters and valence heuristics.

2.3 Geometric Hashing

One of our techniques for heuristic graph matching follows the geometric hashing paradigm [20]. This technique hashes and compresses node-triplets of 3D graphs into a disk-based table. The algorithm works by storing a coordinate frame in each hash table entry, and using the coordinates to determine matches between node-triplets of a query graph and a data graph. By augmenting node-triplet matches, the system is able to detect whether there is a match between the query graph and the data graph. In processing the discovery query, the system can find approximately

common substructures in a database of 3D graphs without prior knowledge of their structures, positions, or occurrence frequency.

2.4 MetricMap

The *MetricMap* data structure [18] maps the objects (or subobjects) to points in a high-dimensional target space in such a way that the distances among the (sub)objects are approximately preserved. Given a query object, the system decomposes the query object to subobjects and maps the subobjects to the same target space. The algorithm then conducts searches in the target space. to partition the points in the target space.

CHAPTER 3

GRAPHIC INTERFACE DESIGN

3.1 Overview of Graphic Interface

The Graphic Interface module of the system consists of three components: Database Browser, Pattern Discovery, and Object and Subobject Search. The Database Browser component provides the users with text-based or image-based visualization of the data objects from the given structural database, for example, a 3D molecule database. Pattern Discovery answers query (3) described in Chapter 1. For the 3D molecule database, the substructures, or patterns, may occur in the molecule items in which the users are interested. If there exist qualified patterns, all of them can be discovered and visualized in terms of images. The alignment between each found pattern and a substructure in the molecule can also be shown. Object and Subobject Search answers query (1) and query (2) described in Chapter 1. For the 3D molecule database, a target molecule needs to be specified. This component carries out two operations: best match retrieval and substructure search. When applying the query (1) to the 3D molecule database, the query is that given a target molecule o, which molecule o' in the 3D molecule database $\mathcal{D}$ is within a certain distance d of o or which is closest to o. This is what the best match retrieval does. When applying the query (2) to the 3D molecule database, the query is that given a target molecule o, which molecule o' in the 3D molecule database $\mathcal{D}$ has substructures within a certain distance d of o or which are closest to o. This is what the substructure search does. The detailed descriptions of the components are given below.

3.2 Database Browser Component

Figure 3.1 shows the control flow for the Database Browser component. When entering this component, the user selects a data object from the database. Database Browser

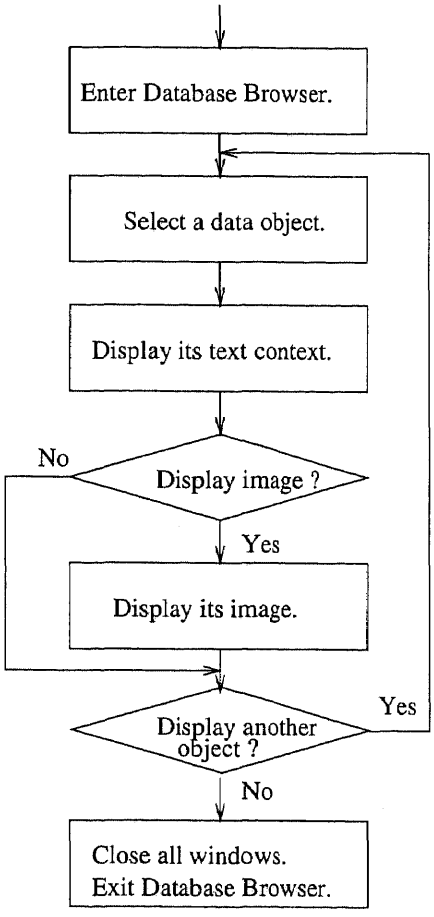

Figure 3.1 Database Browser Component

will display the text contents of the selected data object. The image of the selected data object can also be displayed by clicking the image control button.

3.3 Pattern Discovery Component

Figure 3.2 illustrates the control flow of the Pattern Discovery component. It answers query (3), that is, which substructures approximately occur in all or some of the objects in database $\mathcal{D}$? If there is one, what is it? First, the user is prompted to choose three different molecules, which are then displayed in a window. The user can rotate the molecule image to determine the appropriate parameter values, i.e., the size of the pattern, the occurrence number, and the mutation number, when invoking

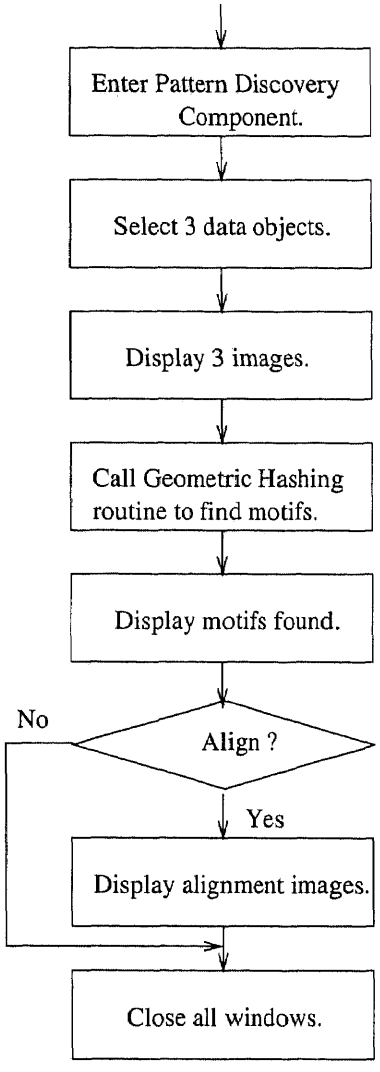

Figure 3.2 Pattern Discovery Component

the Geometric Hashing routine. What the Geometric Hashing routine does is to find the qualified patterns occurring in the chosen three different molecules. Here we use "motif" and "pattern" interchangeably. Motifs can be found by the Geometric Hashing routine. The size of the pattern refers to the minimum size of interesting patterns. The occurrence number refers to the minimum occurrence number required whereas the mutation number refers to the maximum number of mutation allowed. If there are qualified motifs, the text contents and the image of the first motif found are displayed in a window. The user can choose to display any other motif found by

clicking the identification number of the motifs. The Geometric Hashing routine also generates the alignment information between the motif and the molecules in which the motif occurs. The alignment image between one of the molecule and the motif can be displayed by clicking the align control button.

When displaying several images in one window, we make use of multithreading programming technique. Multithreading means that multiple dispatch routines can occur concurrently in one user process. In our case, displaying several images simultaneously implies creating one thread for each image. Furthermore, each thread should have the same internal structure and functions. There is no need for them to communicate since they can exist independently. Using this mechanism, we can focus on the individual thread at a time.

3.4 Object and Subobject Search Component

Figure 3.3 describes the control flow of the Object and Subobject Search component. This component is for query (1) and (2) described in Chapter 1.

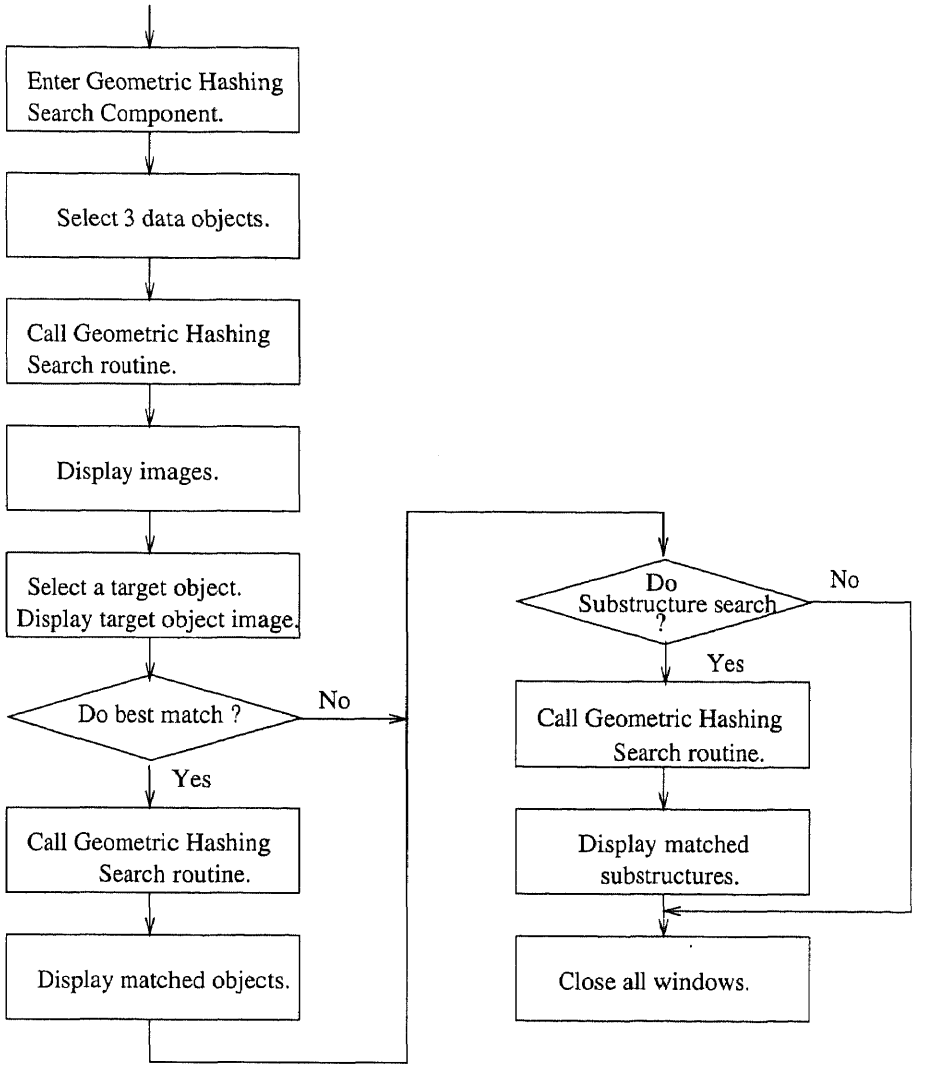

Figure 3.3 Object and Subobject Search Component

As in the Pattern Discovery component, the user is prompted to choose three different molecules, which are then displayed in a window. The three different molecules are put into a hash table by the Geometric Hashing Search routine. Two types of queries can be processed by the Geometric Hashing Search routine, i.e., best match retrieval query and substructure search query. To issue a best match retrieval

query, the user chooses a target molecule and then the query processor is invoked to execute the matching algorithms. It returns the molecule that is closest to the target molecule. It also generates the alignment information between the molecule returned and the target molecule. The user can choose to see the alignment image by clicking the best match menu item. For a substructure search query, the processing is similar to that of a best match retrieval query. The user is also required to choose a target molecule. Then the query processor is invoked to execute the matching algorithms. It returns the molecule containing a substructure that is closest to the target molecule. At the same time, it generates the alignment information between the molecule returned and the target molecule. The alignment image is displayed by clicking the substructure search menu item.

CHAPTER 4

CLASS DESIGN SPECIFICATION

4.1 Class Hierarchy

The graphic interface of the system is designed using the object-oriented programming methodology. In the object-oriented programming methodology, part of the state of a user interface object is its *class*, which determines its general behavior. An object is called an *instance* of its class, and classes act as templates for the construction of new instances. A class of user interface objects may have subclasses; the class is called a superclass relative to its subclasses, which inherit from it. The graphic interface of the system consists of three components that are discussed in the last chapter. From the user point of view, the characteristics of the user interface [21] can be summarized as follows.

- Users perceive and act on objects. The user interface objects must provide appearance and behavior for users' tasks. The behavior of the objects includes visual appearance, changes in appearance, and responses to user actions such as mouse clicks.

- Users can classify objects based on how they behave. This does not imply that a user could write down a class hierarchy, but simply that users will make predictions based on how they classify what they see and interact with.

- In the context of what users are trying to accomplish, all the interface objects fit together into a coherent overall representation. This means that the composition of the interface objects provides users with a representation that is meaningful in context. No object is an island.

There exist two sorts of composition of the interface objects:

- Objects contain other objects. For example, a window contains panels, each of which contains image. When such a window is modeled as an abstract data type, it has not only the characteristics of a general window in the graphics context but also its own distinct point, i.e., the images contained. It reveals a relationship called *is* relationship in object-oriented programming methodology. The *is* relationship indicates that one class can be derived from another class based on inheritance mechanism.

- Objects are constructed from other objects. For example, a window on the display is built from a title bar, a border, scroll bars, etc. A chemistry molecule is built from atoms, bonds, and matrices. This kind of relationship is called *has* relationship.

In contemporary object-oriented programming languages, Java provides its Abstract Window Toolkit (AWT) package as a solution to the development of the graphic user interface. In the AWT package, all kinds of the graphic components are organized in terms of class hierarchy. Each component can be regarded as a class such as a window class, a dialog class, a panel class, a button class, etc. Each class encapsulates its own data members and methods. The methods are the service provided by the object of that kind of component whereas the data members are the ones on which the methods operate. For a class, its methods can be overridden by the subclass' methods which have the same name as that of superclass. For example, the window class has a method called paint. When one wants to put the images into the window, it is realized by deriving a subclass inherited from the window class and overriding the paint method of the window class. Based on the consideration of the object-oriented user interface, we choose the AWT package in Java Development Kit (JDK) with version 1.1.7 to implement the interface of the system.

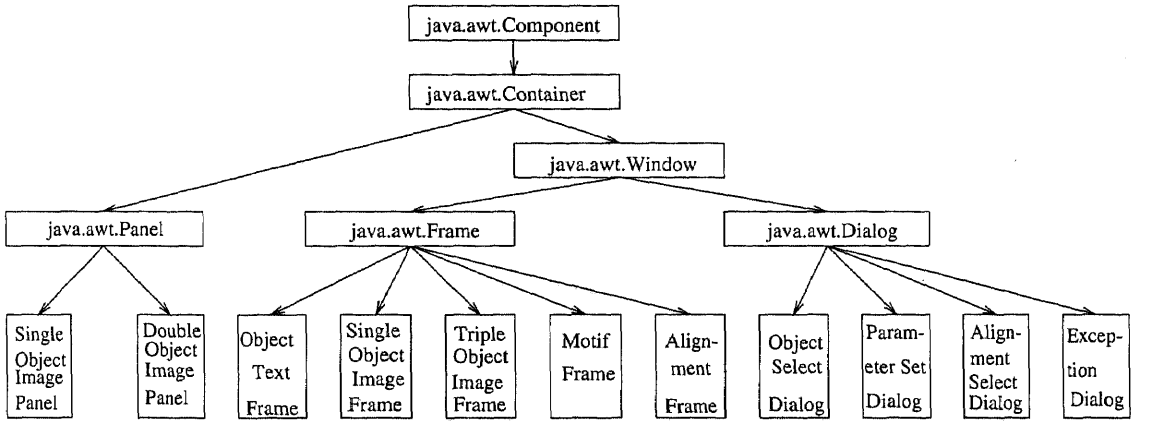

Figure 4.1 Class Hierarchy

Figure 4.1 presents the overview of the class hierarchy in which an arrow represents an "is" relationship. The root of the figure, java.awt.Component, is the parent of all the AWT components. It is an abstract class meaning that it is impossible to create an object of its own. The second level from the top is also a kind of abstract class, java.awt.Container. The containers in the graphic environment can hold components. The components of a container can also be containers. Thus, we can put some components in one container and other components in another container. Finally we can encapsulate all these containers into a higher level container. With the aid of a layout manager, we can construct different windows of different appearance. java.awt.Panel and java.awt.Window are the children of java.awt.Container. The panel looks like a painter's canvas. java.awt.Window is a rectangle region without a title bar. The children of java.awt.Window are java.awt.Frame and java.awt.Dialog. Both of them have features to be maximized, minimized, resized, and moved. java.awt.Dialog can be modaled meaning that java.awt.Dialog can block the input to the parent window when shown, while java.awt.Frame can not be modaled.

We designed all the leaf node classes in the class hierarchy tree. Single Object Image Panel and Double Objects Image Panel class are derived from Panel. We create

an object of class, Single Panel, when we display images in a window. Double Panel is similar to Single Panel except that it has no mouse motion interface. Thus, it is used to display the image of an object alignment where there is no need to rotate the image. Object Text Frame, Single Object Image Frame, Triple Object Image Frame and Motif Frame are derived from java.awt.Frame. Object Text Frame is designed to hold the text area component. Single Object Image Frame is designed to contain only one image panel, whereas Triple Object Image Frame is used to hold three image panels. Motif Frame is defined to hold both a text panel and an image panel. Finally, the derived classes from Dialog are the customized dialogs. They are used to accept the user's inputs in different situations.

4.2 Stand-alone Classes

The system has four stand-alone classes which do not have parent classes. Molecule Model Class is designed for 3D NCI molecules. Figure 4.2 shows the relationship between Molecule Model Class and other classes. The arrow in the figure represents a "has" relation. That is, an NCI molecule has atoms, bonds, and matrices. Molecule Model Class has attributes such as coordinates in the 3D space, the composed atoms, bonds, the matrices, and the bounding box in 3D space. The bounding box is the smallest imaginary cube into which a molecule is embedded. Atom Class contains data members describing the shape, size, color, and light intensity of an atom. Bond Class records the identities of the connected endpoint atoms. Matrix Class defines transformation operations on coordinates in the 3D space where the 3D molecules are located.

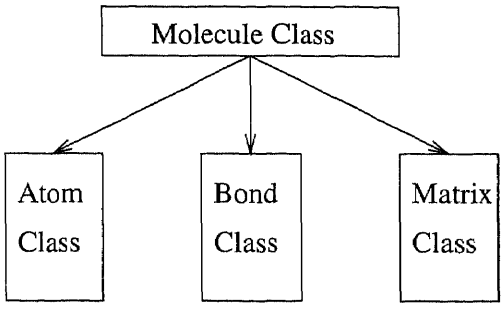

Figure 4.2 The Data Structure for an NCI Molecule

4.3 Algorithm for Visualization of the 3D Molecule Model

4.3.1 Transformation of 3D Coordinates

Each 3D object, e.g., an NCI 3D molecule, which contains many atoms, is displayed in a panel which has no visible boundary. The goal is to map the center of the 3D molecule to the center of a 2D panel. We use the following steps to translate the 3D real world coordinates (left-handed reference system) to 2D screen coordinates in order to visualize 3D molecules [22]. In each step, we calculate the corresponding transformation matrix. The transformation matrix for each step is described below.

Let M_1 be the initial transformation matrix.

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.1}$$

The last column represents the offset of a 3D coordinate to the origin of the 3D world coordinate system. Thus, the offset of the origin of the 3D world coordinate is zero. The unit length is 1.

Let M_2 be the aid matrix, which is the transformation matrix when a 3D molecule is first rotated 20^0 along the y-axis, then rotated 20^0 along the x-axis. The purpose of these rotation is to get a better visual effect of the molecule.

The transpose of the aid matrix M_2 is:

$$M_2^T = M_1^T \begin{pmatrix} \cos 20 & 0 & -\sin 20 \\ 0 & 1 & 0 \\ \sin 20 & 0 & \cos 20 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos 20 & -\sin 20 \\ 0 & \sin 20 & \cos 20 \end{pmatrix} \tag{4.2}$$

When painting or repainting 3D molecules, we conduct the following operations on the transformation matrix M_1.

- Initialize the transformation matrix to M_1. The 3D molecules could have been moved and/or rotated by the user when the 3D molecules are painted(or repainted). So we have to reinitialize the transformation matrix to M_1.

- Calculate the offsets of the center of a 3D molecule to the origin of the 3D world coordinate system. The offsets are denoted by Tx, Ty, Tz. The origin of the 3D world coordinate system is denoted by xo, yo, zo, which are initially 0s. Move the origin of the 3D world coordinate system to the center of the object by adding Tx, Ty, Tz to xo, yo, zo. Essentially we substitute the xo, yo, zo, which are the elements of the last column of M_1, by xo', yo', zo' where xo', yo', zo' are defined by:

$$xo' = xo + Tx,$$

$$yo' = yo + Ty,$$

$$zo' = zo + Tz \tag{4.3}$$

Here Tx, Ty, and Tz are defined respectively:

$$Tx = -(x_{min} + x_{max})/2$$

$$Ty = -(y_{min} + y_{max})/2$$

$$Tz = -(z_{min} + z_{max})/2 \tag{4.4}$$

Hence, we get the transformation matrix M_3.

$$M_3 = \begin{pmatrix} 1 & 0 & 0 & xo' \\ 0 & 1 & 0 & yo' \\ 0 & 0 & 1 & zo' \end{pmatrix} \tag{4.5}$$

In Equation (4.4), x_{min} (y_{min} and z_{min}, respectively) is the minimum of the x-axis (y-axis and z-axis, respectively) coordinates of all the atoms of the NCI molecule, and x_{max} (y_{max} and z_{max}, respectively) is the maximum of the x-axis (y-axis and z-axis, respectively) coordinates of all the atoms of the NCI molecule. The negative sign is used to transform the 3D world coordinate reference system (left-handed coordinate system) to the right-handed coordinate system. This is necessary because the origin of the screen is located at the upper left corner of its own.

- Add a row, $(0, 0, 0, 1)$ to M_3. The result is M_3':

$$M_3' = \begin{pmatrix} 1 & 0 & 0 & xo' \\ 0 & 1 & 0 & yo' \\ 0 & 0 & 1 & zo' \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

Multiply the transformation matrix M_2 by the aid matrix M_3'. The result is M_4.

$$M_4 = M_2 \times M_3' \tag{4.7}$$

- Scale the x-axis, y-axis, and z-axis with the different scale factors of each axis for the transformation matrix M_4 so that the size of the molecule can fit in the size of the panel where the molecule is displayed. For matrix M_4, let X denote the vector of the first row, Y denote the vector of the second row, Z denote the vector of the third row. The scaled vectors are marked as X', Y', Z'. Then $M_5 = (X', Y', Z')^T$. The general transformation form is:

$$X' = X * xf,$$

$$Y' = Y * yf$$

$$Z' = Z * zf \tag{4.8}$$

where xf, yf, and zf represent the scale factor with respect to axis. Their values are specified by:

$$xf = xfac,$$

$$yf = -xfac,$$

$$zf = 16 * xfac/w \tag{4.9}$$

The value of $xfac$ depends on the size of the bounding box of the molecule and the size of the window. The bounding box is the smallest imaginary cube into which the molecule is embedded. Thus, $maxc$, the length of the edge of the bounding box, can be determined by:

$$maxc = max(x_{max} - x_{min}, max(y_{max} - y_{min}, z_{max} - z_{min})) \tag{4.10}$$

The value of $xfac$ is:

$$xfac = 0.7 * scalefudge * max(w/maxc, h/maxc) \tag{4.11}$$

where $scalefudge$ is a specified constant imposed by one who wants to get an appropriate size of the molecule image, usually 1, w and h are the width and height of the panel in which the 3D molecule is displayed, and $maxc$ represents the length of the edge of the bounding box. Scaling in this way ensures the molecule can be displayed within the panel.

- For matrix M_5, calculate the offset to the center of the panel again so that the center of the molecule goes to the center of the panel. The offsets are given by:

$$Tx' = w/2,$$

$$Ty' = h/2,$$

$$Tz' = 8 \tag{4.12}$$

The offset of z-axis is set to 8, the middle index value of an image array which is explained in details in Section 4.3.2. Adding Tx', Ty', Tz' to the last column of M_5, we get the finalized transformation matrix M_6.

- Transform every coordinate vector of the atoms using M_6. In this case, let's denote matrix M_6 by the following:

$$M_6 = \begin{pmatrix} lxx & lxy & lxz & lxo \\ lyx & lyy & lyz & lyo \\ lzx & lzy & lzz & lzo \end{pmatrix} \tag{4.13}$$

the 3D world coordinates of an atom by vector (x, y, z), and the transformed coordinates by vector (x', y', z'). Since x' and y' represent the number of pixels along the horizontal and vertical line of the screen, and z' is the index value of an image array, x', y', and z' are rounded. Thus,

$$x' = lxo + x * lxx + y * lxy + z * lxz$$

$$y' = lyo + x * lyx + y * lyy + z * lyz$$

$$z' = lzo + x * lzx + y * lzy + z * lzz \tag{4.14}$$

We use an example to illustrate the above algorithm.

Assume $x_{min} = -5.554$, $x_{max} = 4.8864$, $y_{min} = -3.4533$, $y_{max} = 4.4558$, $z_{min} = -2.5147$, $z_{max} = 2.412$. The 3D molecule is displayed in a panel with $w = 200$ and $h = 300$, $scalefudge = 1.0$, $(x, y, z) = (4.8864, -2.7377, -0.06)$. Calculate the transformed (x', y', z').

- The initial transformation matrix M_1 is:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.15}$$

The aid matrix M_2 is:

$$M_2 = \begin{pmatrix} 0.9396926 & 0 & 0.34202015 & 0 \\ -0.11697778 & 0.9396926 & 0.32139382 & 0 \\ -0.32139382 & -0.34202015 & 0.8830222 & 0 \end{pmatrix} \tag{4.16}$$

- Tx, Ty, and Tz are:

$$Tx = -(4.8864 - 5.554)/2 = 0.33379984$$

$$Ty = -(4.4558 - 3.4533)/2 = -0.50125$$

$$Tz = -(2.412 - 2.5147)/2 = 0.05134998$$

Then M_3 is:

$$M_3 = \begin{pmatrix} 1 & 0 & 0 & 0.33379984 \\ 0 & 1 & 0 & -0.50125 \\ 0 & 0 & 1 & 0.05134998 \end{pmatrix} \tag{4.17}$$

- M_4 is:

$$M_4 = \begin{pmatrix} 0.9396926 & 0 & 0.34202015 & 0.33123198 \\ -0.11697778 & 0.9396926 & 0.32139382 & -0.49356455 \\ -0.32139382 & -0.34202015 & 0.8830222 & 0.10949959 \end{pmatrix} \tag{4.18}$$

- Calculate $maxc$ and $xfac$.

$$maxc = max(x_{max} - x_{min}, max(y_{max} - y_{min}, z_{max} - z_{min}))$$

$$= max(4.8864 - (-5.554), max(4.4558 - (-3.4533), 2.412 - (-2.5147)))$$

$$= 10.4404$$

$$xfac = 0.7 * 1.0 * max(200/10.4404, 300/10.4404)$$

$$= 14.750392$$

Thus, M_5 is:

$$M_5 = \begin{pmatrix} 13.860834 & 0 & 5.0449314 & 4.885802 \\ 1.7254682 & -13.860834 & -4.740685 & 7.2802706 \\ -0.37925476 & -0.4035945 & 1.0419939 & 0.12921295 \end{pmatrix} \quad (4.19)$$

- Calculate Tx', Ty', Tz'.

$$Tx' = 200/2 = 100, Ty' = 300/2 = 150, Tz' = 8$$

Finally, we get the transformation matrix M_6:

$$M_6 = \begin{pmatrix} 13.860834 & 0 & 5.0449314 & 104.8858 \\ 1.7254682 & -13.860834 & -4.740685 & 124.28027 \\ -0.37925476 & -0.4035945 & 1.0419939 & 8.129213 \end{pmatrix} \quad (4.20)$$

Hence, the (x', y', z') can be calculated according to Equation (4.20), and we obtain $x' = 172$, $y' = 170$, $z' = 7$.

4.3.2 The Algorithm for Generating Projection Effect

When a 3D object is displayed in a panel, some surfaces of the objects are possibly invisible because of the effect of perspective projection. We use the **depth-sorting** method [22] to create the projection effect.

In Section 4.3.1, the coordinates (x, y, z) of the molecule have been converted to (x', y', z'), where x' and y' are the integer coordinates, representing the number of pixels, and z' is the index value of an image array. The image array has 16 atom images of different size. The atom image are a kind of data structure which contains the bit information of the image. The lower the index of an image element, the

smaller its image. The rear atoms of a molecule are displayed with smaller images of lower index values, and the front atoms of a molecule are displayed with larger images of higher index values. Thus, this mechanism creates the projection effect.

When we paint each atom of a molecule, which atom should be painted first? We sort the z' values of the atoms from the smallest to the largest and store the sorted result into a buffer. Next we choose the atom with the smallest z' value, which represents the farthest point of the molecule to draw first. Then the second smallest atom is selected and drawn and so on until the buffer becomes empty. By this means, we paint all the atoms of the molecules.

4.3.3 Shading and Color Models

The intensity of light on different surfaces of an object can be varied in a reasonable way to create a 3D effect. It is related to the light source and the characteristics of the object. We need a **shading** model [22] to reflect the variety of the intensity of the light.

In the shading model, the points closer to the light source are brighter, while the points farther away from the light source are darker. We use the atom image as an example to illustrate the shading model. Each atom is displayed in its circle within a panel. Each point within the circle with radium R is represented by (X, Y) coordinate where $-x_0 < X < x_0$, $0 < Y < 2R$ as illustrated in Figure 4.3.

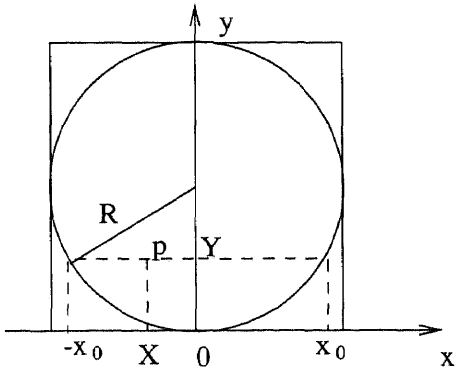

Figure 4.3 The Shading Model of an Atom

In Figure 4.3, x_0 is calculated from Y and R. Its value is given by:

$$x_0 = \sqrt{R^2 - (R - Y)^2} \tag{4.21}$$

We put all the points within the circle in an one dimensional array. The index value p of a point (X, Y) in the circle can be determined by:

$$p = Y * (2 * R) + R + X \tag{4.22}$$

The scan line will scan the circle from top to bottom, and from left to right. The light intensity of every point of the circle along the scan line is given by:

$$I(p) = \sqrt{(X + hx)^2 + (Y - R + hy)^2} \tag{4.23}$$

where $0 < Y < 2R$, $-x_0 < X < x_0$, hx, hy are base values of the light intensity which are constants, e.g., 15 in our example.

When a scan line goes cross the circle, each $I(p)$ along the scan line can be determined from Equation (4.23). In this way, the circle area is scanned by all the scan lines. We then get all the values of $I(p)$ of all the points of the circle.

Besides the shading model, we need the color model to produce colorful image. In video monitor there are three primary colors: red, green, blue, referred to as the **RGB** color model. This color scheme is an additive model: the intensities of the primary colors are added to produce different colors. To create a color, we need to calculate the percentage of red, green, and blue for the given color. Any colors of the objects can be expressed with the triple (R, G, B), where values of R, G, and B are in the range from 0 to 255 (we have 256 kinds of colors in the video monitor). For example, magenta is represented by the triple $(255, 0, 255)$, and yellow is represented by the triple $(255, 255, 0)$.

We use different colors to represent different kinds of atoms. For example, a carbon atom is represented by black; an oxygen atom is represented by red. Since

the light intensity is varied in the circle, the color representing a specific atom will be darker in those areas farther away from the light source and brighter in other areas closer to the light source. In addition, we also have an image array with 16 elements, which are images of an atom in different sizes sorted by distances between the viewer and the atom, so the larger image will appear brighter than the smaller one. The percentage of red, green, and blue is determined by:

$$C(d, b) = bg + ((fg - bg) * d) * b \tag{4.24}$$

where $C(d, b)$ represents the percentage of the primary color red, green, and blue, d is the ratio of the light intensity of the current pixel of the circle to the maximum light intensity of the circle, while b is the ratio of the index of the image to 16, bg and fg represent the background color and foreground color respectively.

CHAPTER 5

CONCLUSIONS

We have implemented the search engine tools for a structural database and the graphic interface based on those search engine tools. We use edit operations to measure the difference between two molecules. We also use Geometric Hashing and *MetricMap* to facilitate graph matching and retrieval. Thus, one can visualize the results that are returned from our search engine tools.

The Graphical Interface module is implemented using the Java Development Kit (JDK) version 1.1.7 run under the UNIX operating system. The other components are implemented using the C programming language.

APPENDIX A

USER MANUAL

This appendix describes how to use our Graphic Interface. First, we show how the Database Browser component works. Then, we discuss Pattern Discovery component. Finally, we illustrate Object and Subobject Search component. We also use some screen-shots to illustrate each component.

- The main menu contains four items: Database, Pdiscover, Gsearch, and Help. The user can continue running the system by clicking one of the four items.

- Under the Database submenu, there are three items: Open, Close, and Exit. Initially the Close item is disabled because nothing is opened yet. As soon as the user clicks Open, a window is popped up with the title "Open Database." This window has three components. The left part is the data list which are names of the molecules in the database. By clicking one of the items in the data list, the user can choose any NCI molecule. The right part is the Text Area which displays the text contents of the chosen NCI molecule. The Text Area lists the molecule's name, the atom's name, the atom's coordinates and their bonds. At the bottom of the window, there are two buttons. One is Image and the other is Cancel. When the Image button is clicked, another window is popped out. It contains the image of the selected molecule. One can rotate the image by moving the mouse. Clicking the Close item will close all the displaying windows and clicking the Exit item will exit the application. Figure A.1 shows the screen-shot of the Database Browser component.

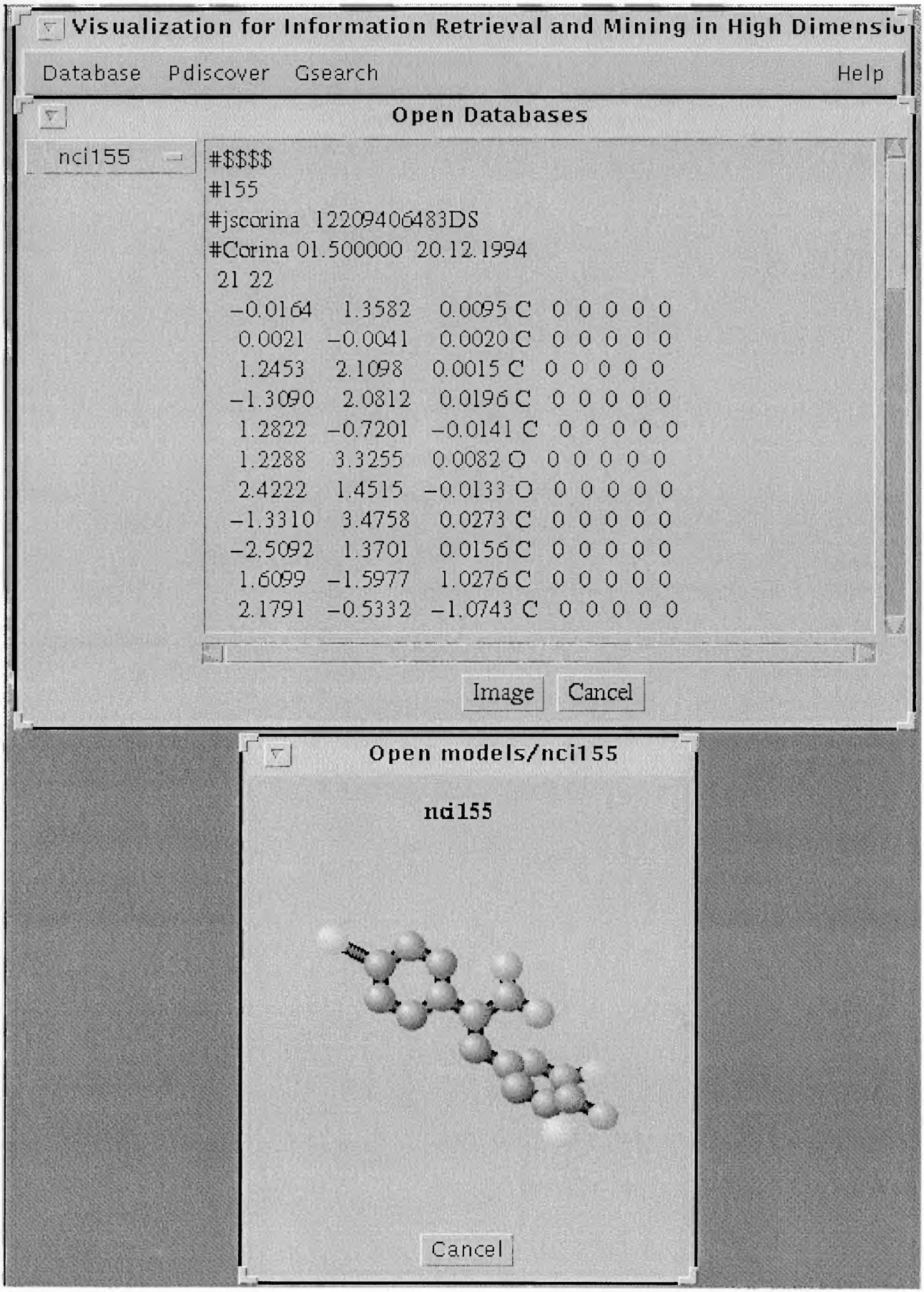

Figure A.1 Database Browser Component Screen-shot.

- Under the Pdiscover submenu, there are two items. One is Select and the other is Run. When the Select item is clicked, a dialog window will be popped out which contains a list of the molecule's names in our database. If the user chooses three molecules, then another window will be popped up. The window contains the images for the chosen molecules. One can rotate any of them. The name of the molecule is listed on top of the corresponding image, along with a button labeled "Text." The user can view the text contents by clicking the Text button next to the molecule names. When the Run item is clicked, the results will be displayed in a separate motif window. The motif window is made up of four parts. The leftmost part is a list including the IDs of the motifs. The user can select any of them to see the discovered motif. In the middle of the window is the text area which displays the motif's text contents. In the rightmost region is the image area which displays the image of the selected motif. At the bottom of the window are two control buttons. One of them is Align. When the Align button is clicked, a dialog window will be popped out to ask the user which molecule to be aligned. Then, one can view the alignment images in a separate window. Figure A.2 illustrates the component.

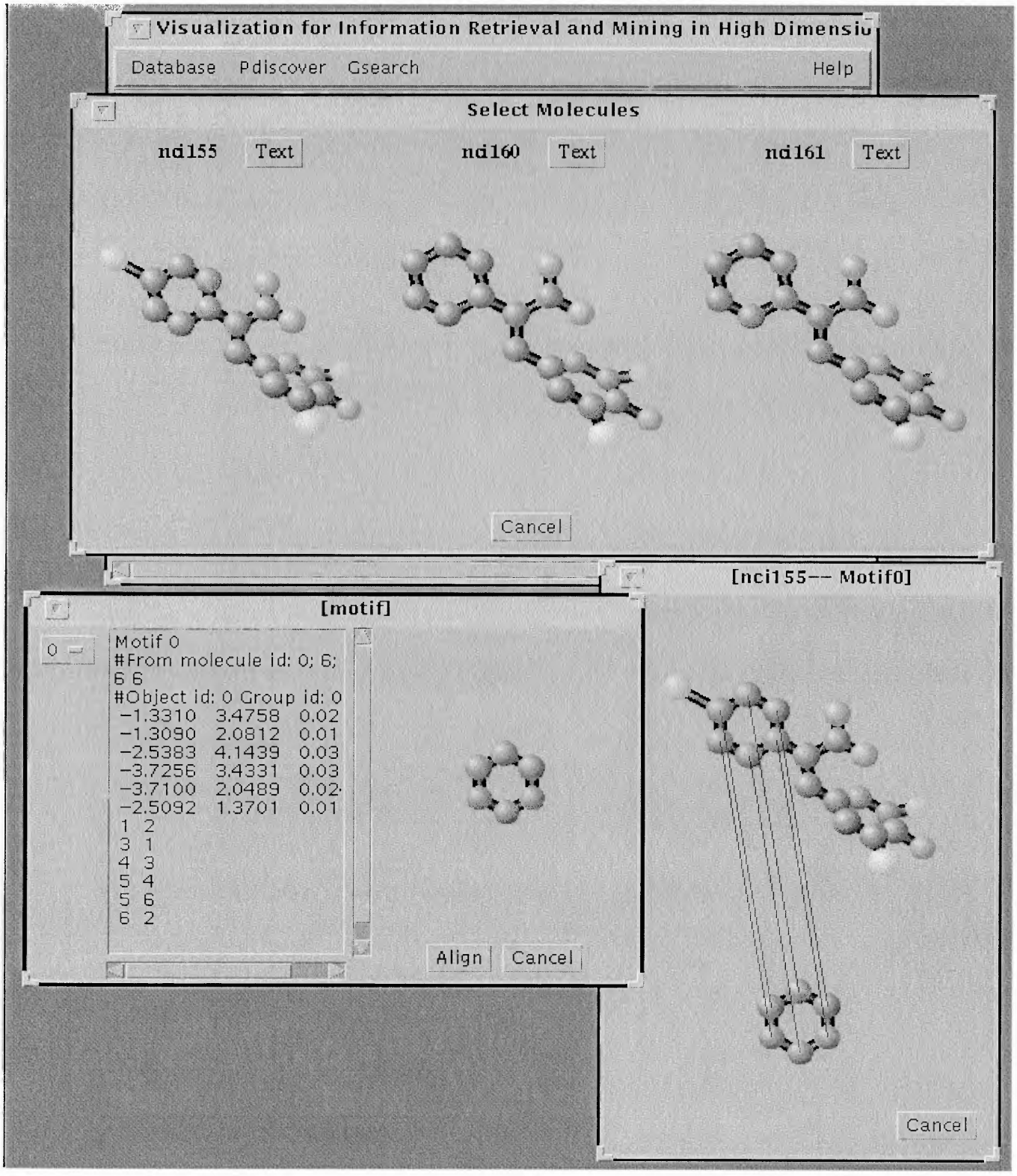

Figure A.2 Pattern Discovery Component Screen-shot.

- Under the Gsearch submenu, there are four items: Hash Data, Select Target, Best Match, and Substructure Search. A dialog window is popped up to allow the user to select three molecules by clicking the Hash Data menu item. Then the images of the chosen molecules will be displayed. Clicking the Select Target item enables the user to choose a target from the database. Then the image for the target molecule will be displayed in a separate window. The user can either select Best Match or Substructure Search to issue the corresponding query. The results of the queries and the alignment images are displayed finally. Figure A.3 illustrates the component.

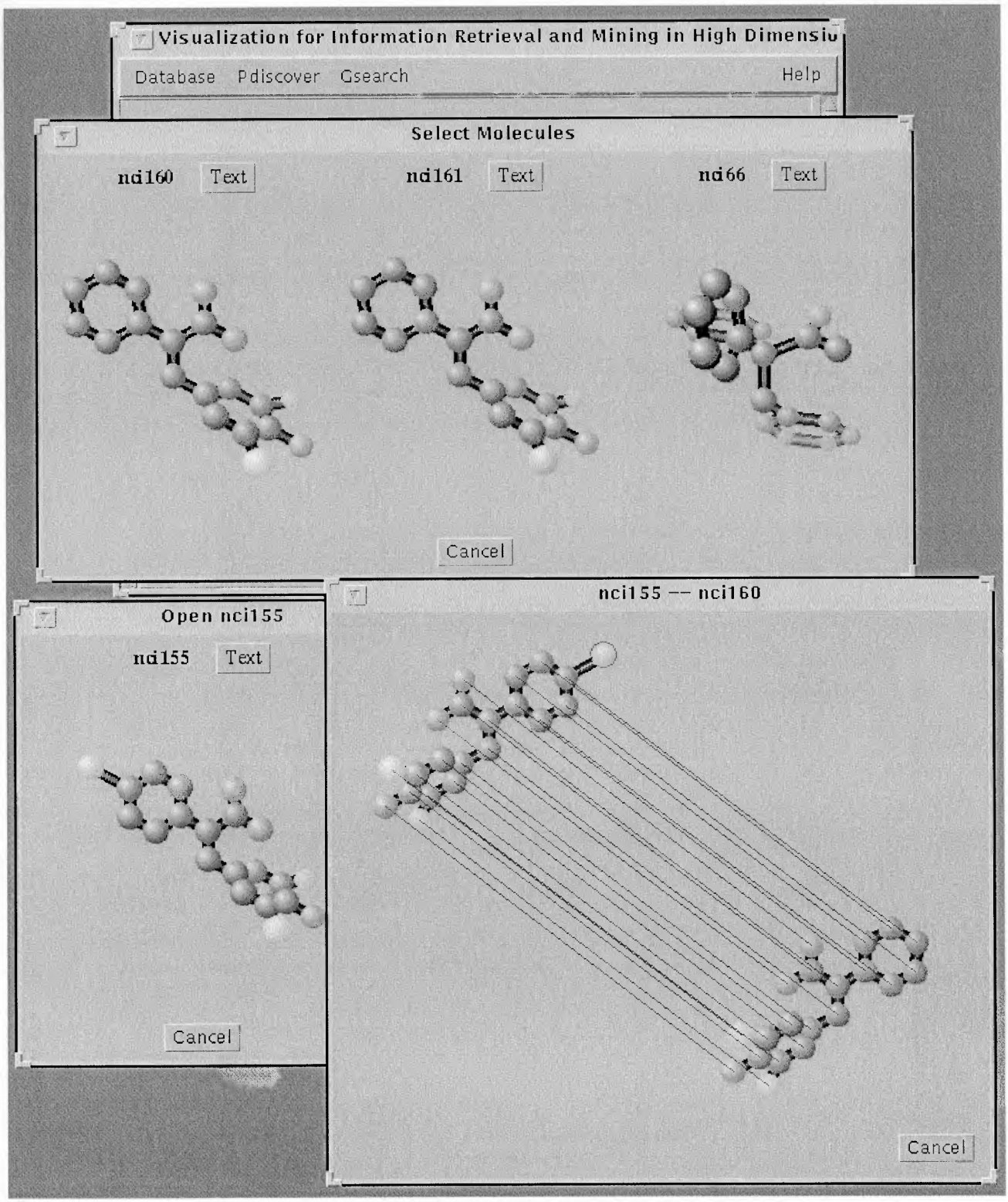

Figure A.3 Object and Subobject Search Component Screen-shot.

APPENDIX B

SOURCE CODE

This appendix includes the source code for implementing the graphic interface.

```
/**
 * @(#)AlignFrame.java       99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.*;
import java.io.*;
import java.net.*;

/**
 * Generate a pop-out window when doing alignment
 */
public class AlignFrame extends Frame {
  GhFrame frame;
  MotifFrame mtframe;
  AlignPanel alignPanel;
  Button cancel = new Button("Cancel");
  Vector objects = new Vector();
  Vector subs = new Vector();
  XYZChemModel obj;
  int an, bn;
  float vt[];
  String at[];
  int rl[];
  boolean ParseError = false, ParseError2 = false;
  AlignPoint ap, ap2 = new AlignPoint();
  String Searched="";
  int Width = 256, Height = 396;
  int Width2 = 456, Height2 = 400;
  String Option=null;
```

```
  AlignFrameListener afl;

  /**
   * Generate a pop-out window in the case of gsearch
   */
  public AlignFrame(GhFrame f, String Target, String opt) {
    super();

    frame = f;
    Option = opt;
    InputStream is = null;
    try {
      if ( "b".equals(Option) ) {
        is = new FileInputStream( Target + ".b" );
        AlignParser2(is);
      }
      else {
        is = new FileInputStream( Target + ".s" );
        AlignParser2(is);
      }

    } catch( Exception e ) {
      SimpleDialog exceptionDialog = new SimpleDialog(frame,
                                        "File Format Error",e.toString());
      exceptionDialog.setVisible(true);
      ParseError2 = true;
    }

    if (!ParseError2) {
      setTitle( Target + " -- " + Searched );
      alignPanel = new AlignPanel(frame, this, "models/" + Target,
                          "models/" + Searched);
      add(alignPanel, "Center");
      Panel control = new Panel();
      add(control, "South");
      control.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
      control.add(cancel);
      afl = new AlignFrameListener( frame, this, 0 );
      cancel.addActionListener( afl );
    }
  }

/**
  * Generate a pop-out window in the case of pdiscover
  */
```

```
public AlignFrame(GhFrame f, MotifFrame mf, String objname, String alino) {
    super("[" + objname + "-- Motif" + alino + "]");

    frame = f;
    mtframe = mf;
    InputStream is = null;
    try {
      is = new FileInputStream( objname + ".molecule" );
      ObjectParser( is );

      is = null;
      is = new FileInputStream( objname + ".align" );
      AlignParser( is );

    } catch(Exception e) {
     SimpleDialog exceptionDialog = new SimpleDialog( frame,
                                        "File Format Error",e.toString());
     exceptionDialog.setVisible( true );
     ParseError = true;
    }

    if ( !ParseError ) {
      alignPanel = new AlignPanel( this, Integer.parseInt( alino ),
                          objects, subs, mf.motifs );
      add(alignPanel, "Center");
      Panel control = new Panel();
      control.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
      control.add( cancel );
      add(control, "South");
      afl = new AlignFrameListener ( frame, this );
      cancel.addActionListener( afl );
    }
  }
  public void addNotify() {

    super.addNotify();

    Toolkit tk = Toolkit.getDefaultToolkit();
    if ( Option == null ) {
      Dimension scsz = tk.getScreenSize();
      Insets insets = getInsets();

      setBounds(scsz.width/2 - Width/2, scsz.height/2 - Height/2,
            Width + insets.left + insets.right,
            Height + insets.top + insets.bottom);
    }
```

```
    else {
      Dimension scsz = tk.getScreenSize();
      Insets insets = getInsets();
      setBounds(scsz.width/2 - Width2/2, scsz.height/2 - Height2/2,
            Width2 + insets.left + insets.right,
            Height2 + insets.top + insets.bottom);
    }
}

/**
 * Parse the format with the postfix .molecule in the case of pdiscover
 */
private void ObjectParser(InputStream is) throws Exception {
  StreamTokenizer st;
  st = new StreamTokenizer(new BufferedReader(new InputStreamReader(is)));
  st.eolIsSignificant(true);
  st.commentChar('#');

  try {

    scan:
    while( true ) {
      switch ( st.nextToken() ) {
        case StreamTokenizer.TT_EOF:
          break scan;
        default:
          break;
        case StreamTokenizer.TT_WORD:
          for (int i=0;i<mtframe.MotifNum;i++) {
            for (int j = 0;j<3;j++)
            st.nextToken();
           if(st.nextToken() == StreamTokenizer.TT_NUMBER) {
             an = (int)st.nval;
             vt = new float[an*3];
             at = new String[an];
             if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
              bn = (int)st.nval;
              rl = new int[bn*2];
              st.nextToken();
            for (int j=0,k=0;j<(an*3)&& k<an;j+=3,k++) {
                if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                 vt[j] = (float)st.nval;
                 if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                   vt[j+1] = (float)st.nval;
                   if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                     vt[j+2] = (float)st.nval;
```

```
                        if (st.nextToken() == StreamTokenizer.TT_WORD) {
                          at[k] = st.sval;
                          st.nextToken();
                        }
                      }
                    }
                  }
                }
                for (int j = 0;j<(bn*2);j+=2) {
                  if(st.nextToken() == StreamTokenizer.TT_NUMBER) {
                    rl[j] = (int)st.nval - 1;
                    if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                      rl[j+1] = (int)st.nval - 1;
                      st.nextToken();
                    }
                  }
                }
                  st.nextToken();
                  st.nextToken();
                }
              }
              obj = new XYZChemModel(an, bn, vt, rl, at);
              objects.addElement(obj);
            }
          } // end switch
        } // end while
        is.close();
      } catch(IOException ioe) {
      } // end try
      if (st.ttype != StreamTokenizer.TT_EOF)
        throw new Exception(st.toString());

    }

  /**
     * Parse the format with the postfix .align in the case of pdiscover
     */
    private void AlignParser(InputStream is) throws Exception {
      StreamTokenizer st;
      st = new StreamTokenizer(new BufferedReader(new InputStreamReader(is)));
      st.eolIsSignificant(true);
      st.commentChar('#');

      try {

        scan:
```

```
while(true) {
  switch( st.nextToken() ) {
    case StreamTokenizer.TT_EOF:
      break scan;
    default:
      break;
    case StreamTokenizer.TT_WORD:
      for (int i=0;i<mtframe.MotifNum;i++) {
        st.nextToken();
        st.nextToken();
        XYZChemModel xm =
          (XYZChemModel)mtframe.motifs.elementAt(i);
        int len = xm.AtomNum;
        ap = new AlignPoint(len);
        for (int j=0;j<len;j++) {
         int x, y;
         if (st.nextToken() == StreamTokenizer.TT_WORD) {
            if ("INSERT".equals(st.sval)) {
              st.nextToken();
              x = (int)st.nval - 1;
              ap.AddIDR(x, true);
              ap.changeLength();
              st.nextToken();
            }
           else if ("DELETE".equals(st.sval)) {
             st.nextToken();
             x = (int)st.nval - 1;
             ap.AddIDR(x, false);
             ap.changeLength();
             st.nextToken();
           }
           else if ("RELABEL".equals(st.sval)) {
             st.nextToken();
             x = (int)st.nval - 1;
             for(int k=0;k<5;k++)
               st.nextToken();
             st.nextToken();
             y = (int)st.nval - 1;
             ap.AddIDR(y, x);
             ap.changeLength();
             for (int k=0;k<4;k++)
               st.nextToken();
           }
         }
         else {
            x = (int)st.nval - 1;
```

```
                    st.nextToken();
                    st.nextToken();
                    if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                      y = (int)st.nval - 1;
                      ap.AddPoint(y, x, j);
                      st.nextToken();
                    }
                  }
                }
                subs.addElement(ap);
                st.nextToken();
                st.nextToken();
              } // end for
            } // end switch
          } // end while
          is.close();
        } catch(IOException ioe) { } // end try

        if (st.ttype != StreamTokenizer.TT_EOF)
          throw new Exception(st.toString());
      }

      /**
       * Parse the format with the postfix .align in the case of gsearch
       */
      private void AlignParser2(InputStream is)
      throws Exception {
        String Result="";
        char buf[] = new char[1024];
        int ptr=0,i,start,end;
        int x = 0, y = 0;
        StreamTokenizer st;

        while((i=is.read())!=10)
          buf[ptr++] = (char)i;
        Result = new String(buf, 0, ptr);
        if ("b".equals(Option)) {
          start = Result.indexOf("matches");
          end = Result.indexOf("with");
          Searched = Result.substring(start+15,end-1);
        }
        else {
          start = Result.indexOf("/");
          end = Result.indexOf("contains");
          Searched = Result.substring(start+1,end-1);
        }
```

```
st = new StreamTokenizer(new BufferedReader(new InputStreamReader(is)));
st.eolIsSignificant(true);
st.commentChar('#');

try {

   scan:
   while(true) {
     switch( st.nextToken() ) {
       case StreamTokenizer.TT_EOF:
         break scan;
       default:
         break;
       case StreamTokenizer.TT_WORD:
         if ( !st.sval.equals("INSERT") && !st.sval.equals("DELETE")
             && !st.sval.equals("RELABEL"))
                 break;
         else {
               if ("INSERT".equals(st.sval)) {
                 if ( st.nextToken() == StreamTokenizer.TT_NUMBER ) {
                   x = (int)st.nval - 1;
                   ap2.AddIDR(x, true);
                 }
               }
               else if ("DELETE".equals(st.sval)) {
                 if ( st.nextToken() == StreamTokenizer.TT_NUMBER ) {
                   x = (int)st.nval - 1;
                   ap2.AddIDR(x, false);
                 }
               }
               else {
                 if ( st.nextToken() == StreamTokenizer.TT_NUMBER ) {
                   x = (int)st.nval - 1;
                 for(int k=0;k<5;k++)
                   st.nextToken();
                 if ( st.nextToken() == StreamTokenizer.TT_NUMBER ) {
                   y = (int)st.nval - 1;
                   ap2.AddIDR(x, y);
                 }
               }
         }
         break;
       case StreamTokenizer.TT_NUMBER:
         x = (int)st.nval - 1;
         st.nextToken();
         st.nextToken();
```

```
            if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
              y = (int)st.nval - 1;
              ap2.AddPoint(x, y);
            }
          while( st.ttype != StreamTokenizer.TT_EOL &&
                 st.ttype != StreamTokenizer.TT_EOF )
                   st.nextToken();
          } // end switch
        } // end while
        is.close();
      } catch(IOException ioe) { } // end try
      if (st.ttype != StreamTokenizer.TT_EOF)
        throw new Exception(st.toString());
    }

    class AlignFrameListener implements ActionListener {
      GhFrame frame;
      AlignFrame af;

      public AlignFrameListener( GhFrame f, AlignFrame af, int pad) {
        frame = f;
        this.af = af;
      }
      public AlignFrameListener( GhFrame f, AlignFrame af ) {
        frame = f;
        this.af = af;
      }
      public void actionPerformed( ActionEvent ae ) {
        if ( Option == null ) {
          frame.alignFrame_1.setVisible( false );
          frame.alignFrame_1 = null;
        }
        else {
          if ( "b".equals(Option) ) {
            frame.hashData.setEnabled(true);
            frame.selectTarget.setEnabled(true);
            frame.alignFrame_2.setVisible( false );
            frame.alignFrame_2 = null;
          }
          else {
            frame.hashData.setEnabled(true);
            frame.selectTarget.setEnabled(true);
            frame.alignFrame_3.setVisible( false );
            frame.alignFrame_3 = null;
          }
        }
```

```
    }
  }
}

/**
 * @(#)AlignPanel.java        99/12/01
 *
 * Author        Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.awt.event.*;
import java.awt.Image.*;
import java.util.*;
import java.io.*;

/**
 * Generate a panel containing the alignment images
 */
class AlignPanel extends Panel {
  GhFrame frame;
  AlignFrame af;
  XYZChemModel obj;
  XYZChemModel mot;
  XYZChemModel target;
  XYZChemModel searched;
  AlignPoint ap;
  float xfac;
  float scalefudge = 0.9f;
  Matrix3D amat = new Matrix3D();
  Matrix3D amat2 = new Matrix3D();
  String message;
  Image ImageBuffer;
  Graphics BackScreen;
```

```
/**
  * Generate a panel in the case of gsearch
  */
 public AlignPanel( GhFrame f, AlignFrame af, String Target, String Searched ) {
   frame = f;
   this.af = af;
   Atom.setPanel(this);
   amat.yrot(180);
   amat2.yrot(180);
   scalefudge = 0.6f;
   InputStream is = null;
   try {
       is = new FileInputStream( Target );
       target = new XYZChemModel(is);
       is = null;
       is = new FileInputStream(Searched);
       searched = new XYZChemModel(is);

   } catch(Exception ex) {
       SimpleDialog exceptionDialog = new SimpleDialog(frame,
                           "File Format Error", ex.toString() );
       exceptionDialog.setVisible(true);
   }
 }

 /**
  * Generate a panel in the case of pdiscover
  */
 public AlignPanel( AlignFrame af, int alino,  Vector objects,
        Vector subs, Vector motifs) {
   this.af = af;
   obj = (XYZChemModel)objects.elementAt( alino );
   mot = (XYZChemModel)motifs.elementAt( alino );
   ap = (AlignPoint)subs.elementAt( alino );
 }
 public void SetupImages() {
   if ( af.Option == null ) {
     if (ImageBuffer == null)
       ImageBuffer = createImage(af.Width, af.Height - 46);
     BackScreen = ImageBuffer.getGraphics();
     BackScreen.setColor(getBackground());
     BackScreen.fillRect(0, 0, af.Width, af.Height - 46);
   }
   else {
     if (ImageBuffer == null)
```

```
      ImageBuffer = createImage(af.Width2, af.Height2 - 36);
    BackScreen = ImageBuffer.getGraphics();
    BackScreen.setColor(getBackground());
    BackScreen.fillRect(0,0,af.Width2,af.Height2 - 36);
  }
}
public void addNotify() {
  super.addNotify();

  if (af.Option==null) {
    setSize(af.Width, af.Height - 46);
    obj.findBB();
    mot.findBB();
    float xw1 = obj.xmax-obj.xmin;
    float yw1 = obj.ymax-obj.ymin;
    float zw1 = obj.zmax-obj.zmin;
    xw1 = (yw1>xw1) ? yw1 : xw1;
    xw1 = (zw1>xw1) ? zw1 : xw1;
    float xw = xw1;
    xw1 = mot.xmax-mot.xmin;
    yw1 = mot.ymax-mot.ymin;
    zw1 = mot.zmax-mot.zmin;
    xw1 = (yw1>xw1) ? yw1 : xw1;
    xw1 = (zw1>xw1) ? zw1 : xw1;
    xw = (xw1>xw) ? xw1 : xw;
    float f1 = getSize().width/xw;
    float f2 = getSize().height/xw;
    xfac =0.7f*(f1<f2?f1:f2)*scalefudge;
  }
  else {
    setSize(af.Width2, af.Height2-36);
    target.findBB();
    searched.findBB();
    float xw1 = target.xmax-target.xmin;
    float yw1 = target.ymax-target.ymin;
    float zw1 = target.zmax-target.zmin;
    xw1 = (yw1>xw1) ? yw1 : xw1;
    xw1 = (zw1>xw1) ? zw1 : xw1;
    float xw = xw1;
    xw1 = searched.xmax-searched.xmin;
    yw1 = searched.ymax-searched.ymin;
    zw1 = searched.zmax-searched.zmin;
    xw1 = (yw1>xw1) ? yw1 : xw1;
    xw1 = (zw1>xw1) ? zw1 : xw1;
    xw = (xw1>xw) ? xw1 : xw;
    float f1 = getSize().width/xw;
```

```
    float f2 = getSize().height/xw;
    xfac = 0.7f*(f1<f2?f1:f2)*scalefudge;
  }
}
public void paint(Graphics g) {
  SetupImages();

  if ( af.Option == null ) {
    Graphics copy = g.create(0, 250, 250, 400);
    try {
      obj.mat.unit();
      obj.mat.translate(
            -(obj.xmin+obj.xmax)/2,
            -(obj.ymin+obj.ymax)/2,
            -(obj.zmin+obj.zmax)/2);
      obj.mat.mult(amat);
      obj.mat.scale(xfac, -xfac, 16*xfac/getSize().width);
      obj.mat.translate(getSize().width/2,
                (getSize().height-100)/2,8);
      obj.transformed = false;
      obj.paint(BackScreen);
      g.drawImage(ImageBuffer,0,0,this);
      mot.mat.unit();
      mot.mat.translate(-(mot.xmin+mot.xmax)/2,
                        -(mot.ymin+mot.ymax)/2,
                        -(mot.zmin+mot.zmax)/2);
      mot.mat.mult(amat);
      mot.mat.scale(xfac, -xfac, 16*xfac/getSize().width);
      mot.mat.translate(getSize().width/2,
                (getSize().height-250)/2,8);
      mot.transformed = false;
      SetupImages();
      mot.paint(BackScreen);
      copy.drawImage(ImageBuffer,0,0,this);
    }
    finally {
      copy.dispose();
    }

    int p, q;
    int sx, sy, ex, ey;
    int [] src = new int[ap.size()];
    int [] end = new int[ap.size()];
    src = ap.getSrc();
    end = ap.getObj();
    for (int i=0;i<ap.size();i++) {
```

```
    g.setColor(Color.red);
    p = src[i];
    q = end[i];
    sx = obj.tvert[p*3];
    sy = obj.tvert[p*3+1];
    ex = mot.tvert[q*3];
    ey = mot.tvert[q*3+1] + 250;
    g.drawLine(sx, sy, ex, ey);
  }
  if (!ap.isEmptyIns()) {
    g.setColor(Color.green);
    src = new int[ap.getInsLength()];
    src = ap.getIns();
    for (int i=0;i<ap.getInsLength();i++) {
      p = src[i];
      sx = obj.tvert[p*3];
      sy = obj.tvert[p*3+1];
      g.drawOval(sx - 10, sy - 10, 18, 18);
    }
  }
  if (!ap.isEmptyDel()) {
    g.setColor(Color.magenta);
    end = new int[ap.getDelLength()];
    end = ap.getDel();
    for (int i=0;i<ap.getDelLength();i++) {
      p = end[i];
      ex = mot.tvert[p*3];
      ey = mot.tvert[p*3+1] + 250;
      g.drawOval(ex - 10, ey - 10, 18, 18);
    }
  }
  if (!ap.isEmptyRel()) {
    g.setColor(Color.blue);
    src = new int[ap.getRelLength()];
    src = ap.getRel();

    for (int i=0;i<ap.getRelLength();i+=2) {
      p = src[i];
      q = src[i+1];
      sx = obj.tvert[p*3];
      sy = obj.tvert[p*3+1];
      ex = mot.tvert[q*3];
      ey = mot.tvert[q*3+1] + 250;
      g.drawLine(sx, sy, ex, ey);
    }
  }
```

```
}
else {
  Graphics copy = g.create(af.Width2/2, af.Height2/2,
              af.Width2/2, af.Height2/2);
  try {
    target.mat.unit();
    target.mat.translate(
          -(target.xmin+target.xmax)/2,
          -(target.ymin+target.ymax)/2,
          -(target.zmin+target.zmax)/2);
    target.mat.mult(amat);
    target.mat.scale(xfac, -xfac, 16*xfac/getSize().width);
    target.mat.translate(getSize().width/4,
                getSize().height/4,8);
    target.transformed = false;
    target.paint(BackScreen);
    g.drawImage(ImageBuffer,0,0,this);
    searched.mat.unit();
    searched.mat.translate(
          -(searched.xmin+searched.xmax)/2,
          -(searched.ymin+searched.ymax)/2,
          -(searched.zmin+searched.zmax)/2);
    searched.mat.mult(amat2);
    searched.mat.scale(xfac, -xfac, 16*xfac/getSize().width);
    searched.mat.translate(getSize().width/4, getSize().height/4-10,8);
    searched.transformed = false;
    SetupImages();
    searched.paint(BackScreen);
    copy.drawImage(ImageBuffer,0,0,this);
  }
  finally {
    copy.dispose();
  }

  int p, q;
  int sx, sy, ex, ey;
  int [] src = new int[af.ap2.size()];
  int [] end = new int[af.ap2.size()];
  src = af.ap2.getSrc();
  end = af.ap2.getObj();
  for (int i=0;i<af.ap2.size();i++) {
    g.setColor(Color.red);
    p = src[i];
    q = end[i];
    sx = target.tvert[p*3];
    sy = target.tvert[p*3+1];
```

```
        ex = searched.tvert[q*3] + af.Width2/2;
        ey = searched.tvert[q*3+1] + af.Height2/2;
        g.drawLine(sx, sy, ex, ey);
      }
      if (!af.ap2.isEmptyIns()) {
        g.setColor(Color.green);
        end = new int[af.ap2.getInsLength()];
        end = af.ap2.getIns();
        for (int i=0;i<af.ap2.getInsLength();i++) {
          p = end[i];
          ex = searched.tvert[p*3] + af.Width2/2;
          ey = searched.tvert[p*3+1] + af.Height2/2;
          g.drawOval(ex - 10, ey - 10, 18, 18);
        }
      }
      if (!af.ap2.isEmptyDel()) {
        g.setColor(Color.magenta);
        src = new int[af.ap2.getDelLength()];
        src = af.ap2.getDel();
        for (int i=0;i<af.ap2.getDelLength();i++) {
         p = src[i];
         sx = target.tvert[p*3];
         sy = target.tvert[p*3+1];
         g.drawOval(sx - 10, sy - 10, 18, 18);
        }
      }
      if (!af.ap2.isEmptyRel()) {
        g.setColor(Color.blue);
        src = new int[af.ap2.getRelLength()];
        src = af.ap2.getRel();
        for (int i=0;i<af.ap2.getRelLength();i+=2) {
          p = src[i];
          q = src[i+1];
          sx = target.tvert[p*3];
          sy = target.tvert[p*3+1];
          ex = searched.tvert[q*3] + af.Width2/2;
          ey = searched.tvert[q*3+1] + af.Height2/2;
          g.drawLine(sx, sy, ex, ey);
        }
      }
    }
  }
  public void update(Graphics g) {
    paint(g);
  }
}
```

```
/**
 * @(#)AlignPoint.java        99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */

/**
 * A stand-alone class recording the alignment IDs.
 */
class AlignPoint {
  private int src[];
  private int obj[];
  private int length;
  private int ins[];
  private int del[];
  private int rel[];
  private int maxna, n1, n2, n3,maxn1, maxn2, maxn3;

  public AlignPoint() {
    length = 0;
    src = null;
    obj = null;
  }
  public AlignPoint(int l) {
     length = l;
     src = new int[l];
     obj = new int[l];
  }
  public int AddPoint(int x, int y) {
    if ( length>= maxna) {
      if (src == null && obj == null) {
         maxna = 25;
```

```
            src = new int[maxna];
            obj = new int[maxna];
        }
        else {
            maxna *= 2;
            int srcTemp[] = new int[maxna];
            System.arraycopy(src, 0, srcTemp, 0, src.length);
            src=srcTemp;
            int objTemp[] = new int[maxna];
            System.arraycopy(obj, 0, objTemp, 0, obj.length);
            obj=objTemp;
        }
    }
    src[length] = x;
    obj[length] = y;
    return ++length;
}
public void AddPoint(int x, int y, int pos) {
    src[pos] = x;
    obj[pos] = y;
}
public int size() {
    return length;
}
public int changeLength() {
    return --length;
}
public int [] getSrc() {
    return src;
}
public int [] getObj() {
    return obj;
}
public int AddIDR(int p, boolean isInsert) {
    if (isInsert) {
        int i=n1;
        if (i>=maxn1) {
            if (ins == null) {
                maxn1 = 20;
                ins = new int[maxn1];
            }
            else {
                maxn1 *= 2;
                int temp[] = new int[maxn1];
                System.arraycopy(ins, 0, temp, 0, ins.length);
                ins=temp;
```

```
        }
      }
      ins[i] = p;
      return ++n1;
    }
    else {
      int i=n2;
      if (i>=maxn2) {
        if (del == null) {
          maxn2 = 20;
          del = new int[maxn2];
        }
        else {
          maxn2 *= 2;
          int temp [] = new int[maxn2];
          System.arraycopy(del, 0, temp, 0, del.length);
          del=temp;
        }
      }
      del[i] = p;
      return ++n2;
    }
  }
  public int AddIDR(int p, int q) {
    int i=n3;
    if(i>=maxn3) {
      if (rel == null) {
        maxn3 = 20;
        rel = new int[maxn3*2];
      }
      else {
        maxn3 *= 2;
        int temp [] = new int[maxn3*4];
        System.arraycopy(rel, 0, temp, 0, rel.length);
        rel=temp;
      }
    }
    rel[i  ] = p;
    rel[i+1] = q;
    return ++n3;
  }
  public int getInsLength() {
    return n1;
  }
  public int getDelLength() {
    return n2;
```

```
}
public int getRelLength() {
   return n3;
}
public int [] getIns() {
   return ins;
}
public int [] getDel() {
   return del;
}
public int [] getRel() {
   return rel;
}
public boolean isEmptyIns() {
   if (ins == null)
     return true;
   else
     return false;
}
public boolean isEmptyDel() {
   if (del == null)
     return true;
   else
     return false;
}
public boolean isEmptyRel() {
   if (rel == null)
     return true;
   else
     return false;
}
public void paramString() {
   for (int i=0;i<length;i++) {
      System.out.println("src["+i+"]="+src[i]);
      System.out.println("obj["+i+"]="+obj[i]);
   }
   if (ins != null) {
      for (int i=0;i<getInsLength();i++)
        System.out.println("ins["+i+"]="+ins[i]);
   }
   if (del != null) {
      for (int i=0;i<getDelLength();i++)
        System.out.println("del["+i+"]="+del[i]);
   }
   if (rel != null) {
      for (int i=0;i<getRelLength();i+=2) {
```

```
            System.out.println("rel["+i+"]="+rel[i]);
            System.out.println("rel["+(i+1)+"]="+rel[i+1]);
          }
        }
      }
    }

/**
 * @(#)Atom.java          99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.awt.Image.*;
import java.util.*;
import java.io.*;
import java.net.URL;
import java.util.Hashtable;
import java.util.Vector;
import java.awt.image.IndexColorModel;
import java.awt.image.ColorModel;
import java.awt.image.MemoryImageSource;
import java.awt.event.*;

/**
 * A stand-alone class generating an atom
 */

class Atom {
   private static Panel panel;
   private static Frame frame;
   private static byte[] data;
   private static Color cl[];
```

```
private final static int R = 40;
private final static int hx = 15;
private final static int hy = 15;
private final static int bgGrey = 192;
private final static int nBalls = 16;
private static int maxr;

private int Rl;
private int Gl;
private int Bl;
private Image balls[];

/**
 * Caculating the light intensity along the scan line within the circle
 * representing an atom
 */
static {
   data = new byte[R * 2 * R * 2];
   int mr = 0;
   for (int Y = 2 * R; --Y >= 0;) {
      int x0 = (int) (Math.sqrt(R * R - (Y - R) * (Y - R)) + 0.5);
      int p = Y * (R * 2) + R - x0;
      for (int X = -x0; X < x0; X++) {
         int x = X + hx;
         int y = Y - R + hy;
         int r = (int) (Math.sqrt(x * x + y * y) + 0.5);
         if (r > mr)
            mr = r;
         data[p++] = r <= 0 ? 1 : (byte) r;
      }
   }
   maxr = mr;
   cl=new Color[255];
   for(int i=0,j=193;i<254;i++)
    { if(j-->0)
       cl[i]=new Color(j,j,j) ;
      else
       cl[i]=new Color(10,10,10) ;
    }
}


static void setPanel(Panel p) {
   panel = p;
}
static void setPanel(Frame f) {
```

```
    frame = f;
  }
  Atom(int Rl, int Gl, int Bl) {
    this.Rl = Rl;
    this.Gl = Gl;
    this.Bl = Bl;
  }
  private final int blend(int fg, int bg, float fgfactor) {
    return (int) (bg + (fg - bg) * fgfactor);
  }

  /**
   * Calculating the color percentage for each pixel within the circle
   * representing the atom
   */
  private void Setup() {
    balls = new Image[nBalls];
    byte red[] = new byte[256];
    red[0] = (byte) bgGrey;
    byte green[] = new byte[256];
    green[0] = (byte) bgGrey;
    byte blue[] = new byte[256];
    blue[0] = (byte) bgGrey;
    for (int r = 0; r < nBalls; r++) {
      float b = (float) (r+1) / nBalls;
      for (int i = maxr; i >= 1; --i) {
        float d = (float) i / maxr;
        red[i] = (byte) blend(blend(Rl, 255, d), bgGrey, b);
        green[i] = (byte) blend(blend(Gl, 255, d), bgGrey, b);
        blue[i] = (byte) blend(blend(Bl, 255, d), bgGrey, b);
      }
      IndexColorModel model = new IndexColorModel(8, maxr + 1,
                                    red, green, blue, 0);
      balls[r] = panel.createImage(
        new MemoryImageSource(R*2, R*2, model, data, 0, R*2));
    }
  }
  void paint(Graphics gc, int x, int y, int r)
  {
        Image ba[] = balls;
        if (ba == null)
        {
          Setup();
          ba = balls;
        }
        Image i = ba[r];
```

```
        int size = 10 + r;
        gc.drawImage(i, x - (size >> 1), y - (size >> 1), size, size, panel);
    }

void paintline(Graphics g, int x1, int y1,int z1,int x2 , int y2,int z2)  {
    int max_r=4,min_r=2,ind;
    float tfdr,tfnr,fnr,fdr,slop,fzi,ax1,ax2,ay1,ay2,tzi;
    int delx,dely,delz,delr,dx,dy;
    int pz1,px1,py1,pz2,px2,py2;
    int a,b,c,d,r,zi;

    if(z1<0) z1=0;
    if(z1>16) z1=16;
    if(z2<0) z2=0;
    if(z2>16) z2=16;
    if(z1<=z2) {pz1=z1;px1=x1;py1=y1;px2=x2;py2=y2;pz2=z2;}
    else {pz1=z2;px1=x2;py1=y2;px2=x1;py2=y1;pz2=z1;}
    delz=(pz2-pz1);
    delx=(px2-px1);
    dely=(py2-py1);
    dx=Math.abs(delx);
    dy=Math.abs(dely);
    if(pz2<=0)
      r=min_r;
    else
      r=max_r-Math.round((delz*max_r)/32);
    if(dy>dx) {
      slop=((float)dx / (float)dy);
     if(dy!=0)
      tzi=(float)(delz*12)/(float)dy;
     else
      tzi=(float)0;
    ax1=px1;zi=pz1*12;fzi=zi;
    for(int i=0;i<dy;i++) {
      if(dely>0)
        py2=py1+1;
      else
      py2=py1-1;
    if(delx>0)
      ax2=ax1+(slop);
    else
      ax2=ax1-slop;
    a=Math.round(ax1);
    b=Math.round(ax2);
    for(int j=r;j>=0;j--) {
     int k=25*j;
```

```
      if((ind=zi+k)>193) ind=193;
      g.setColor(cl[ind]);
      g.drawLine(a+j,py1,b+j,py2);
      g.drawLine(a-j,py1,b-j,py2);
      }
     ax1=ax2;
     fzi=fzi+tzi;
     zi=(int)Math.round(fzi);
     py1=py2;
    }
   }
 if(dy<=dx) {
     slop=(float)dy/(float)dx;
   if(dx!=0)
    tzi=(float)(delz*12)/ (float)dx;
   else
    tzi=(float)0;
   ay1=py1;zi=pz1*12;fzi=zi;
    for(int i=0;i<dx;i++) {
     if(delx>0)
       px2=px1+1;
     else
       px2=px1-1;
     if(dely>0)
       ay2=ay1+(slop);
      else
       ay2=ay1-slop;
      a=Math.round(ay1);
      b=Math.round(ay2);
      for(int j=r;j>=0;j--) {
         int k=20*j;
         if((ind=zi+k)>193) ind=193;
          g.setColor(cl[ind]);
          g.drawLine(px1,a+j,px2,b+j);
          g.drawLine(px1,a-j,px2,b-j);
       }
    ay1=ay2;
    fzi=fzi+tzi;
    zi=(int)Math.round(fzi);
    px1=px2;
   }
  }
 }
}

/**
```

```
 * @(#)GhFrame.java        99/12/01
 *
 * Author        Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.awt.event.*;
import java.awt.Image.*;
import java.util.*;
import java.util.Vector;
import java.io.*;
import java.net.*;

/**
 * Generate the main interface
 */
public class GhFrame extends Frame {
  MenuBar mb;
  Menu Database, Pdiscover, Gsearch, Help;
  MenuItem open, close, exit, select, run, hashData, selectTarget,
       bestMatch, subSearch, about;
  Panel info;
  protected Font InfoFont;
  protected FontMetrics InfoFontMetrics;
  TextArea ta;
  String InfoText;
  boolean TextFileError = false;
  boolean TextFileRead = false;
  int Width = 500, Height = 350;
  NCITextsFrame nciTextsFrame = null;
  NCIImageFrame_1 nciImageFrame_1 = null;
  NCIImageFrame_2 nciImageFrame_2 = null;
  NCIImagesFrame nciImagesFrame_1 = null;
  NCIImagesFrame nciImagesFrame_2 = null;
```

```
NCITextFrame nciTextFrame = null;
MotifFrame motifFrame = null;
AlignFrame alignFrame_1 = null;
AlignFrame alignFrame_2 = null;
AlignFrame alignFrame_3 = null;
ExitDialog exitDialog = null;
SelectDialog selectDialog = null, hashDataDialog = null;
SelectDialog selectTargetDialog = null;
RunDialog runDialog = null;
AlignDialog alignDialog = null;
HelpDialog helpDialog = null;
MenuItemListener ml;
final int NUM = 3;
java.awt.List nciDataNames = new java.awt.List();
Choice nciDataChoice = new Choice();
String [] SelectData = new String[NUM];
String Target;
Vector Text1 = new Vector(), Text2 = new Vector();
final int ImagesType_1 = 1;
final int ImagesType_2 = 2;
final int ImageType = 4;
String dfVls[] = {"6", "1", "3"};
boolean runerror = false;

public GhFrame() {
  super("Visualization for Information Retrieval and Mining
        in High Dimensional Databases");

  setLayout(new BorderLayout());
  mb = new MenuBar();
  Database = new Menu("Database", false);
  Pdiscover = new Menu("Pdiscover", false);
  Gsearch = new Menu("Gsearch", false);
  Help = new Menu("Help", false);
  Database.add(open = new MenuItem("Open"));
  Database.addSeparator();
  Database.add(close = new MenuItem("Close"));
  Database.add(exit = new MenuItem("Exit"));
  Pdiscover.add(select = new MenuItem("Select"));
  Pdiscover.add(run = new MenuItem("Run"));
  Gsearch.add(hashData = new MenuItem("Hash Data"));
  Gsearch.add(selectTarget = new MenuItem("Select Target"));
  Gsearch.add(bestMatch = new MenuItem("Best Match"));
  Gsearch.add(subSearch = new MenuItem("Substructure Search"));
  Help.add(about = new MenuItem("About..."));
  close.setEnabled(false);
```

```
  run.setEnabled(false);
  selectTarget.setEnabled(false);
  bestMatch.setEnabled(false);
  subSearch.setEnabled(false);
  mb.add(Database);
  mb.add(Pdiscover);
  mb.add(Gsearch);
  mb.add(Help);
  setMenuBar(mb);
  mb.setHelpMenu(Help);

  setupList();
  readInfo();
  do {
    try {
      Thread.currentThread().sleep(250);
    } catch(Exception e) {}
  } while ( !TextFileRead );
  info = new Panel();
  info.setLayout(new BorderLayout());
  InfoFont = new Font("TimesRoman", Font.BOLD, 14);
  InfoFontMetrics = getFontMetrics(InfoFont);
  info.setFont(InfoFont);
  info.add(ta = new TextArea(InfoText, 10, 40), "Center");
  ta.setEditable(false);
  add(info, "Center");
  ml = new MenuItemListener(this);
  open.addActionListener(ml);
  close.addActionListener(ml);
  exit.addActionListener(ml);
  select.addActionListener(ml);
  run.addActionListener(ml);
  hashData.addActionListener(ml);
  selectTarget.addActionListener(ml);
  bestMatch.addActionListener(ml);
  subSearch.addActionListener(ml);
  about.addActionListener(ml);
}
private void setupList() {
  String path = System.getProperty("user.dir");
  File Models = new File(path, "models");
  String dataNames[] = Models.list();
  for ( int i = 0; i < dataNames.length; i++) {
      nciDataNames.add( dataNames[i] );
      nciDataChoice.add( dataNames[i] );
  }
```

```
}
private void readInfo() {
  String InfoFileName = "help/About_demo.info";
  int BufPtr;
  String Result;
  InputStream is = null;
  int i=0;
  char c;
  int BufSize = 1024;
  char [] Buf = new char[BufSize];

  try {
    Thread.sleep(20);
  } catch(InterruptedException ie) {}

  BufPtr = 0;
  Result = "";
  try {
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
    is = new FileInputStream(InfoFileName);
    while ( (i = (int)is.read()) != -1 ) {
      c = (char)i;
      Buf[BufPtr++] = c;
      if ( BufPtr >= BufSize) {
        Result += new String(Buf, 0, BufSize);
        BufPtr = 0;
      }
    }
    Result += new String(Buf, 0, BufPtr);
  } catch(Exception e) {
    Result += e.toString();
    InfoText = Result;
    TextFileError = true;
  }
  try {
    if ( is != null)
      is.close();
  } catch(Exception e) {
    Result += e.toString();
    InfoText = Result;
    TextFileError = true;
  }
  if (!TextFileError) {
    InfoText = Result;
    TextFileRead = true;
  }
```

```
}
public void addNotify() {
  super.addNotify();

  Toolkit tk = Toolkit.getDefaultToolkit();
  Insets insets = getInsets();
  Dimension scrnsz = tk.getScreenSize();
  setBounds(scrnsz.width/2 - Width/2,
        scrnsz.height/2 - Height/2,
        Width + insets.left + insets.right,
        Height + insets.top + insets.bottom);
}
public void destroyFrame() {
  NCITextFrame tempt;

  if ( nciTextsFrame != null ) {
    nciTextsFrame.setVisible( false );
    nciTextsFrame = null;
  }
  if ( nciImageFrame_1 != null ) {
    nciImageFrame_1.setVisible( false );
    nciImageFrame_1 = null;
  }
  if ( nciImageFrame_2 != null ) {
    nciImageFrame_2.setVisible( false );
    nciImageFrame_2 = null;
  }
  if ( nciImagesFrame_1 != null ) {
    nciImagesFrame_1.setVisible( false );
    nciImagesFrame_1 = null;
  }
  if ( nciImagesFrame_2 != null ) {
    nciImagesFrame_2.setVisible( false );
    nciImagesFrame_2 = null;
  }
  if ( motifFrame != null ) {
    motifFrame.setVisible( false );
    motifFrame = null;
  }
  for ( int i = 0; i < Text1.size(); i++) {
      tempt = (NCITextFrame)Text1.elementAt(i);
      tempt.setVisible( false );
      Text1.removeElementAt(i);
  }
  for ( int i = 0; i < Text2.size(); i++) {
      tempt = (NCITextFrame)Text2.elementAt(i);
```

```
        tempt.setVisible(false);
        Text2.removeElementAt(i);
    }
    if ( alignFrame_1 != null ) {
      alignFrame_1.setVisible( false );
      alignFrame_1 = null;
    }
    if ( alignFrame_2 != null ) {
      alignFrame_2.setVisible( false );
      alignFrame_2 = null;
    }
    if ( alignFrame_3 != null ) {
      alignFrame_3.setVisible( false );
      alignFrame_3 = null;
    }
}
public static void main(String argv[]) {
    GhFrame ghf = new GhFrame();
    ghf.setVisible( true );
}
class MenuItemListener implements ActionListener {
    GhFrame frame;

    public MenuItemListener(GhFrame f) {
      frame = f;
    }
    private void  exgsearch(String Target, String opt) {
      File file = new File("./temp.gsearch");

      try {
          String cmd = new String("gsearch");

          Runtime rt = Runtime.getRuntime();
          if ("b".equals(opt))
              cmd += new String(" -b models/"+Target);
          else
              cmd += new String(" -s models/"+Target);
          Process prcs = rt.exec(cmd);
      } catch(IOException ioe) {
          SimpleDialog exceptionDialog = new SimpleDialog(frame,
                        "gsearch error", ioe.toString());
          exceptionDialog.setVisible(true);
          runerror = true;
      }

      do {
```

```
    try { Thread.sleep(250); } catch(Exception e) { }
  } while(!file.exists());
  file.delete();
}
public void actionPerformed(ActionEvent ae) {
  MenuItem item = (MenuItem)ae.getSource();

  if ( item == exit ) {
    exitDialog = new ExitDialog(
                frame, "Exit Demo",
                "Are you sure you want to exit?");
    exitDialog.setVisible( true );
  }
  else if ( item == open ) {
    open.setEnabled(false);
    close.setEnabled(true);
    ta.setText("");
    nciTextsFrame = new NCITextsFrame( nciDataChoice, frame );
    nciTextsFrame.setVisible( true );
  }
  else if ( item == close ) {
    destroyFrame();
    open.setEnabled(true);
    close.setEnabled(false);
    select.setEnabled(true);
    run.setEnabled(false);
    hashData.setEnabled(true);
    selectTarget.setEnabled(false);
    bestMatch.setEnabled(false);
    subSearch.setEnabled(false);
  }
  else if ( item == select ) {
      selectDialog = new SelectDialog( frame, nciDataNames,
                  ImagesType_1 );
      selectDialog.setVisible( true );
  }
  else if ( item == run ) {
      runDialog = new RunDialog( frame );
      runDialog.setVisible( true );
  }
  else if ( item == hashData ) {
      hashDataDialog = new SelectDialog(frame, nciDataNames, ImagesType_2 );
      hashDataDialog.setVisible( true );
  }
  else if ( item == selectTarget ) {
      selectTargetDialog = new SelectDialog(frame, nciDataNames, ImageType );
```

```
            selectTargetDialog.setVisible( true );
        }
        else if ( item == bestMatch ) {
            exgsearch(Target,"b");
            if ( !runerror ) {
              if ( alignFrame_2 != null ) {
                alignFrame_2.setVisible(false);
                alignFrame_2 = null;
              }
              alignFrame_2 = new AlignFrame( frame, Target, "b" );
              if (!alignFrame_2.ParseError2) {
                alignFrame_2.setVisible( true );
                hashData.setEnabled(false);
                selectTarget.setEnabled(false);
                bestMatch.setEnabled(true);
                subSearch.setEnabled(true);
              }
            }
        }
        else if ( item == subSearch ) {
            exgsearch(Target, "s");
            if (!runerror) {
              if ( alignFrame_3 != null ) {
                alignFrame_3.setVisible(false);
                alignFrame_3 = null;
              }
              alignFrame_3 = new AlignFrame( frame, Target, "s" );
              if (!alignFrame_3.ParseError2) {
                alignFrame_3.setVisible( true );
                hashData.setEnabled(false);
                selectTarget.setEnabled(false);
                bestMatch.setEnabled(true);
                subSearch.setEnabled(true);
              }
            }
        }
        else if ( item == about ) {
            helpDialog = new HelpDialog( frame, "help/About_demo.help" );
            helpDialog.setVisible( true );
        }
      }
    }
}

/**
 * @(#)Matrix3D.java        99/12/01
```

```
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */


/**
 * A fairly conventional 3D matrix object that can transform sets of
 * 3D points and perform a variety of manipulations on the transform
 */
class Matrix3D {
  float xx, xy, xz, xo;
  float yx, yy, yz, yo;
  float zx, zy, zz, zo;
  static final double pi = 3.14159265;
  /**
   * Create a new unit matrix
   */
  Matrix3D () {
    xx = 1.0f;
    yy = 1.0f;
    zz = 1.0f;
  }
  /**
   * Scale by f in all dimensions
   */
  void scale(float f) {
    xx *= f;
    xy *= f;
    xz *= f;
    xo *= f;
    yx *= f;
    yy *= f;
    yz *= f;
```

```
    yo *= f;
    zx *= f;
    zy *= f;
    zz *= f;
    zo *= f;
}
/**
 * Scale along each axis independently
 */
void scale(float xf, float yf, float zf) {
    xx *= xf;
    xy *= xf;
    xz *= xf;
    xo *= xf;
    yx *= yf;
    yy *= yf;
    yz *= yf;
    yo *= yf;
    zx *= zf;
    zy *= zf;
    zz *= zf;
    zo *= zf;
}
/**
 * Translate the origin
 */
void translate(float x, float y, float z) {
    xo += x;
    yo += y;
    zo += z;
}


/**
 * Rotate theta degrees about the y axis
 */
void yrot(double theta) {
    theta *= (pi / 180);
    double ct = Math.cos(theta);
    double st = Math.sin(theta);

    float Nxx = (float) (xx * ct + zx * st);
    float Nxy = (float) (xy * ct + zy * st);
    float Nxz = (float) (xz * ct + zz * st);
    float Nxo = (float) (xo * ct + zo * st);
```

```
    float Nzx = (float) (zx * ct - xx * st);
    float Nzy = (float) (zy * ct - xy * st);
    float Nzz = (float) (zz * ct - xz * st);
    float Nzo = (float) (zo * ct - xo * st);

    xo = Nxo;
    xx = Nxx;
    xy = Nxy;
    xz = Nxz;
    zo = Nzo;
    zx = Nzx;
    zy = Nzy;
    zz = Nzz;
  }
  /**
   * Rotate theta degrees about the x axis
   */
  void xrot(double theta) {
    theta *= (pi / 180);
    double ct = Math.cos(theta);
    double st = Math.sin(theta);

    float Nyx = (float) (yx * ct + zx * st);
    float Nyy = (float) (yy * ct + zy * st);
    float Nyz = (float) (yz * ct + zz * st);
    float Nyo = (float) (yo * ct + zo * st);

    float Nzx = (float) (zx * ct - yx * st);
    float Nzy = (float) (zy * ct - yy * st);
    float Nzz = (float) (zz * ct - yz * st);
    float Nzo = (float) (zo * ct - yo * st);

    yo = Nyo;
    yx = Nyx;
    yy = Nyy;
    yz = Nyz;
    zo = Nzo;
    zx = Nzx;
    zy = Nzy;
    zz = Nzz;
  }
  /**
   * Rotate theta degrees about the z axis
   */
  void zrot(double theta) {
    theta *= (pi / 180);
```

```
    double ct = Math.cos(theta);
    double st = Math.sin(theta);

    float Nyx = (float) (yx * ct + xx * st);
    float Nyy = (float) (yy * ct + xy * st);
    float Nyz = (float) (yz * ct + xz * st);
    float Nyo = (float) (yo * ct + xo * st);

    float Nxx = (float) (xx * ct - yx * st);
    float Nxy = (float) (xy * ct - yy * st);
    float Nxz = (float) (xz * ct - yz * st);
    float Nxo = (float) (xo * ct - yo * st);

    yo = Nyo;
    yx = Nyx;
    yy = Nyy;
    yz = Nyz;
    xo = Nxo;
    xx = Nxx;
    xy = Nxy;
    xz = Nxz;
  }
  /**
   * Multiply this matrix by a second: M = M*R
   */
  void mult(Matrix3D rhs) {
    float lxx = xx * rhs.xx + yx * rhs.xy + zx * rhs.xz;
    float lxy = xy * rhs.xx + yy * rhs.xy + zy * rhs.xz;
    float lxz = xz * rhs.xx + yz * rhs.xy + zz * rhs.xz;
    float lxo = xo * rhs.xx + yo * rhs.xy + zo * rhs.xz + rhs.xo;

    float lyx = xx * rhs.yx + yx * rhs.yy + zx * rhs.yz;
    float lyy = xy * rhs.yx + yy * rhs.yy + zy * rhs.yz;
    float lyz = xz * rhs.yx + yz * rhs.yy + zz * rhs.yz;
    float lyo = xo * rhs.yx + yo * rhs.yy + zo * rhs.yz + rhs.yo;

    float lzx = xx * rhs.zx + yx * rhs.zy + zx * rhs.zz;
    float lzy = xy * rhs.zx + yy * rhs.zy + zy * rhs.zz;
    float lzz = xz * rhs.zx + yz * rhs.zy + zz * rhs.zz;
    float lzo = xo * rhs.zx + yo * rhs.zy + zo * rhs.zz + rhs.zo;

    xx = lxx;
    xy = lxy;
    xz = lxz;
    xo = lxo;
```

```
    yx = lyx;
    yy = lyy;
    yz = lyz;
    yo = lyo;

    zx = lzx;
    zy = lzy;
    zz = lzz;
    zo = lzo;
  }

  /**
   * Reinitialize to the unit matrix
   */
  void unit() {
    xo = 0;
    xx = 1;
    xy = 0;
    xz = 0;
    yo = 0;
    yx = 0;
    yy = 1;
    yz = 0;
    zo = 0;
    zx = 0;
    zy = 0;
    zz = 1;
  }

  /**
   * Transform nvert points from v into tv.  v contains the input
   * coordinates in floating point.  Three successive entries in
   * the array constitute a point.  tv ends up holding the transformed
   * points as integers; three successive entries per point  */
  void transform(float v[], int tv[], int nvert) {
    float lxx = xx, lxy = xy, lxz = xz, lxo = xo;
    float lyx = yx, lyy = yy, lyz = yz, lyo = yo;
    float lzx = zx, lzy = zy, lzz = zz, lzo = zo;
    for (int i = nvert * 3; (i -= 3) >= 0;) {
      float x = v[i];
      float y = v[i + 1];
      float z = v[i + 2];
      tv[i    ] = (int) (x * lxx + y * lxy + z * lxz + lxo);
      tv[i + 1] = (int) (x * lyx + y * lyy + z * lyz + lyo);
      tv[i + 2] = (int) (x * lzx + y * lzy + z * lzz + lzo);
    }
```

```
    }
    public String toString() {
       return ("[" + xo + "," + xx + "," + xy + "," + xz + ";"
             + yo + "," + yx + "," + yy + "," + yz + ";"
             + zo + "," + zx + "," + zy + "," + zz + "]");
    }
}

/**
 * @(#)MotifFrame.java       99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.io.*;
import java.net.URL;
import java.util.Hashtable;
import java.util.Vector;
import java.awt.image.IndexColorModel;
import java.awt.image.ColorModel;
import java.awt.image.MemoryImageSource;
import java.awt.event.*;

/**
 * Generate a pop-out window when displaying the discovered motifs
 */
public class MotifFrame extends Frame {
   GhFrame frame;
   Choice SelectMotif = new Choice();
   String [] MotifText;
   boolean MotifTextError = false;
   boolean MotifTextRead = false;
   Panel image;
```

```
SinglePanel sp;
TextArea MotifTextArea;
Button align = new Button("Align");
Button cancel = new Button("Cancel");
int MotifType = 6;
int width = 400, height = 250;
XYZChemModel motif;
int MotifNum;
int an, rn;
float [] vt;
int [] rl;
String AtomNames[];
Vector motifs = new Vector();
boolean found = true, ParseError = false;
MotifFrameListener mfl;

public MotifFrame( GhFrame f, String motifname ) {
  super("[" + motifname + "]");

  frame = f;
  MotifExtractor( frame, motifname );
  do {
    try {
       Thread.currentThread().sleep(250);
    } catch(Exception e) {}
  } while ( !MotifTextRead );

  if ( found ) {
    MotifParser( frame, motifname );
    if ( !ParseError ) {
      for ( int i = 0; i < MotifNum; i++)
         SelectMotif.add( new Integer(i).toString() );
      Panel choicePanel = new Panel();
      add( choicePanel, "West" );
      choicePanel.add( SelectMotif );
      SelectMotif.select( 0 );
      Panel main = new Panel();
      add( main, "Center" );
      main.setLayout( new GridLayout(1, 2) );
      Panel text = new Panel();
      image = new Panel();
      main.add( text );
      main.add( image );
      text.setLayout( new BorderLayout() );
      text.add( MotifTextArea = new TextArea( MotifText[0], 10, 40 ), "Center" );
      MotifTextArea.setEditable(false);
```

```
        image.setLayout( new BorderLayout() );
        XYZChemModel temp = (XYZChemModel)motifs.elementAt(0);
        sp = new SinglePanel( this, temp, MotifType );
        image.add(sp, "Center");
        Panel control = new Panel();
        image.add(control, "South");
        control.setLayout(new FlowLayout());
        control.add(align);
        control.add(cancel);
        mfl = new MotifFrameListener( frame, this );
        SelectMotif.addItemListener( mfl );
        image.addContainerListener( mfl );
        align.addActionListener( mfl );
        cancel.addActionListener( mfl );
        repaint();
      }
    }
}
public void addNotify() {
    super.addNotify();

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension scsz = tk.getScreenSize();
    Insets insets = getInsets();
    setBounds(scsz.width/2 - width/2, scsz.height/2 - height/2,
            width + insets.left + insets.right,
            height + insets.top + insets.bottom);
}

/**
 * Extract each motif information from the output of pdiscover
 */
public void MotifExtractor(GhFrame frame, String mn) {

  int BufPtr;
  String Result;
  int i=0, j;
  char c;
  InputStream is = null;
  int BufSize = 1024;
  char [] Buf = new char[BufSize];
  int lastIndex;
  SimpleDialog exceptionDialog;

  try {
    Thread.sleep(20);
```

```
} catch(InterruptedException ie) {}

BufPtr = 0;
Result = "";
try {
  //Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
  is = new FileInputStream( mn );
  while ( (i = (int)is.read()) != -1 ) {
    c = (char)i;
    Buf[BufPtr++] = c;
    if ( BufPtr >= BufSize) {
      Result += new String(Buf, 0, BufSize);
      BufPtr = 0;
    }
  }
  Result += new String(Buf, 0, BufPtr);
} catch(Exception e) {
  Result += e.toString();
  MotifTextError = true;
  found = false;
  exceptionDialog = new SimpleDialog( frame, "Motif File Error",
                                      Result );
  exceptionDialog.setVisible(true);
}
try {
  if ( is != null)
    is.close();
} catch(Exception e) {
  Result += e.toString();
  MotifTextError = true;
  found = false;
  exceptionDialog = new SimpleDialog( frame, "Motif File Error",
                                      Result );
  exceptionDialog.setVisible(true);
}
if (!MotifTextError) {
  MotifTextRead = true;
  MotifNum = new Integer( Result.substring(0,1) ).intValue();
  if ( MotifNum == 0 ) {
        found = false;
        exceptionDialog = new SimpleDialog(frame, "Motif Not Found",
                                      "No qualified pattern is found!");
        exceptionDialog.setVisible(true);
  }
  else {
    MotifText = new String[MotifNum];
```

```
      int loc = Result.indexOf( "Motif", 0 );
      for ( j = 0; j < (MotifNum-1); j++) {
        MotifText[j] = "";
        lastIndex = Result.indexOf( "Motif", loc+5 );
        BufPtr = 0;
        for ( int k = loc; k < lastIndex; k++)  {
          Buf[BufPtr++] = Result.charAt(k);
          if ( BufPtr >= BufSize ) {
            MotifText[j] += new String(Buf, 0, BufSize);
            BufPtr = 0;
          }
        }
        MotifText[j] += new String(Buf, 0, BufPtr);
        loc = lastIndex;
      }
      MotifText[j] = "";
      BufPtr = 0;
      for ( int k = loc; k < (Result.length() - 1); k++) {
        Buf[BufPtr++] = Result.charAt(k);
        if ( BufPtr >= BufSize ) {
          MotifText[j] += new String(Buf, 0, BufSize);
          BufPtr = 0;
        }
      }
      MotifText[j] += new String(Buf, 0, BufPtr);
    }
  }
}

/**
 * Parse the format with the name of motif
 */
public void MotifParser(GhFrame frame, String mn)  {
  InputStream is = null;
  StreamTokenizer st = null;
  SimpleDialog exceptionDialog;

  try
  {
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
    is = new FileInputStream( mn );
    st = new StreamTokenizer(new BufferedReader(new InputStreamReader(is)));
    st.eolIsSignificant(true);
    st.commentChar('#');
      scan:
     while (true)
```

```
{
  switch ( st.nextToken() )
  {
    case StreamTokenizer.TT_EOF:
      break scan;
    default:
      break;
    case StreamTokenizer.TT_NUMBER:
      float x, y, z;
      int source, target;
      String name;
      MotifNum = (int)st.nval;
      if (MotifNum == 0) {
        SimpleDialog simpleDialog =
          new SimpleDialog(frame, "Failure to find a motif",
                "No qualified pattern can be found" );
        ParseError = true;
        return;
      }
      st.nextToken();
      st.nextToken();
      for ( int j=0;j<MotifNum;j++) {
      if ( st.nextToken() == StreamTokenizer.TT_WORD) {
        if (st.sval.equals("Motif")) {
          if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
              st.nextToken();
              st.nextToken();
            if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
              an = (int)st.nval;
              AtomNames = new String[an];
              vt = new float[an*3];
              if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                rn = (int)st.nval;
                rl = new int[rn*2];
                 st.nextToken();
                 st.nextToken();
                for (int i=0;i<(an*3);i=i+3) {
                  if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                    x = (float)st.nval;
                    if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                      y = (float)st.nval;
                      if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                        z = (float)st.nval;
                        if(st.nextToken() == StreamTokenizer.TT_WORD) {
                          name = st.sval;
                          AtomNames[i/3] = name;
```

```
                        vt[i] = x;
                        vt[i+1] = y;
                        vt[i+2] = z;
                        st.nextToken();
                      }
                    }
                  }
                }
              }
              for (int i=0;i<(rn*2);i=i+2) {
                if (st.nextToken() ==
                  StreamTokenizer.TT_NUMBER) {
                  source = (int)st.nval-1;
                  if (st.nextToken() ==
                   StreamTokenizer.TT_NUMBER) {
                    target = (int)st.nval-1;
                    rl[i] = source;
                    rl[i+1] = target;
                    st.nextToken();
                   }
                  }
              }
              st.nextToken();
            }
          }
        }
      }
    }

    motif = new XYZChemModel(an, rn, vt, rl, AtomNames);
    motifs.addElement(motif);
    }

     while( st.ttype != StreamTokenizer.TT_EOL &&
        st.ttype != StreamTokenizer.TT_EOF )
          st.nextToken();

  } // end Switch

 } // end while

 is.close();

} // end Try
catch( IOException e) {
 ParseError = true;
```

```
      exceptionDialog = new SimpleDialog( frame,
                  "File Format Error", e.toString() );
      exceptionDialog.setVisible(true);
    }

    //if (st.ttype != StreamTokenizer.TT_EOF)
     // throw new Exception(st.toString());
  } // end MotifParser()

  class MotifFrameListener implements
  ContainerListener, ItemListener, ActionListener {
    GhFrame frame;
    MotifFrame mf;
    int i;

    public MotifFrameListener( GhFrame f, MotifFrame mf ) {
      frame = f;
      this.mf = mf;
    }
    public void itemStateChanged( ItemEvent ie ) {
      i = Integer.parseInt( SelectMotif.getSelectedItem() );
      MotifTextArea.setText( MotifText[i] );
      MotifTextArea.setEditable(false);
      image.remove(sp);
      sp = null;
      XYZChemModel temp = (XYZChemModel)motifs.elementAt(i);
      sp = new SinglePanel(mf, temp, MotifType);
      image.add(sp, "Center");

      if (frame.alignFrame_1 != null) {
         frame.alignFrame_1.setVisible(false);
         frame.alignFrame_1 = null;
      }
    }
    public void componentRemoved( ContainerEvent ce ) {
        Component sp = ce.getChild();
        //System.out.println("motif removed");
    }
    public void componentAdded( ContainerEvent ce ) {
        Component sp = ce.getChild();
    }
    public void actionPerformed( ActionEvent ae ) {
      Button button = (Button)ae.getSource();

      if ( button == cancel ) {
        if ( frame.alignFrame_1 != null ) {
```

```
            frame.alignFrame_1.setVisible( false );
            frame.alignFrame_1 = null;
          }
          if ( frame.motifFrame != null ) {
            frame.motifFrame.setVisible( false );
            frame.motifFrame = null;
          }
          frame.select.setEnabled(false);
          frame.run.setEnabled(true);
        }
        else {
          frame.alignDialog = new AlignDialog( frame,
                                    frame.SelectData, mf );
          frame.alignDialog.setVisible( true );
        }
      }
    }
  }
/**
 * @(#)NCIImagesFrame.java        99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.net.URL;
import java.util.Hashtable;
import java.util.Vector;
import java.awt.image.IndexColorModel;
import java.awt.image.ColorModel;
import java.awt.image.MemoryImageSource;
import java.awt.event.*;
```

```
/**
 * Generate a pop-out window when displaying
 * three different molecules
 */
class NCIImagesFrame extends Frame {
  GhFrame frame;
  Panel label;
  Label name;
  SinglePanel sp;
  protected Font NCIImagesFont;
  protected FontMetrics NCIImagesFontMetrics;
  int width = 200, height = 300;
  Button text1, text2, text3, cancel;
  NCIImagesFrameListener nciifl;

  public NCIImagesFrame( GhFrame f, String[] SelectData, int ImageType ) {
    super( "Select Molecules" );
    frame = f;
    Panel labels = new Panel();
    labels.setLayout(new GridLayout(1, f.NUM));
    for ( int i = 0; i < f.NUM; i++) {
      label = new Panel();
      label.setLayout( new FlowLayout() );
      NCIImagesFont = new Font("TimesRoman", Font.BOLD, 14);
      NCIImagesFontMetrics = getFontMetrics(NCIImagesFont);
      label.add( name = new Label( SelectData[i] ) );
      name.setFont(NCIImagesFont);
      NCIImagesFont = new Font("TimesRoman", Font.PLAIN, 14);
      NCIImagesFontMetrics = getFontMetrics(NCIImagesFont);
      if ( i == 0 ) {
        label.add( text1 = new Button( "Text" ) );
        text1.setFont(NCIImagesFont);
      }
      else if ( i == 1 ) {
        label.add( text2 = new Button( "Text" ) );
        text2.setFont(NCIImagesFont);
      }
      else {
        label.add( text3 = new Button( "Text" ) );
        text3.setFont(NCIImagesFont);
      }

      labels.add( label );
    }
    add( labels, "North" );
```

```
  Panel NCIpanel = new Panel();
  NCIpanel.setLayout(new GridLayout(1,f.NUM));
  add(NCIpanel, "Center");
  for ( int i = 0;i < f.NUM;i++) {
    if ( ImageType == f.ImagesType_1 ) {
      sp = new SinglePanel(f, this, "models/" + SelectData[i],
                           f.ImagesType_1);
      NCIpanel.add(sp);
    }
    else if ( ImageType == f.ImagesType_2 ) {
      sp = new SinglePanel(f, this, "models/" + SelectData[i],
                           f.ImagesType_2);
      NCIpanel.add(sp);
    }
  }
  Panel control = new Panel();
  add( control, "South" );
  control.setLayout( new FlowLayout() );
  control.add( cancel = new Button( "Cancel" ) );
  nciifl = new NCIImagesFrameListener( f, this, SelectData, ImageType );
  text1.addActionListener( nciifl );
  text2.addActionListener( nciifl );
  text3.addActionListener( nciifl );
  cancel.addActionListener( nciifl );
  addWindowListener(
    new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        setVisible( false ); } } );
  }
public void addNotify() {
    super.addNotify();
    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension scsz = tk.getScreenSize();
    Insets insets = getInsets();
    setBounds(scsz.width/2 - width*frame.NUM/2,
              scsz.height/2 - height/2,
              width*frame.NUM + insets.left + insets.right,
              height + insets.top + insets.bottom);
}
class NCIImagesFrameListener implements ActionListener {
  GhFrame frame;
  NCIImagesFrame nciif;
  String [] SelectData;
  int ImageType;

  public NCIImagesFrameListener( GhFrame f, NCIImagesFrame nf, String[] s, int t ) {
```

```
    frame = f;
    nciif = nf;
    SelectData = s;
    ImageType = t;
  }
  public void actionPerformed( ActionEvent ae ) {
    Button button = (Button)ae.getSource();
    NCITextFrame temp;

    if (button == cancel) {
      if ( ImageType == frame.ImagesType_1 ) {
        frame.select.setEnabled(true);
        frame.run.setEnabled(false);
        for ( int i = 0; i<frame.Text1.size();i++) {
          temp = (NCITextFrame)frame.Text1.elementAt(i);
          temp.setVisible(false);
          frame.Text1.removeElementAt(i);
        }
        if ( frame.alignFrame_1 != null ) {
          frame.alignFrame_1.setVisible(false);
          frame.alignFrame_1 = null;
        }
        if ( frame.motifFrame != null ) {
          frame.motifFrame.setVisible(false);
          frame.motifFrame = null;
        }
        frame.nciImagesFrame_1.setVisible( false );
        frame.nciImagesFrame_1 = null;
      }
      else if (ImageType == frame.ImagesType_2 ) {
        frame.hashData.setEnabled(true);
        frame.selectTarget.setEnabled(false);
        frame.bestMatch.setEnabled(false);
        frame.subSearch.setEnabled(false);
        for ( int i = 0; i<frame.Text2.size();i++) {
          temp = (NCITextFrame)frame.Text2.elementAt(i);
          temp.setVisible(false);
          frame.Text2.removeElementAt(i);
        }
        if ( frame.alignFrame_2 != null ) {
          frame.alignFrame_2.setVisible(false);
          frame.alignFrame_2 = null;
        }
        if ( frame.alignFrame_3 != null ) {
          frame.alignFrame_3.setVisible(false);
          frame.alignFrame_3 = null;
```

```
            }
            frame.nciImagesFrame_2.setVisible( false );
            frame.nciImagesFrame_2 = null;
          }
        }
        else {
          if ( button == text1 ) {
            text1.setEnabled(false);
            frame.nciTextFrame = new NCITextFrame( frame, nciif,
                                 SelectData[0], 0, ImageType );
          }
          else if ( button == text2 ) {
            text2.setEnabled( false );
            frame.nciTextFrame = new NCITextFrame( frame, nciif,
                                 SelectData[1], 1, ImageType );
          }
          else if ( button == text3 ) {
            text3.setEnabled( false );
            frame.nciTextFrame = new NCITextFrame( frame, nciif,
                                 SelectData[2], 2, ImageType );
          }
          frame.nciTextFrame.setVisible( true );
          if (ImageType == frame.ImagesType_1)
            frame.Text1.addElement( frame.nciTextFrame );
          else if (ImageType == frame.ImagesType_2)
            frame.Text2.addElement( frame.nciTextFrame );
        }
      }
    }
  }

/**
 * @(#)Relation.java        99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
```

```
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */

/**
 * A stand-alone class recording the endpoint atom's IDs
 * for a bond.
 */
class Relation {
  public int sourceid = -1;
  public int targetid = -1;
  public boolean draw = false;
  public Relation(int i, int j) {
    sourceid = i;
    targetid = j;
  }
  public void init() {
    draw = false;
  }
}

/**
 * @(#)SinglePanel.java     99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.io.*;
import java.net.URL;
import java.util.Hashtable;
import java.util.Vector;
import java.awt.image.IndexColorModel;
import java.awt.image.ColorModel;
```

```
import java.awt.image.MemoryImageSource;
import java.awt.event.*;
import java.awt.Image.*;

/**
 * Generate a panel containing the image of one
 * molecule and apply the panel to constructing
 * the window having the images of three molecules
 */
class SinglePanel extends Panel implements MouseMotionListener {
  GhFrame                frame;
  MotifFrame             mf;
  NCIImageFrame_1        ncif1;
  NCIImageFrame_2        ncif2;
  NCIImagesFrame         ncif;
  int                    FrameID;
  XYZChemModel           md;
  float                  xfac;
  int                    prevx, prevy;
  float                  scalefudge = 1.1f;
  Matrix3D               amat = new Matrix3D(), tmat = new Matrix3D();
  String                 message;
  Image                  ImageBuffer;
  Graphics               BackScreen;
  SimpleDialog           exceptionDialog = null;

  /**
   * Generate a panel containing the image of the motif
   */
  public SinglePanel(MotifFrame f, XYZChemModel m, int FID) {
    mf = f;
    FrameID = FID;
    scalefudge = 0.35f;
    md = m;
    Atom.setPanel(this);
    repaint();
    super.addMouseMotionListener(this);
  }
  /**
   * Generate a panel containing the image of the molecule
   * in the case of opening database
   */
  public SinglePanel(GhFrame gf, NCIImageFrame_1 f, String mdname, int FID) {
    frame = gf;
    ncif1 = f;
    FrameID = FID;
```

```
  scalefudge = 0.9f;
  Atom.setPanel( this );
  CreateModel( mdname );
  repaint();
  super.addMouseMotionListener(this);
}

/**
 * Generate a panel containing the image of the molecule
 * in the case of displaying the target molecule
 */
public SinglePanel(GhFrame gf, NCIImageFrame_2 f, String mdname, int FID) {
  frame = gf;
  ncif2 = f;
  FrameID = FID;
  scalefudge = 0.9f;
  Atom.setPanel( this );
  CreateModel( mdname );
  repaint();
  super.addMouseMotionListener(this);
}

/**
  * Generate a panel containing the image of one molecule
  * which is the base to construct the window containg
  * the images of three  molecules
  */

public SinglePanel(GhFrame gf, NCIImagesFrame f, String mdname, int FID) {
   frame = gf;
   ncif = f;
   FrameID = FID;
   Atom.setPanel( this );
   CreateModel( mdname );
   repaint();
   super.addMouseMotionListener(this);
 }
 private void CreateModel( String mdname ) {
   InputStream is = null;
   try {
     is = new FileInputStream( mdname );
     md = new XYZChemModel(is);
   } catch(Exception e) {
    message = e.toString();
    md = null;
    exceptionDialog = new SimpleDialog(frame, "File Format Error", message);
```

```
    }
    try {
      if (is !=null )
        is.close();
    } catch(Exception e) {
      message = e.toString();
      md = null;
      exceptionDialog = new SimpleDialog(frame, "File Format Error", message);
    }
  }
  public void SetupImages() {
    if (FrameID == frame.ImageType) {
      if (ncif1 != null) {
        ImageBuffer = createImage(ncif1.width, ncif1.height - 58);
        BackScreen = ImageBuffer.getGraphics();
        BackScreen.setColor(getBackground());
        BackScreen.fillRect(0,0,ncif1.width, ncif1.height - 58);
      }
      else {
        ImageBuffer = createImage(ncif2.width, ncif2.height - 58);
        BackScreen = ImageBuffer.getGraphics();
        BackScreen.setColor(getBackground());
        BackScreen.fillRect(0,0,ncif2.width, ncif2.height - 58);
      }
    }
    else if (FrameID == frame.ImagesType_1 || FrameID == frame.ImagesType_2 ) {
      ImageBuffer = createImage(ncif.width, ncif.height - 58);
      BackScreen = ImageBuffer.getGraphics();
      BackScreen.setColor(getBackground());
      BackScreen.fillRect(0,0,ncif.width,ncif.height - 58);
    }
    else if (FrameID == mf.MotifType) {
      ImageBuffer = createImage( (mf.width-64)/2,mf.height - 36 );
      BackScreen = ImageBuffer.getGraphics();
      BackScreen.setColor(getBackground());
      BackScreen.fillRect(0,0,(mf.width-64)/2, mf.height - 36);
    }
  }
  public void addNotify() {
    super.addNotify();
    if (FrameID == frame.ImageType ) {
      if (ncif1 != null)
        setSize(ncif1.width, ncif1.height - 58);
      else
        setSize(ncif2.width, ncif2.height - 58);
    }
```

```
  else if (FrameID == frame.ImagesType_1 || FrameID == frame.ImagesType_2 )
      setSize(ncif.width, ncif.height - 58);
  else if (FrameID == mf.MotifType)
     setSize( (mf.width - 64)/2, mf.height - 36 );

  md.findBB();
  float xw = md.xmax-md.xmin;
  float yw = md.ymax-md.ymin;
  float zw = md.zmax-md.zmin;
  if (yw>xw)
    xw = yw;
  if (zw>xw)
    xw = zw;
  float f1 = getSize().width/xw;
  float f2 = getSize().height/xw;
  xfac =0.7f*(f1<f2?f1:f2)*scalefudge;
}
public void paint(Graphics g) {
  SetupImages();
  md.mat.unit();
  md.mat.translate(
         -(md.xmin+md.xmax)/2,
         -(md.ymin+md.ymax)/2,
         -(md.zmin+md.zmax)/2);
  md.mat.mult(amat);
  md.mat.scale(xfac, -xfac, 16*xfac/getSize().width);
  md.mat.translate(getSize().width/2,getSize().height/2,8);
  md.transformed = false;
  md.paint(BackScreen);
  g.drawImage(ImageBuffer,0,0,this);
}
public void mouseMoved(MouseEvent e) {
  prevx = e.getX();
  prevy = e.getY();
}
public void mouseDragged(MouseEvent e) {
  tmat.unit();
  int y = e.getY();
  int x = e.getX();
  float xtheta = (prevy - y)*(360.0f/getSize().width);
  float ytheta = (x - prevx)*(360.0f/getSize().height);
  tmat.xrot(xtheta);
  tmat.yrot(ytheta);
  amat.mult(tmat);
  repaint();
  prevx = x;
```

```
    prevy = y;
  }
  public void update(Graphics g) {
      paint(g);
  }
}

/**
 * @(#)XYZChemModel.java        99/12/01
 *
 * Author       Xinhuan Zheng
 *
 * Department of Computer and Information Science
 * New Jersey Institute of Technology
 * University Heights, Newark, 07102, USA
 * RA in the group of Jason T.L. Wang (NJIT)
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for any purpose and without
 * fee is hereby granted, provided that this copyright
 * notice appears in all copies.  Programmer(s) makes no
 * representations about the suitability of this
 * software for any purpose.  It is provided "as is" without
 * express or implied warranty.
 */
import java.awt.*;
import java.io.*;
import java.net.URL;
import java.util.Hashtable;
import java.util.Vector;
import java.awt.image.IndexColorModel;
import java.awt.image.ColorModel;
import java.awt.image.MemoryImageSource;
import java.awt.event.*;
import java.util.*;
import java.awt.*;
import java.awt.Image.*;
import java.awt.event.*;

/**
 * A stand-alone class generating the model of
 * one molecule
 */
public class XYZChemModel {
   int AtomNum, BondNum;
   float vert[];
```

```
Atom atoms[];
int tvert[];
int ZsortMap[];
int nvert, maxvert;
public Vector rs = new Vector();
static Hashtable atomTable = new Hashtable();
static Atom defaultAtom;
static
{
        atomTable.put("c", new Atom(0, 0, 0));
        atomTable.put("h", new Atom(210, 210, 210));
        atomTable.put("n", new Atom(0, 0, 255));
        atomTable.put("o", new Atom(255, 0, 0));
        atomTable.put("p", new Atom(255, 0, 255));
        atomTable.put("s", new Atom(255, 255, 0));
        atomTable.put("hn", new Atom(150, 255, 150)); /* !!*/
        atomTable.put("br", new Atom(0, 255, 255));
        atomTable.put("cl", new Atom(0, 255, 0));
        defaultAtom = new Atom(255, 100, 200);
}
boolean transformed;
Matrix3D mat;
float xmin, xmax, ymin, ymax, zmin, zmax;

XYZChemModel () {
        mat = new Matrix3D();
        mat.xrot(20);
        mat.yrot(30);
        rs = new Vector();
}

XYZChemModel (int an, int rn, float vt[], int rl[], String names[]) {
    this();
    AtomNum = an;
    BondNum = rn;
    nvert = an;
    vert = new float[AtomNum*3];
    atoms = new Atom[AtomNum];
    for (int i=0;i<(AtomNum*3);i++)
      vert[i] = vt[i];
    for (int i=0;i<AtomNum;i++) {
      Atom a = (Atom)atomTable.get(names[i].toLowerCase());
      if (a == null)
          a = defaultAtom;
      atoms[i] = a;
    }
```

```
      for (int i=0;i<BondNum*2;i=i+2)
        addRelation(rl[i], rl[i+1]);
}

/** Create a Cehmical model by parsing an input stream */
   XYZChemModel (InputStream is) throws Exception
   {
     this();
     StreamTokenizer st;
     st = new StreamTokenizer(new BufferedReader(new InputStreamReader(is)));
     st.eolIsSignificant(true);
     st.commentChar('#');
     int slot = 0;

     try
     {
           scan:
       while (true) {
         switch ( st.nextToken() )
         case StreamTokenizer.TT_EOF:
             break scan;
         default:
           break;
         case StreamTokenizer.TT_NUMBER:
           AtomNum = (int)st.nval;
           vert = new float[AtomNum*3];
           atoms = new Atom[AtomNum];
           float x = 0, y = 0, z = 0;
           String name;
           if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
             BondNum = (int)st.nval;
             for (int i=0;i<=AtomNum;i++) {
              if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                x = (float)st.nval;
                if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                   y = (float)st.nval;
                  if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                      z = (float)st.nval;
                    if (st.nextToken() == StreamTokenizer.TT_WORD) {
                      name = st.sval;
                      addVert(name, x, y, z);

                      while(st.ttype != StreamTokenizer.TT_EOL &&
                            st.ttype != StreamTokenizer.TT_EOF)
                        st.nextToken();
                    }
```

```
                    }
                  }
                }
              }
            }
              for (int i=0;i<=BondNum;i++) {
                int source, target;
                if (st.nextToken() == StreamTokenizer.TT_NUMBER) {
                  source = (int)st.nval-1;
                  if (st.nextToken() ==
                        StreamTokenizer.TT_NUMBER) {
                    target = (int)st.nval-1;
                    addRelation(source, target);
                    while(st.ttype != StreamTokenizer.TT_EOL &&
                      st.ttype != StreamTokenizer.TT_EOF)
                      st.nextToken();

                  }
                }
              }
            while( st.ttype != StreamTokenizer.TT_EOL &&
              st.ttype != StreamTokenizer.TT_EOF )
                st.nextToken();
        } // end Switch
      } // end while
      is.close();
    } // end Try
    catch( IOException e) { }
    if (st.ttype != StreamTokenizer.TT_EOF)
      throw new Exception(st.toString());
  } // end XYZChemModel()

void addRelation(int source , int target
    rs.addElement(new Relation(source,target));
}

/** Add a vertex to this model */
int addVert(String name, float x, float y, float z) {
    int i = nvert;
    Atom a = (Atom) atomTable.get(name.toLowerCase());
      if (a == null)
        a = defaultAtom;
        atoms[i] = a;
        i *= 3;
        vert[i] = x;
        vert[i + 1] = y;
```

```
            vert[i + 2] = z;
            return nvert++;
    }

/** Transform all the points in this model */
void transform() {
            if (transformed || nvert <= 0)
                return;
            if (tvert == null || tvert.length < nvert * 3)
                tvert = new int[nvert * 3];
            mat.transform(vert, tvert, nvert);
            transformed = true;
    }

    /** Paint this model to a graphics context.  It uses the matrix associated
        with this model to map from model space to screen space.
     */
    void paint(Graphics g)
            if (vert == null || nvert <= 0)
                return;
            transform();
            int v[] = tvert;
            int zs[] = ZsortMap;
            if (zs == null)
            {
                ZsortMap = zs = new int[nvert];
                for (int i = nvert; --i >= 0;)
                        zs[i] = i * 3;
            }

        /**
         * I use a bubble sort since from one iteration to the next, the sort
         * order is pretty stable, so I just use what I had last time as a
         * "guess" of the sorted order.  With luck, this reduces O(N log N)
         * to O(N)
         */
            for (int i = nvert - 1; --i >= 0;) {
                boolean flipped = false;
                for (int j = 0; j <= i; j++) {
                        int a = zs[j];
                        int b = zs[j + 1];
                        if (v[a + 2] > v[b + 2]) {
                            zs[j + 1] = a;
                            zs[j] = b;
                            flipped = true;
                        }
```

```
            }
            if (!flipped)
               break;
         }

         int lg = 0;
         int lim = nvert;
         Atom ls[] = atoms;
         if (lim <= 0 || nvert <= 0)
            return;
         int mm = 0 ;
         int vsize = rs.size();
         for (int i = 0; i < lim; i++,mm++)  {
            int j = zs[i];
            int grey = v[j + 2];
            if (grey < 0)
                     grey = 0;
            if (grey > 15)
                     grey = 15;
            Relation r = null;
            for(int k = 0 ; k < vsize ; k++) {
              r = (Relation)rs.elementAt(k);
              if(r.draw == false) {
                if(r.sourceid == (j/3))  {
                 atoms[j/3].paintline(g, v[j], v[j + 1], v[j+2],v[r.targetid*3],
                                          v[r.targetid*3 + 1],v[r.targetid*3+2]);
                 r.draw = true;
                 }
                 else if (r.targetid == (j/3)) {
                     atoms[j/3].paintline(g, v[j],
                               v[j + 1],v[j+2],v[r.sourceid*3],
                               v[r.sourceid*3 + 1],v[r.sourceid*3+2]);
                      r.draw = true;
                 }
              }
             }
            atoms[j/3].paint(g, v[j], v[j + 1], grey);
         }
         for(int k = 0 ; k < vsize ; k++) {
            ((Relation)rs.elementAt(k)).init();
         }
}

/** Find the bounding box of this model */
void findBB()
{
```

```
        if (nvert <= 0)
            return;
        float v[] = vert;
        float xmin = v[0], xmax = xmin;
        float ymin = v[1], ymax = ymin;
        float zmin = v[2], zmax = zmin;
        for (int i = nvert * 3; (i -= 3) > 0;)
        {
            float x = v[i];
            if (x < xmin)
                xmin = x;
            if (x > xmax)
                xmax = x;
            float y = v[i + 1];
            if (y < ymin)
                ymin = y;
            if (y > ymax)
                ymax = y;
            float z = v[i + 2];
            if (z < zmin)
                zmin = z;
            if (z > zmax)
                zmax = z;
        }
        this.xmax = xmax;
        this.xmin = xmin;
        this.ymax = ymax;
        this.ymin = ymin;
        this.zmax = zmax;
        this.zmin = zmin;
    }
}
```

REFERENCES

1. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 94–105, Seattle, Washington, 1998.

2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley Longman, Reading, Massachusetts, 1999.

3. L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 30–36, New York, New York, 1998.

4. S. Djoko, D. J. Cook, and L. B. Holder. An empirical study of domain knowledge and its benefits to substructure discovery. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):575–586, 1997.

5. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996.

6. C. Faloutsos. *Searching Multimedia Databases by Content.* Kluwer Academic Publishers, Norwell, Massachusetts, 1996.

7. C. Faloutsos and K.-I. Lin. *FastMap*: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995.

8. S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 73–84, Seattle, Washington, 1998.

9. J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the International Conference on Data Engineering*, pages 106–115, 1999.

10. M. G. Hicks and C. Jochum. Substructure search system. 1. Performance comparison of the MACCS, DARC, HTSS, CAS registry MVSSS, and S4 substructure search systems. *Journal of Chemical Information and Computer Sciences*, 30:191–199, 1990.

11. J. Kleinberg and A. Tomkins. Application of linear algebra in information retrieval and hypertext analysis. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 185–193, 1999.

12. The National Cancer Institute 3D structure database. `http://dtp.nci.nih.gov/docs/3d_database/dis3d.html`, January 10, 1999.

13. R. T. Ng, and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 144–155, Santiago, Chile, 1994.

14. Phylogenetics databases and information. `http://www.ucmp.berkeley.edu/subway/phylo/phylodat.html`, January 10, 1999.

15. The Protein Data Bank (PDB). `http://www.rcsb.org/pdb/`, January 10, 1999.

16. The Structure Group at the National Center for Biotechnology Information. `http://www.ncbi.nlm.nih.gov/Structure`, January 10, 1999.

17. J. T. L. Wang, B. A. Shapiro, and D. Shasha. *Pattern Discovery in Biomolecular Data: Tools, Techniques and Applications*. Oxford University Press, New York, New York, 1999.

18. J. T. L. Wang, X. Wang, K-I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 307 - 311, San Diego, California, 1999.

19. J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):559–571, 1994.

20. X. Wang, J. T. L. Wang, D. Shasha, B. A. Shapiro, S. Dikshitulu, I. Rigoutsos, and K. Zhang. Automated discovery of active motifs in three dimensional molecules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 89–95, Newport Beach, California, 1997.

21. D. Collins. *Designing object-oriented user interfaces*, pages 89–90, Benjamin Cummings, Redwood City, California, 1995.

22. D. Hearn. *Computer graphics*, pages 221–265, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.