MineSet™
User's Guide

Document Number 007-3214-002

CONTRIBUTORS

Written by Dieter Rathjens
Illustrated by Dany Galgani
Edited by Cindy Kleinfeld and Christina Cary
Production by Heather Hermstad
Engineering contributions by Joel Tesler, John Hawkes, Arun Swami, Gerald Sang'udi, Eben Haber, Jim Kelly, Peter Rathmann, Mario Schkolnick, Ishantha Lokuge, Gautam Bhargava, Ronny Kohavi, and Barry Becker.
St Peter's Basilica image courtesy of ENEL SpA and InfoByte SpA. Disk Thrower image courtesy of Xavier Berenguer, Animatica.

MineSet™ User's Guide
Document Number 007-3214-002

Contents

List of Figures

List of Tables

About This Guide

The *MineSet User's Guide* describes the features and capabilities of this suite of three database mining and five visualization tools. Current information about the MineSet product can be found on the World Wide Web at *http://www.sgi.com/Products/software/MineSet*.

Audience for This Guide

If you are using the Tool Manager to extract data from a database into the MineSet tools, you should understand database structures. It also would be helpful to know SQL.

If you are configuring the tools directly (through the configuration files, or through the command line in the case of the association rules), you should have some knowledge of UNIX as well as some programming experience.

Once the data has been loaded into the various visualization tools, you will not need a database or programming background, although you will be able to interpret the displays more easily if you have an understanding of the data and what it represents.

Structure of This Document

In addition to this preface, the documentation for MineSet consists of the following chapters:

Chapter 1, “Getting Started”
This provides a brief overview of each MineSet tool and describes the processes that occur when invoking and using a tool.

Chapter 2, “Setting Up MineSet”
This chapter describes how to set up MineSet by configuring the DataMover.

Chapter 3, “The Tool Manager”
This chapter describes the menus and functions of the initial interface for invoking tools and tells how to produce their respective configuration files.

Chapter 4, “Using the Tree Visualizer”
This chapter provides a complete description of the Tree Visualizer tool interface. This tool is valuable for visualizing hierarchical data.

Chapter 5, “Using the Map Visualizer”
This chapter provides a complete description of the Map Visualizer interface. This tool is valuable for visualizing data that is connected with a geographical location.

Chapter 6, “Using the Scatter Visualizer”
This chapter provides a complete description of the Scatter Visualizer interface. This tool is valuable for visualizing multidimensional data.

Chapter 7, “Using the Rules Visualizer”
This chapter provides a complete description of the Rules Visualizer. This tool is valuable for mining large datasets and visualizing correlations in that data.

Chapter 8, “MineSet Inducers and Classifiers”
This chapter provides a brief introduction to classifiers and the algorithms that generate them, called inducers. Specifically, it introduces the two MineSet classifiers: Decision Tree and Evidence.

Chapter 9, “Inducing and Visualizing the Decision Tree Classifier”
This chapter describes how to generate and use the Decision Tree Classifier. This tool is valuable for classifying data according to a set of attributes by making a series of decisions based on those attributes.

Chapter 10, “Inducing and Visualizing the Evidence Classifier”
This chapter describes how to generate and use the Evidence Classifier. This tool is valuable for classifying data by examining the probabilities of a specified result occurring based on a given attribute.

Chapter 11, “Column Importance”
This chapter provides a complete description of the column importance tool. It also describes the relationship between column importance and the importance ranking in the other data mining tools.

Appendix A, “Creating Data and Configuration Files for the Tree Visualizer”
This appendix explains the required formats of the Tree Visualizer data and configuration files.

Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer”
This appendix explains the required formats of the Map Visualizer data, configuration, hierarchy, and *.gfx* files.

Appendix C, “Creating Data and Configuration Files for the Scatter Visualizer”
This appendix explains the required formats of the Scatter Visualizer data and configuration files.

Appendix D, “Creating Data and Configuration Files for the Rules Visualizer”
This appendix explains the required formats of the Rules Visualizer data and configuration files.

Appendix E, “Command-Line Interface to MIndUtil: Classifiers, Discretization, Column Importance, and File Conversions”
This appendix describes the MIndUtil program and its options.

Appendix F, “Format of the Evidence Visualizer’s Data File”
This appendix describes the format of the Evidence Visualizer’s data file.

Appendix G, "Nulls in MineSet"
This appendix describes how MineSet supports nulls in the data access tools, the mining tools, and the visualization tools.

Appendix H, "Examples of Tool Usage"
This appendix provides demonstration scripts that guide you through some of the features of each of the MineSet tools.

Appendix I, "Further Reading and Acknowledgments"
This appendix lists reference sources for further reading about concepts and their implementations used in the MineSet tools. It also lists acknowledgments for data sources used in the examples provided with these tools.

Note: The hard copy of this documentation provides all screen shots and illustrations in black and white. The online version, however, provides these visuals in full, original color. Thus, if you are reading the hard copy version and find a particular graphic or screen shot difficult to see, go to the respective page of the online version for greater clarity.

Typographical Conventions

The following type conventions and symbols are used in this guide:

Italics — Executable names, filenames, program variables, tools, utilities, variable command-line arguments, and variables to be supplied by the user in examples, code, and syntax statements.

Bold — Keywords

`Fixed-width type` — On-screen text and prompts.

`Bold fixed-width type` — User input, including keyboard keys (printing and non-printing); literals supplied by the user in examples, code, and syntax statements.

[ ] — Syntax statement arguments surrounded by square brackets denote that these arguments are optional.

Chapter 1

Getting Started

This introduction provides an overview of MineSet™, an integrated suite of database mining and visualization tools, and describes the basic tool execution scenario.

Note: Before using any of the MineSet tools, follow the installation and licensing instructions in the MineSet release notes. Then your system administrator must set up the DataMover configuration file. You also can choose to set up some options. The setup details are described in Chapter 2.

MineSet Tools Suite

The MineSet suite tools let you mine and graphically display quantitative information in ways that can help you better visualize, explore, and understand your data. This suite of data mining and analysis tools can help you organize and examine your data in new and meaningful ways. The mining tools automatically find patterns and build models that can be viewed using the visualization tools. Also, the visualization tools can be applied directly to the data for more insights. These tools provide an enabling power that lets you gain a deeper, intuitive understanding of your data, and helps you discover hidden patterns and important trends.

These tools provide a highly interactive, three-dimensional (3D) visual interface that lets you manipulate visual objects on the screen, as well as perform animations. This ability to visualize and survey complex data patterns can prove invaluable as a decision support mechanism.

The MineSet suite consists of three basic components:

- **a centralized control module**, consisting of a graphical user interface tool called the Tool Manager, and a process called the DataMover, which runs on the server
- **database mining**, with four database mining tools:
  - Association Rules Generator
  - Decision Tree Inducer and Classifier
  - Evidence Inducer and Classifier
  - Column Importance
- **visualization tools**, of which there are five that let you view your data using different visual metaphors:
  - Tree Visualizer
  - Map Visualizer
  - Scatter Visualizer
  - Rules Visualizer
  - Evidence Visualizer

The following sections provide a brief description of each of the mentioned above components.

The Tool Manager

Each of the mining and visualization tools described below can be configured and started via a consistent graphical user interface known as the Tool Manager. The Tool Manager

- connects you to the server on which the database and mining tools reside
- lets you access, query, and manipulate data
- creates configuration files for each tool
- extracts data from the database to generate input files for each of the tools

The DataMover

The DataMover is a process that runs on the server on behalf of the user. The DataMover

- connects to the database or flat files, and retrieves the data
- invokes the mining tools
- performs additional data manipulation such as binning and aggregation
- returns the data to the Tool Manager for distribution to the visualization tools
- can store the data in files on the server or client for future operations.

The Association Rules Generator

The Association Rules Generator part of this tool processes an input file, then generates an output file consisting of rules. These rules indicate the frequency with which one item occurs in a record along with another item. The strength of the association is quantified by three numbers.

- The first number, the *predictability* of the rule, quantifies how often *X* and *Y* occur together as a fraction of the number of records in which *X* occurs. For example, given that someone has bought milk, how often do they also buy eggs.
- The second number, the *prevalence* of the rule, quantifies how often *X* and *Y* occur together in the file as a fraction of the total number of records. For example, how often were milk and eggs bought together.
- The third number is *expected predictability*. This gives an indication of what the predictability would be if there were no relationship between the items in the record. For example, how often were eggs bought, regardless of whether milk was bought as well.

The Decision Tree Inducer and Classifier

The Decision Tree Classifier classifies data according to a set of attributes by making a series of decisions based on those attributes. The process is similar to using a biological key to identify plants. Applying this classifier to determine the profile of someone with credit worthiness, for example, a decision tree might determine if someone who owns a home, owns a car that cost between $15,000 and $23,000, and has two children, is a good credit risk.

The Decision Tree Inducer generates a decision tree classifier from a "training set" (a set of data that the user has already classified). Then, the structure of the classifier's decision tree is displayed using the Tree Visualizer, with each decision being represented by a node of the tree. The graphical representation can help the user understand the classification algorithm, as well as provide valuable insights into the data. Finally, the classifier can be used to classify unclassified data.

The Evidence Inducer and Classifier

The Evidence Classifier classifies data by examining the probabilities of a specified result occurring based on a given attribute. For example, it might determine that someone who owns a car that cost between $15,000 and $23,000 has a 70% chance of being a good credit risk, and a 30% chance of being a bad credit risk. The classifier predicts the class with the highest probability based on a simple probabilistic model.

The classifier is first generated from a training set, similar to the decision tree classifier. The analysis of the data is displayed using the Evidence Visualizer, which shows pie charts illustrating the different probabilities. This graphical representation can help the user understand the classification algorithm, as well as providing valuable insights into the data and answering "what if" questions. Finally, the classifier can be used to classify unclassified data.

Column Importance

Column Importance determines how important various attributes are for determining the value of a given label attribute. For example, you can ask MineSet to select automatically the best three attributes that help determine whether someone is a good credit risk. The system might select income, own-house, and car-cost. These attributes then can be mapped to the axes of the Scatter Visualizer, or used in the hierarchy of the Tree Visualizer.

Column Importance has an advanced mode that provides additional capabilities. First, it lets you determine how important each of the attributes are. (For example, you could determine that both income and salary are similar in importance in determining credit risk. Although income might be slightly better in determining importance, perhaps you would prefer to use salary because it is easier to obtain.) Second, once you explicitly choose an attribute, you can determine what other attributes are important in conjunction with it. (For example, if you have chosen salary rather than income, house-cost might become more important than own-house, and income would have a very low importance.)

Tree Visualizer

The Tree Visualizer helps you analyze data that has hierarchical relationships. It provides an interactive “fly-through” capability for examining the relations between data at different hierarchical levels. For example, the Tree Visualizer can be used to examine a company’s product line, graphically displaying each product’s contribution to the company’s total revenue. Each branch of the hierarchy displays information at increasing levels of detail, breaking revenues down by product lines and, eventually, individual products. Another example of using the Tree Visualizer is to show company sales revenue, displaying a company-wide total as well as sub-totals at regional and other levels. The fly-through capability in the Tree Visualizer lets you rapidly reposition your view of the data. The Tree Visualizer’s filtering and searching capabilities let you focus on specific data elements and queries.

The Tree Visualizer is also used to view the results of the Decision Tree Classifier, with each decision being represented by a separate node in the tree. Each node also shows bars showing how the classifier classifies the data based on the decisions up to that point (for example, 73% of people who own a home and have two children are good credit risks, while 27% are not).

Map Visualizer

The Map Visualizer lets you visualize data relationships that exist across geographically meaningful areas. For example, you can visualize different areas of a country, showing the relative impact of a marketing program. The Map Visualizer's drill-down capabilities let you focus on designated regions and perform a more detailed analysis in smaller geographical elements. One application might be analyzing how one or more products are being sold across different geographies. A powerful animation feature, coupled with a capability to connect different views of the same or related data, permits fast comparisons and difference analyses. This tool lets you visually examine patterns in your data that are difficult to detect when that data is shown in a tabular, two-dimensional form.

Scatter Visualizer

The Scatter Visualizer lets you examine the behavior of data across different dimensions. The data is shown in a grid representing up to three dimensions. Extra dimensions can map to the size, color, and label of each displayed entity. Two further independent dimensions can be assigned as dynamic dimensions. A slider can be use to select specific values along those dimensions, or a path can be traced through those dimensions, for animation. During the path traversal, the display changes automatically to reflect the change in the independent variable.

Rules Visualizer

The Rules Visualizer visually represents the results of the Association Rules Generator mining tool. It provides detailed data analysis that lets you examine relationships across data elements in new ways. In doing so, you might discover relationships that significantly differ from what you might have expected; this, in turn, can lead to important discoveries about your data or the processes behind that data. This tool's visualization capabilities let you discover additional patterns of co-occurrence between these data elements. For example, you can use the analysis of products sold during the last sales promotion to guide your advertising campaign for the next sales period. The Rules Visualizer's high performance would let you analyze the results from today's sales data in time to alter the advertising campaign for the following day.

The Evidence Visualizer

The Evidence Visualizer visually represents the results of the Evidence Classifier. It initially shows pie charts that represent how the various attributes contribute to the decision. For example, it might show that owning a home contributes to being a good credit risk. By clicking on the pie charts, one can show the effect of combining various attributes has on the final result; for example: what happens in a household that rents, has one child, and drives a car valued between $8,000 and $12,000.

Basic Tool Execution Scenario

Each of the MineSet tools is started, configured, and run in a consistent manner. The sequence of actions you follow at your workstation and at the host server is shown schematically in Figure 1-1. A description of the steps inherent in this figure follows.

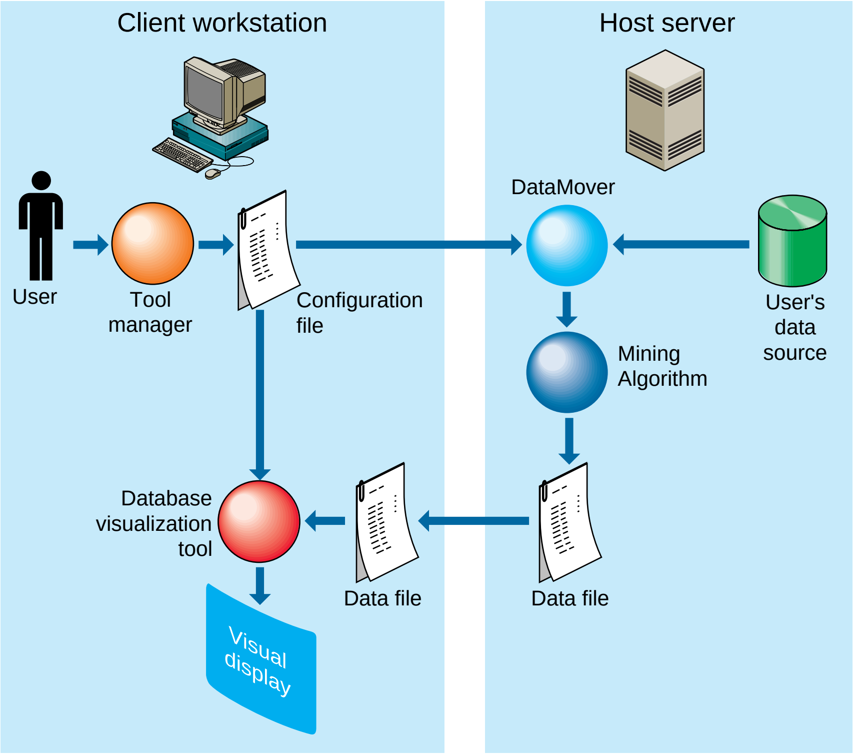


Figure 1-1 Tool Execution Sequence

Note: The following steps describe a "typical" interaction with a MineSet tool, and the sequence of the tool's actions. Depending on your requirements, some steps might be skipped (for instance, if the data and configuration files have been generated in a previous work session).

1. Start the Tool Manager, which is the graphical interface for generating and specifying the configuration file, data file, and tools to be used. The Tool Manager resides on your workstation.
2. The Tool Manager opens a network connection to the DataMover, which runs on the server.
3. Use the Tool Manager to specify
   - the database and table, or a flat file containing the data on either the client or the server
   - which mining tools, if any, are to be applied
   - the data file to be generated
   - what tool visualizes the data
   - how that data is to be displayed
   - an optional file on the client or server in which to save the results for future processing

   Information retrieved via the DataMover is used to guide this interaction. As a result, the Tool Manager generates a configuration file. This file contains the user-defined parameters that determine the execution of the following steps.
4. The Tool Manager transmits a copy of the configuration file from step 3 to the DataMover. The DataMover processes the file by
   - accessing the database or flat file
   - performing the specified data transformations
   - running the mining tools
   - generating the data file

   This data file consists of your data in a specific format readable by the MineSet tool. Then a copy of the data file is placed on your workstation.
5. The Tool Manager invokes the MineSet visualization tool you specified in step 3.
6. The tool accesses the data file and, based on the user-defined parameters entered in step 3, graphically displays the data.
7. If you generated a classifier, that classifier can be applied to additional data (see Figure 8-5).

Chapter 2

Setting Up MineSet

This chapter describes how to set up MineSet, which requires configuring the DataMover. The configuration has two parts:

- configuring the user's account on the server (optional), and
- a global configuration, which usually is done by the system administrator

The DataMover is a process that runs on the server, although it is not directly accessible to users. The DataMover provides access to databases and data stored in flat files, and transforms data for the mining and visualization tools.

Configuring the DataMover Server

In order to use the MineSet tools, two configuration files must be created on the server: one by you, the other by the system administrator.

The User Configuration File

Note: You must have a UNIX® account on every server you want to access.

The DataMover server creates files on the server machine on behalf of each user. The DataMover configuration file, *.datamove*, lets you control where these files are created and whether different classes of files are saved or discarded. This file is located on the server, in your home directory. A sample *.datamove* file is located on the server, in the */usr/lib/MineSet/datamove* directory.

If the *.datamove* file is absent, or if a particular entry is not present in the *.datamove* file, the DataMover uses a default value for that entry.

Each entry in the DataMover's configuration file must be on a separate line. For example:

```
file_cache = directory_name
temp_dir = directory_name
```

where *file_cache* specifies the location to which the DataMover stores its query specification and output data files. The location to which the DataMover stores intermediate files for mining processing is the directory specified after *temp_dir*. If either of the *file_cache* or *temp_dir* directories do not exist, the DataMover attempts to create them on its first invocation. The default *temp_dir* is */usr/tmp* and the default *file_cache* directory is *./mineset_dir/%U*. The *%U* is a wild card that is filled in with the user name on the client machine. This is useful in reducing contention if many users want to log in to a common account on the server. If multiple sessions were simultaneously connected to the same *file_cache* directory, they could overwrite each other's server files, causing incorrect and unexpected results. To prevent this, DataMover maintains a lock at the *file_cache* directory level. The second and later attempts to connect to a particular *file_cache* directory result in failure and an error message.

Once a query result has been returned to the client machine, the DataMover has the option to delete the query result and specification. The DataMover's behavior is controlled by the following options:

```
keep_temp_files
keep_data_files
keep_query_files
```

Each of these entries must be on a separate line. All entries default to *no*. Adding the entry

```
keep_query_files = yes
```

to the *.datamove* file instructs the DataMover not to delete query specification files when it is finished processing them.

Some of the files created by the DataMover can be large. Thus, if you specify one of the *keep_** options as yes, you should log on to the server periodically and clean out the file cache. The settings in the *.datamove* file can be overridden for each file with a Tool Manager option.

Using MineSet to create a classifier via the Tool Manager typically causes a classifier file and a classifier-options file to be created in the *file_cache* directory. You can delete or retain these files for further use by setting the following options in the *.datamove* file:

```
keep_classifier_files=yes
keep_classifier_options_files=no
```

As with other options in the *.datamove* files, these must also be on separate lines. It is worth noting that the predefined default for the *keep_classifier_files* option is “yes.”

Mandatory Configuration File

The MineSet DataMover server must be configured to find information in the databases. The DataMover works with Oracle® versions 7.2 or later, INFORMIX®, and Sybase®.

The DataMover server reads the */usr/lib/MineSet/datamove/dm_config* file during start up. This file is not created by *Inst* during installation. It must be created by the system administrator, who must log in as root to edit this file. It can be created via an editor such as jot, vi, or Emacs. An example file can be found in */usr/lib/MineSet/datamove/dm_config.sample*. The format of this file is

```
Oracle {
"ORACLE_SID", "ORACLE_HOME";
}
Informix {
"INFORMIXSERVER", "INFORMIXDIR";
}
Sybase {
"DSQUERY", "SYBASE";
}
```

Each entry is optional; describe those databases in use at your site. If your server is not running any databases, that is, you intended to use MineSet with ASCII files only, simply make an empty *dm_config* file or copy the sample.

The line "ORACLE_SID", "ORACLE_HOME" is filled in with the specific information and repeated once for each Oracle database to be accessed via the DataMover. ORACLE_SID and ORACLE_HOME are Oracle specific parameters defining an Oracle instance.

Each line in the INFORMIX section defines a database server that, in turn, can contain several databases. The server is interrogated at runtime to determine which databases it contains, so there is no need to record the individual databases in the *dm_config* file. The first entry is the INFORMIX server (corresponding to the *INFORMIXSERVER* environment variable), and the second is the INFORMIX directory (corresponding to the *INFORMIXDIR* environment variable).

Each entry in the Sybase section defines a database server (or, in Sybase terminology, an SQL Server™). The first entry is the Sybase SQL Server name (corresponding to the *DSQUERY* environment variable); the second is the Sybase home directory (corresponding to the *SYBASE* environment variable).

An example configuration file might be as follows:

```
Oracle {
"v73", "/usr/people/oracle/v73";
"wrhse", "/opt/oracle";
}
Informix {
"learn_online", "/u5/informix";
}
Sybase {
"MINESET", "/usr/sybase/10.0.2.4";
}
```

This configuration file lets the DataMover access two Oracle databases, one named “v73” (installed in */usr/people/oracle/v73*), and another named “wrhse” (installed in */opt/oracle*); an INFORMIX Server; and Sybase SQL Server. Each of the INFORMIX and Sybase servers can, in turn, contain multiple databases.

For Oracle and Sybase, DataMover uses vendor-supplied shared libraries as its connection to the databases. One of the purposes of the *dm_config* file is to specify where DataMover must look for its shared libraries. For Oracle, DataMover looks for the file *$ORACLE_HOME/lib/libclntsh.so*. The Oracle installation manual describes how to create this shared library, if it is not already present. For Sybase, DataMover looks in the *$SYBASE/lib/* directory for the following shared libraries: *libct.so*, *libcs.so*, *ibcomn.so*, *libintl.so*, *libtcl.so*, *libinsck.so*.

Using MineSet With Existing Data Files

Sometimes it is convenient to use MineSet with data that is already stored as a file, but requires further processing before it can be mined or visualized. In this case, the data file can be made available (with a modest effort) to the Tool Manager / DataMover.

First, the data file must be in a tab-delimited format, with the same number of fields in each line. A numeric or string field with a single “?” character appearing between delimiters is loaded as a Null value.

For a detailed discussion of null values, refer to Appendix G, “Nulls in MineSet.”

The contents of the data file must be described to Tool Manager / DataMover via a file with the *.schema* extension. The format of the *.schema* file is shown below:

```
#
#  A line beginning with a "#" is a comment
#
input {

#  The first line lists the data file which is described.  It
#  must be a simple filename, not a path.

     file "carmodels.data";

#  Fields are listed left to right in the line, legal
#  types are float, double, int, string, and  dataString
#  Be sure to end every line with a semicolon ";"

     float mpg;
     int cylinders;
     float cubicinches;
     int horsepower;
     int weightlbs;
     double timeaccelerate;
     int year;
     string origin;
     dataString model;
}
```

The schema and data files must be located in the same directory. If you prepare a dataset in this fashion on the client machine, it can be opened with the Tool Manager’s *Find File* dialog. If the file requires any additional processing, it is copied to the server. Sometimes this is not convenient, especially if the file already exists on the server, or is large. In this case, the *.schema* and *.data* files should be copied (or symbolically linked) into the your *file_cache* directory on the server. The directory used as the file cache is specified in your *.datamove* file; the default is *./mineset_dir/%U*.

Using MineSet to Connect to Remote Databases

Sometimes it may be not be feasible to install DataMover on the machine running the database server. In this situation, DataMover may be installed an intermediate server, and DataMover then can use the database vendor's networking facility to connect to the remote database. (This is sometimes referred to as a 3-tier architecture.)

Oracle

An Oracle installation is required on the intermediate server; this Oracle installation need not be running an active database, but is needed for access to the shared library, *libclntsh.so*, and the Oracle names file, *tnsnames.ora*.

On the intermediate server, there should be an entry for the local Oracle install, with `ORACLE_HOME` and `ORACLE_SID`, as usual. Add entries for any desired remote database to the *$ORACLE_HOME/network/admin/tnsnames.ora* file of the Oracle install on the intermediate server.

Then, when a user wishes to log in to user "system", password "manager" at database "remotedb", they should give the name of the intermediate server for the Tool Manager "Log on to server..." dialog and choose the intermediate server's Oracle database. When logging in to the database itself, they should give "system" for the database username, and "manager@remotedb" for the password. The added "@remotedb" tells Oracle to use SQL*Net™ to connect to the remote database, instead of using a local connection.

Operating across SQL*Net is substantially slower than a local connection, especially for queries which return a large amount of data. Hence it is better to install DataMover on the same machine as the Oracle server, if at all possible.

Sybase

A Sybase installation is required on the intermediate DataMover server; this Sybase installation need not be running an active database, but it is needed for access to the shared libraries and the *interfaces* file.

In order to access the Sybase SQL server running on the remote machine, the *interfaces* file on the DataMover server machine should have an entry for this Sybase SQL server. Please refer to your Sybase manuals for the procedure for creating such entries. Also, the name of this Sybase SQL server on the remote machine should be included in the *dm_config* file on the intermediate DataMover server machine.

Once this setup is done, access to the Sybase SQL server on the remote machine is handled transparently. The user can choose it and access data from it just like any other database source, using the panels from the Tool Manager.

Chapter 3

The Tool Manager

This chapter discusses the functions of the Tool Manager, which is the graphical user interface (GUI) that lets you specify data and configuration information for the MineSet tools in this package. It provides an overview of this interface, then describes every component of each panel that this interface displays for all MineSet tools.

Note: Any screens dedicated to a specific tool are discussed in the chapter for that tool; for example, the screen for specifying the Tree Visualizer's configuration file is discussed in Chapter 4, "Using the Tree Visualizer."

Overview of Tool Manager

The Tool Manager provides the initial GUI for most of your interactions with the MineSet tools. This GUI lets you start the individual tools and specify the following:

- the data you want to analyze
  - from a database
  - from a file
- the set of transformations used to get from the data you are capturing to the data that is displayed:
  - mining tools—finding patterns in data
  - binning variables—discretizing column values into groups, such as grouping years by decade
  - making arrays—taking the values of one column and turning them into an array indexed by discrete values in another column
  - distributing columns—making two or more new columns from a single column of values, distributed by the discrete values of another column

  - removing columns—excising unneeded columns to save space
  - adding new columns—functions of old columns
- how you want the data displayed on the screen; for instance,
  - as a hierarchy (Tree Visualizer)
  - as a map (Map Visualizer)
  - as relations of numerous independent variables (Scatter Visualizer)
  - as associated rules (Rules Visualizer)
  - as evidence (Evidence Visualizer)
- specific mappings of data values to visual elements on the screen, such as colors, bars, heights, and so forth
- non-data-related tool options, including
  - background colors
  - grid spacing
  - label sizes

Note: The Tool Manager generally does not support data files not created by the Tool Manager (without some manual work to make them compatible).

Starting the Tool Manager

You can run the Tool Manager in two modes:

- interactive mode—the Tool Manager provides windows, menus, buttons, and so on, to let you access, mine, and visualize your data. Interactive mode also lets you save a description of your actions to a "session file" for future use.
- batch mode—the Tool Manager performs all the actions described in the session file without bringing up windows. For example, batch mode is useful for lengthy computations that need to be done every night, so that the data can be fully prepared each morning.

There are three ways to start the Tool Manager in interactive mode:

- Double-click the MineSet icon, which is in the Applications or the MineSet page of the icon catalog. The Tool Manager starts with the same configuration used in the last Tool Manager session.
- Double-click an icon representing a configuration saved from a previous invocation of the Tool Manager. This starts the Tool Manager with that configuration file.
- Start the Tool Manager from the UNIX shell command line by entering this command at the prompt:

  ```
  mineset [ configFile ]
  ```

  Here, `configFile` is optional and specifies the name of the configuration file to use. If you do not specify a configuration file, MineSet starts up with the configuration most recently used.

To start the Tool Manager in batch mode, enter this command at the UNIX shell prompt:

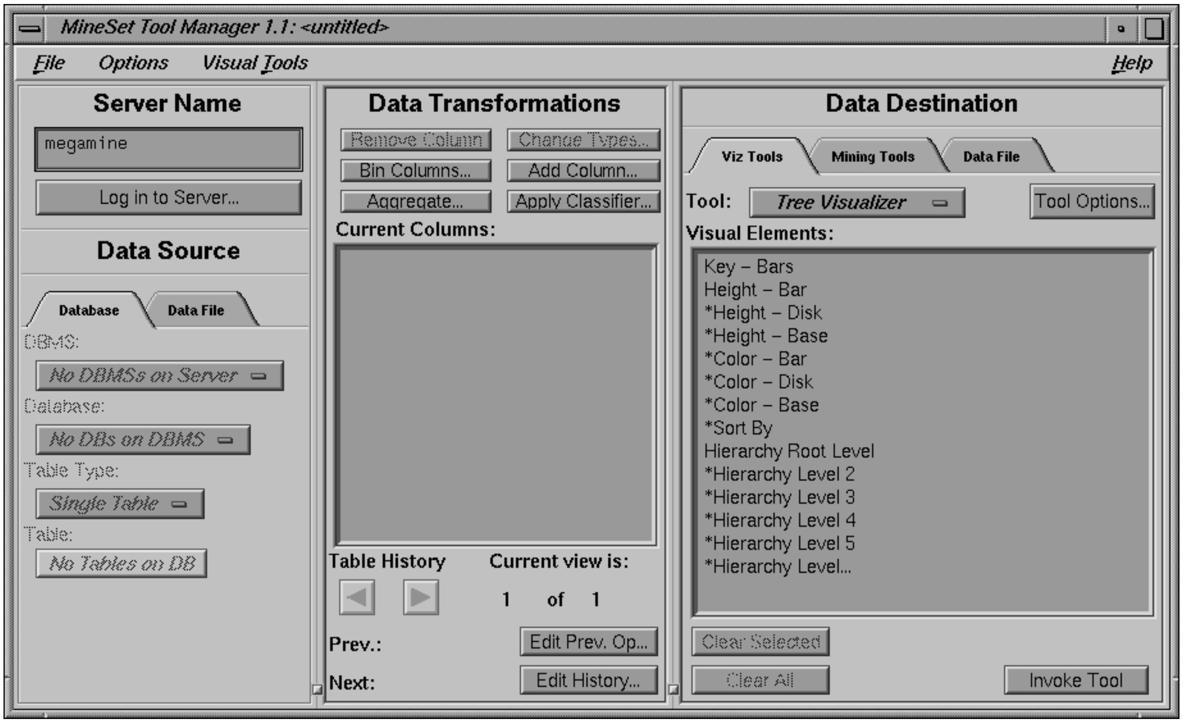
```
mineset_batch [-s serverPassword -d databasePassword]
configFile
```

The **-s** and **-d** options allow you to specify the password for logging into the server and database respectively. If you do not specify these options, *mineset_batch* will ask you to type in the passwords, thus these options are useful when running *mineset_batch* from a shell script. To specify that there is no password for either the server or database, use **-s** or **-d** followed by two double quotes, that is,

```
mineset_batch -s "" -d "" foo.mineset
```

If you specify one of the two passwords, you must specify both.

Figure 3-1 shows the Tool Manager's startup window.



MineSet Tool Manager 1.1: <untitled>

Server Name: megamine

DBMS: No DBMSs on Server

Database: No DBs on DBMS

Table Type: Single Table

Table: No Tables on DB

Current view is: 1 of 1

Tool: Tree Visualizer

Visual Elements:

Key – Bars
Height – Bar
*Height – Disk
*Height – Base
*Color – Bar
*Color – Disk
*Color – Base
*Sort By
Hierarchy Root Level
*Hierarchy Level 2
*Hierarchy Level 3
*Hierarchy Level 4
*Hierarchy Level 5
*Hierarchy Level...

Figure 3-1 The Tool Manager Startup Window

This window consists of three panels:

- The *Server Name* and *Data Source* panel provides a starting point for working with data. *Server Name* lets you specify the server where database and mining operations are to take place. *Data Source* establishes where the data comes from and what type it is.
- *Data Transformations* allows you to modify the data from your data source.
- *Data Destination* lets you create visualizations based on your data, save the data to a file, or mine the data for association rules, create classifiers based on the data, or find important columns in the data.

  Note: The latter screens are explained in the respective tool's chapter.

The following sections describe each panel of the main Tool Manager window.

Connecting to a Server and Choosing a Data Source

The Server Name and Data Source panel (Figure 3-2) lets you specify the name of the server on the network to which you want to connect and choose the data source.

At the top of the Data Source panel are two tabs:

- Database
- Data File

Click one of these, depending on whether you want to work with a database or a data file.

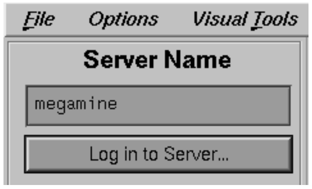


Figure 3-2 Server Name Panel

The first time you use the program, the *Server Name* text box is blank. Type in the name of the server you want to access. After this, the name of the last server accessed appears in the box by default.

You must connect to a server to get information from a database or mining tool, or to apply transformations to an existing data file. It is not necessary if you plan to use an existing client data file without transforming it.

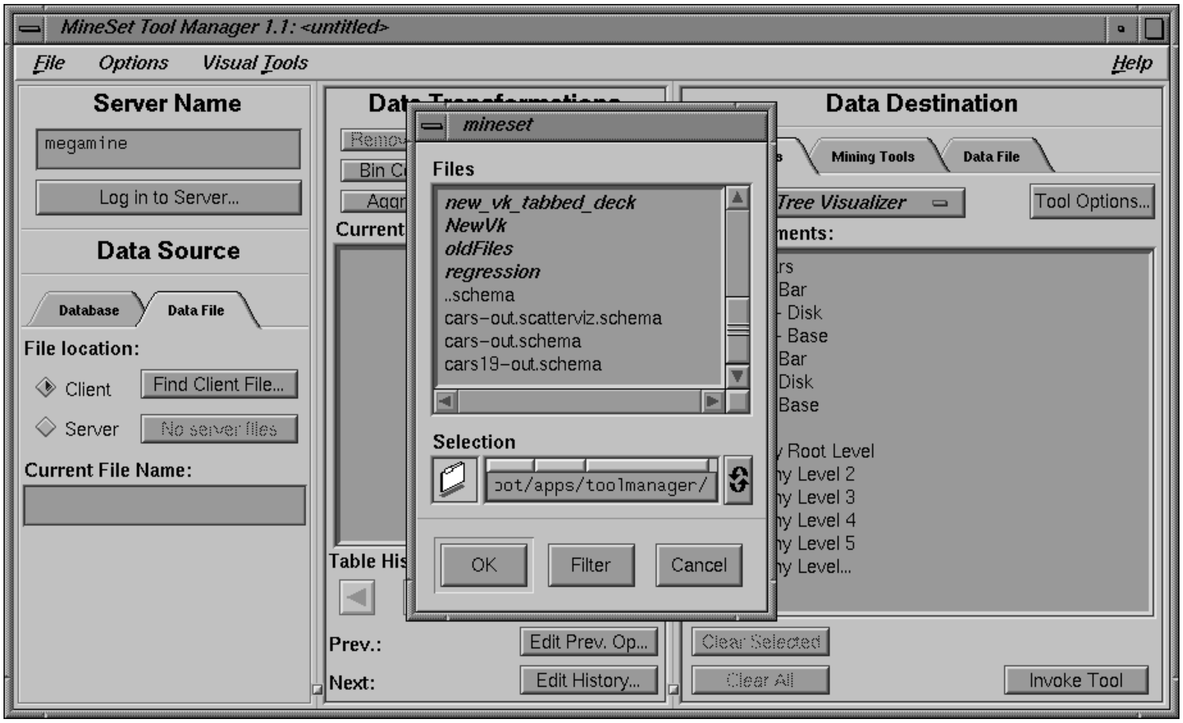
The *Log in to Server* button is the equivalent of pressing the Enter key on the keyboard when the cursor is in the *Server Name* box. After you have entered a server name in the text box, clicking *Log in to Server* (or pressing the *Enter* key) causes the Tool Manager to connect to that server.

When you have specified a server and clicked *Log in to Server* (or pressed the *Enter* key), a dialog box prompts you to type in the login name and password of your account on that server. When these are accepted, your workstation is connected to the server.

Choosing an Existing Data File

If you want to work with a data file previously created by the Tool Manager, select *Data File* from the tabbed list. The buttons in the Data Source panel change (see Figure 3-4), letting you specify the name of a *.schema* file (which describes some data file). To specify the name, choose either *Client* or *Server.*

- If you choose *Client*, click *Find Client File...* to bring up a file selection dialog (Figure 3-3).



Server Name: megamine

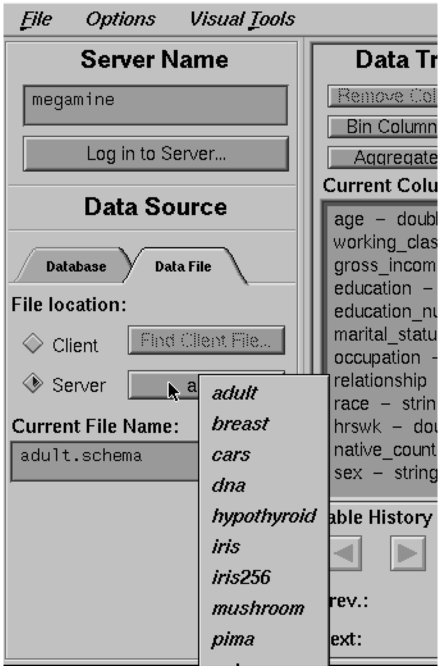
Files:
new_vk_tabbed_deck
NewVk
oldFiles
regression
..schema
cars-out.scatterviz.schema
cars-out.schema
cars19-out.schema

Selection: ɔot/apps/toolmanager/

rs
Bar
- Disk
- Base
Bar
Disk
Base
y Root Level
ny Level 2
ny Level 3
ny Level 4
ny Level 5
ny Level...

Figure 3-3 File Selection Dialog Box

- If you choose *Server*, the menu to the right of the server toggle becomes active, letting you select a server file from a popup menu (Figure 3-4).



Server Name: megamine

File location: ○ Client ● Server

Current File Name: adult.schema

Figure 3-4 Selecting a Server File

If you want to access a data file created outside of Tool Manager, you must create a *.schema* file for it. This is a text file containing a configuration file “input” section, which gives the name of the data file and describes its layout. For a description of the input section, see the appendix for the tool you want to use.

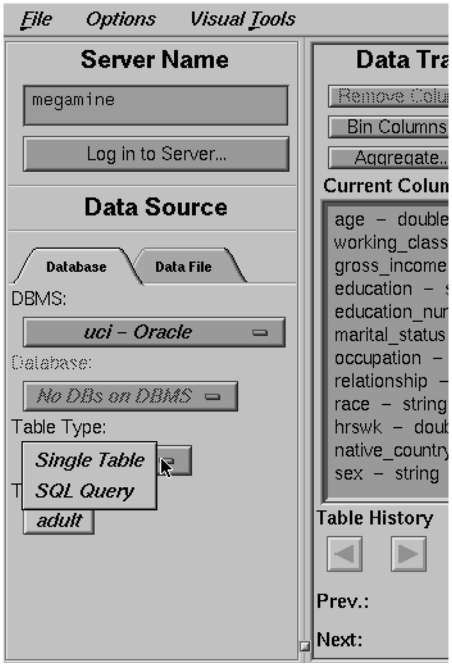
Note: When the Tool Manager and DataMover are running on the same machine, or when the DataMover machine is accessible via NFS, users might try to access files in the DataMover cache directory via the dialog box for selecting client files. Such access corrupts the files. Always access DataMover cache files through the server file popup menu.

Choosing a Database

If you clicked the *Database* tab, the *DBMS* button becomes active after you've logged in to the server. Before you choose a database, the *DBMS* button says *None-Selected*. When you click it, a menu of the database managers appears. Choose the one you want by clicking it. At this point, a dialog window pops up and asks you for your DBMS login name and password before the program grants access to the database. (Note that the login name and password for the DBMS are probably different from those for logging into the server.) After you enter them, the rest of the *Data Source* buttons become active.

The Tool Manager lets you connect to an Oracle, INFORMIX, or Sybase database. If you choose an Oracle database, the *Database* button remains inactive (grayed out) because an Oracle database has only one set of tables per database manager. If, however, you choose an INFORMIX or Sybase manager, the Database button becomes active because each of these managers can have several sub-databases. Click it for a menu of the databases, and choose the one you want to work with.

The *Table Type* popup menu lets you specify whether your data source is a table or view from the database (*Single Table*) or an SQL query (*SQL Query*).



megamine

DBMS: uci – Oracle

Database: No DBs on DBMS

Table Type:

T adult

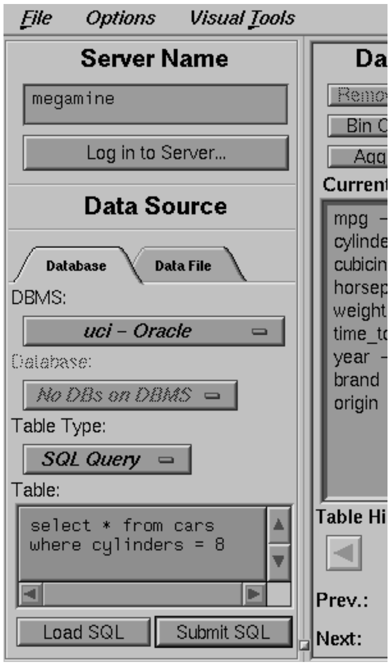
age – double
working_class
gross_income
education – s
education_nur
marital_status
occupation –
relationship –
race – string
hrswk – doub
native_country
sex – string

Figure 3-5 Single Table or SQL Query in the Table Type Menu

Running an SQL Query

To run an SQL query, click the *Table Type* button and choose *SQL Query*. Type your query statement in the window that appears under *Table* (see the example in Figure 3-6).

Instead of typing an SQL query, you can load a previously written query from a file by clicking the *Load SQL* button. A file selection dialog box appears so you can choose the file containing the SQL query.



Server Name: megamine

DBMS: uci – Oracle

Database: No DBs on DBMS

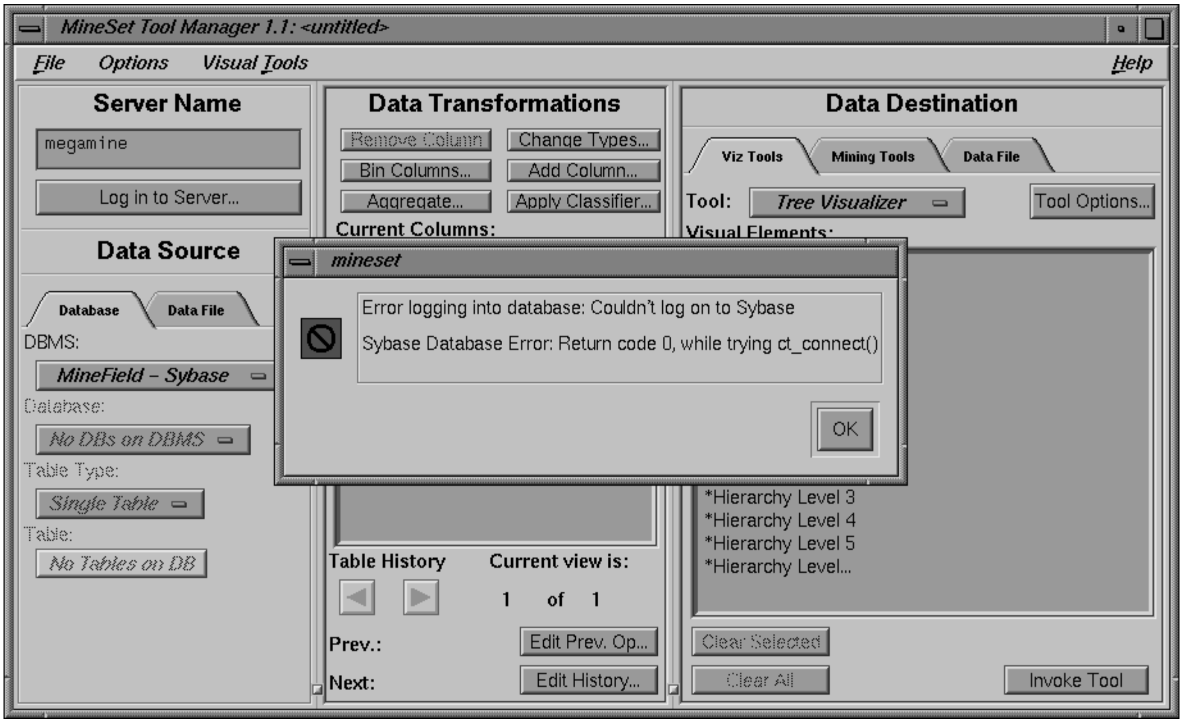
Table Type: SQL Query

Table:

```
select * from cars
where cylinders = 8
```

Figure 3-6 Data Source Panel When You Click the SQL Query Button

After entering the query statement, press *Submit SQL*. If the query is valid, the schema for the table resulting from the query appears in the Data Transformations panel. If there is an error in the query, a dialog box appears, explaining the error (Figure 3-7).



MineSet Tool Manager 1.1: <untitled>

Server Name: megamine

DBMS: MineField – Sybase

Database: No DBs on DBMS

Table Type: Single Table

Table: No Tables on DB

Tool: Tree Visualizer

Current view is: 1 of 1

mineset

Error logging into database: Couldn't log on to Sybase

Sybase Database Error: Return code 0, while trying ct_connect()

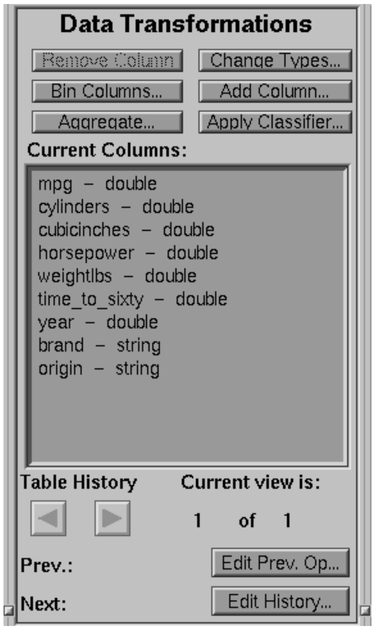
Figure 3-7 Window Displaying Error Message

To work with all the data in a database table, select *Single Table* from the *Table Type* menu. If you want to work with data in multiple tables or a subset of data in one or more tables, run an SQL query to find the information with which you want to work.

When you change servers, the Data Source panel updates its list of tables to reflect those tables found on the new server.

Transforming the Data

The Data Transformations panel lets you manipulate the tables with which you want to work. After you have selected a table (via the Server Name and Data Source panel described above), its column headings appear in the *Current Columns* window of the Data Transformations panel (Figure 3-8).



Current Columns:

```
mpg – double
cylinders – double
cubicinches – double
horsepower – double
weightlbs – double
time_to_sixty – double
year – double
brand – string
origin – string
```

Current view is: 1 of 1

Prev.:

Next:

Figure 3-8 The Data Transformations Panel

The Data Transformation data manipulation options are as follows:

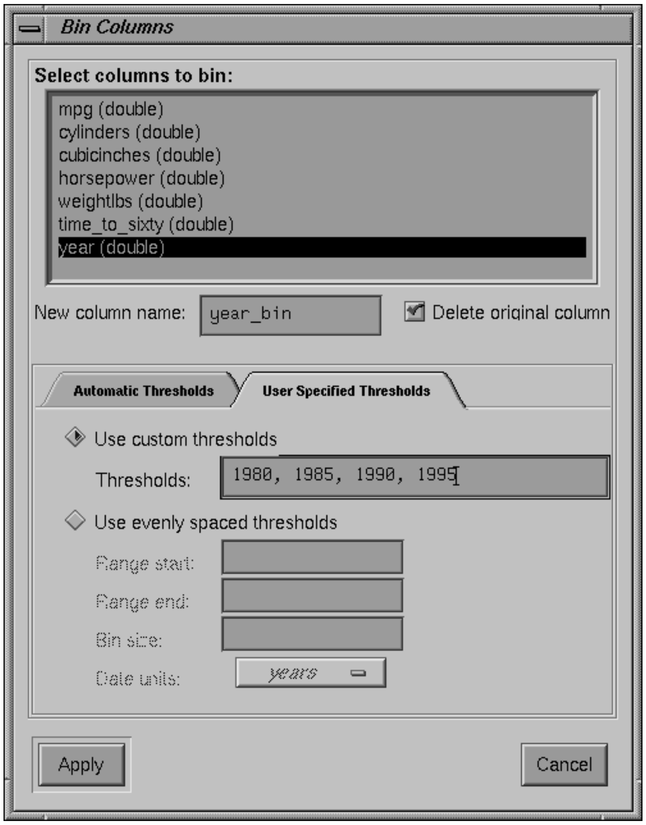
- *Remove Columns*—lets you delete one or more columns that are not relevant to the current visualization or mining.
- *Bin Columns*—lets you take a range of values and assign each record to a group (for example, with a range of ages, 0-18, 19-25, 26-35, and so on).
- *Aggregate*—lets you find aggregations (sum, min, max, and so on), group data into new columns, or make arrays from a column indexed by other columns.
- *Change Types*—lets you change a column's type.
- *Add Column*—lets you add a new column based on a mathematical expression.
- *Apply Classifier*—lets you use a previously created classifier (see Chapter 9 and Chapter 10) on a table for labeling new records.

The Remove Column Button

Remove Column lets you delete columns by selecting the column name or names in the *Current Columns* panel, then clicking this button. The items in the *Current Columns* panel change to show the new table columns. To choose multiple contiguous columns for simultaneous removal, drag the mouse over the columns. To choose multiple non-contiguous columns for simultaneous removal, hold down the Ctrl key while selecting the additional columns.

The Bin Column Button

Binning lets you sort the information from one or more columns into groups in a new column or columns (for example, with a range of ages, 0-18, 19-25, 26-35, and so on). Click *Bin Column* to get a dialog box that lets you specify the binning options (Figure 3-9).



Bin Columns

Select columns to bin:

- mpg (double)
- cylinders (double)
- cubicinches (double)
- horsepower (double)
- weightlbs (double)
- time_to_sixty (double)
- year (double)

New column name: year_bin

☑ Delete original column

● Use custom thresholds

Thresholds: 1980, 1985, 1990, 1995

○ Use evenly spaced thresholds

Range start:

Range end:

Bin size:

Date units: years

Figure 3-9 Bin Columns Dialog Box

This dialog box lets you

- choose the column that is to be divided into bins
- specify the name of the new column to contain values for the bins
- set bin thresholds, or specify a range with thresholds at regular intervals

To specify binning options for one or more columns, select the column name(s), choose the appropriate options below, and click the *Apply* button at the bottom of the dialog box.

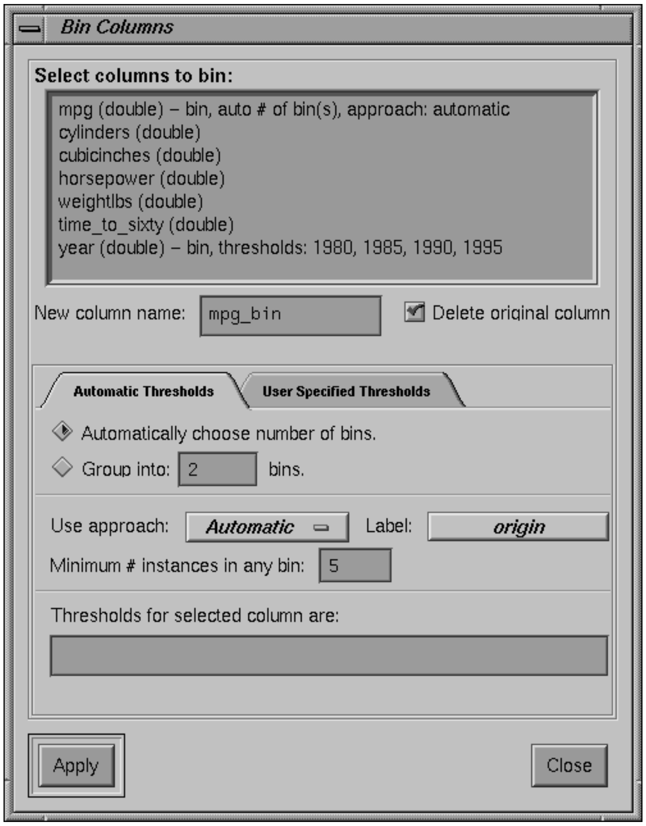
If you select only one column for binning, the name of the resulting binned column appears in the *New column name* box, and you can type in a new name if you like. In the example shown in Figure 3-9, Age_Bin is the name for the new column; in this case, it provides a range of ages. If you select more than one column for binning, *New column name* stays inactive.

Next to *New column name* is a check box labeled *Delete original column*. When chosen, this option automatically deletes the original column after binning. Click the check box to turn this function on or off.

In the middle of the Bin columns dialogue box are two tabs for choosing *Automatic Thresholds* or *User Specified Thresholds*. Choose *Automatic Thresholds* if you'd like the computer to suggest the bins or *User Specified Thresholds* if you'd like to specify the thresholds yourself.

Automatically Computed Thresholds

If you’ve chosen the *Automatic Thresholds* tab, the program can use machine learning to suggest bins.



Select columns to bin:

```
mpg (double) – bin, auto # of bin(s), approach: automatic
cylinders (double)
cubicinches (double)
horsepower (double)
weightlbs (double)
time_to_sixty (double)
year (double) – bin, thresholds: 1980, 1985, 1990, 1995
```

New column name: mpg_bin ☑ Delete original column

◆ Automatically choose number of bins.

◇ Group into: 2 bins.

Use approach: Automatic Label: origin

Minimum # instances in any bin: 5

Thresholds for selected column are:

Figure 3-10 Binning With Automatically Computed Thresholds

The first choice under *Automatic Threshold Computation* is between the *Automatically choose number of bins* and the *Group into: ___ bins* buttons. Click *Automatically choose number of bins* to let the computer decide the best number of bins. If you choose to specify the number of bins, click *Group into: ___ bins*, and type the number of bins you want into the field.

In the *Use approach* menu, you can choose between *Uniform* or *Automatic* binning. If you choose *Uniform*, the algorithm separates the interval into the specified number of bins; the range of the variable is separated into uniformly sized ranges. The upper and lower bounds for the extreme ranges include any values outside the ranges that were seen in the data.

For example, if the values for an attribute are all in the range 5-14, and you choose four ranges, the thresholds are 8 and 11, corresponding to these ranges:

- ≤ 8
- > 8 to 11
- > 11 to 14
- > 14

Uniform lets you decide whether you want to specify the number of intervals or let the algorithm select a number automatically. The automatic selection of the number of bins for the uniform thresholding is based on a number of bins related to the number of distinct values: the more distinct values, the more ranges are chosen (the relationship is logarithmic).

If you choose *Automatic*, you also must select a discrete label. The thresholds are chosen so that the distributions of labels at different ranges are as different as possible. This approach continues to split ranges and create thresholds until no additional interval is considered significant. No interval is split if the two subintervals do not contain the minimum number of instances you can specify (this defaults to 5).

The *Minimum # instances in any bin* text field lets you specify the minimum number of records in any bin; this prevents the creation of bins with fewer records than the number specified.

If you click *Apply*, the Tool Manager picks bin thresholds and displays them in the *Thresholds for selected column are* text field. Output from the process of automatically computing bins appears in a popup window, showing progress of the algorithm and any errors that occur.

Specifying Thresholds

If you specify your own thresholds (as shown in Figure 3-9), you can choose between *Use custom thresholds* or *Use evenly spaced thresholds* by clicking either button. When you type in the thresholds, you must click *Apply* to make those thresholds effective for the selected columns.

The *Use custom thresholds* text box lets you enter the range criteria. For example, you could enter the numbers 18, 30, 50, 60. This results in the following ranges: 0-18, 19-30, 31-50, 51-60, 61+. Note that you enter only the digits and commas, not the ranges.

To specify equally spaced bins over a range of values, click the *Equally Spaced Bins* button. This activates the three text fields below it. You can type the start of the binning range, the end of the range, and the spacing of the bins, respectively, into these fields. If you are binning a column that is a date, you can specify units of time for the bin spacing (using the *Date units* popup menu under the text fields). This would permit you, for example, to bin a time period into bins of three weeks. Dates entered into these fields must be typed in the form “MM / DD / YY”. Possible time units are as follows:

- years
- quarters
- months
- days
- hours
- minutes
- seconds

The *Use custom thresholds* text box accepts dates either in double quotes (as shown below), or without. If you enter dates without quotes, the quotes are added automatically.

```
"1/1/96", "2/1/96", "3/1/96", "4/1/96", "5/1/96", "6/1/96"
```

However, do not put quotes around dates used with *Use evenly spaced thresholds*.

Note: If you enter an invalid parameter, an error message is displayed after you click *Apply*, informing you of the valid options and letting you either cancel the command or return to the dialog box to make the appropriate changes.

Aggregation

Before describing the features and effects of the *Aggregate* button, this section provides an introduction to the concept of arrays and distribution as used in the aggregation feature.

Introduction to Arrays and Distribution

The *Aggregate* button lets you perform simple aggregations (for example, sum, min, max, and so on), make arrays and distribute columns.

Table 3-1 illustrates some sample aggregations / calculations.

Table 3-1 Aggregate Example 1

State	Age_bin	Total $ Spent
CA	0-20	$50
CA	21-40	$454
CA	41-60	$693
NY	0-20	$35
NY	21-40	$541
NY	41-60	$628

If you make *Total $ Spent* into an array indexed by the binned column *Age_bin*, the resulting table, now with only two columns, appear as shown in Table 3-2

Table 3-2 Aggregate Example 2

State	Total $ Spent [Age_bin]
CA	[$50, $454, $693]
NY	[$35, $541, $628]

In this case, making an array reduces the number of columns by one, and also reduces the number of rows by four. Arrays are useful for the Tree Visualizer tool; they are necessary if you want to use sliders in Scatter Visualizer and Map Visualizer displays.

Distributing columns is similar, but different in several important ways. Instead of producing a single new column holding many values, distributing produces one new column for each value of the index. For example, if in the first table *Total $ Spent* were not made an array, but instead distributed by *Age_bin*, Table 3-3 would be the result.

Table 3-3 Aggregate Example 3

State	Total $_0-20	Total $_21-40	Total $_41-60
CA	$50	$454	$693
NY	$35	$541	$628

Thus, distributing increases the number of columns but decreases the number of rows.

If you have more than one binned column (for example, *Age_bins* and *Sex_bin*), you can make a two-dimensional array (indexed by combinations of *Age_bin* and *Sex_bin*). You also can distribute and make an array at the same time.

Table 3-4 has two binned columns: one for age, one for sex.

Table 3-4 Example of binning

State	Age_bin	Sex_bin	Total $ Spent
CA	0-20	1	$20
CA	0-20	2	$30
CA	21-40	1	$220
CA	21-40	2	$234
CA	41-60	1	$401
CA	41-60	2	$292

If you make *Total $ Spent* an array indexed by age, and remove *Sex_bin*, the results are shown in Table 3-5.

Table 3-5 Results When Making Total $ Spent an Array

State	Total $ Spent [Age_bin]
CA	[$50, $454, $693]

If you do not remove *Sex_bin*, the results are shown in Table 3-6.

Table 3-6 Results When Specifying Sex_bin

State	Sex_bin	Total $ Spent [Age_bin]
CA	1	[$20, $220, $401]
CA	2	[$30, $234, $292]

If you make an array by both *Age_bin* and *Sex_bin*, the results are shown in Table 3-7.

Table 3-7 Results of Making an Array by Age_bin and Sex_bin

State	Total $ Spent [Age_bin] [Sex_bin]
CA	[$20, $220, $401, $30, $234, $292]

Finally, if you distribute by *Sex_bin* and index by *Age_bin*, the results are shown in Table 3-8.

Table 3-8 Results of Distributing Sex_bin and Indexing by Age_bin

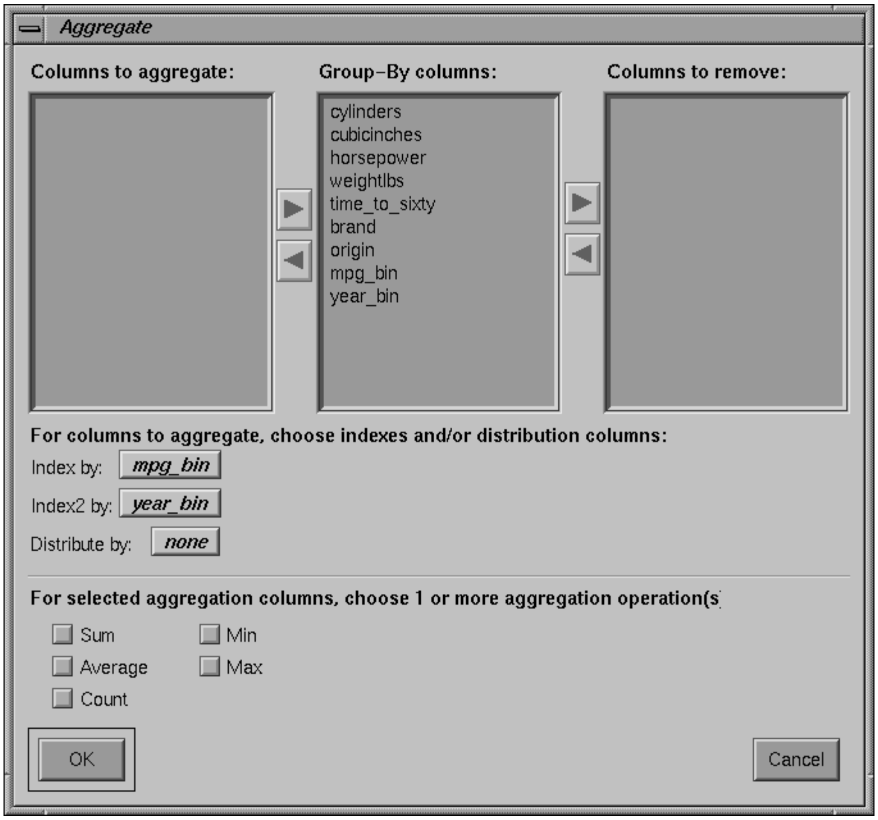
State	Total $ Spent [Age_bin], Sex = 1	Total $ Spent [Age_bin], Sex = 2
CA	[$20, $220, $401]	[$30, $234, $292]

The examples above (with the exception of Table 3-5) had exactly one relevant value for each array element, and the distribution merely rearranged existing data values. In example of Table 3-5, there were two data values for each array element, and these were added. MineSet provides several aggregation options for datasets containing more than one value to be distributed into a given output array element. The most common option is to add the values (as done in Table 3-5). This is useful when accumulating expenditures into budgets, for example. You also can take the minimum, maximum, and average of total number of values, as well as count them.

When distributing values for a given dataset, it is possible that there are no values appropriate for a particular bin. In this case, for MIN, MAX, AVG, and SUM aggregations, the DataMover fills in a value of Null. For COUNT aggregations, the DataMover fills in a value of 0.

The Aggregate Button

You can use the *Aggregate* button to create simple aggregations, make arrays, or distribute columns. Clicking this button causes the Aggregate dialog box to appear (Figure 3-11). It shows three lists, with the columns in the current table appearing in the middle list. If you want to aggregate, distribute, or turn a column into an array, select the name of the column, and click the left arrow button between the left and center lists. Below are popup menus that let you specify indexes (if the result is to be an array) and a distribution column (if the result is to be distributed). In addition, at the bottom of the dialog box are five toggles that let you specify how different values are to be combined when aggregated: either summed, averaged, the min or max value, or the count. When you are aggregating number-valued columns, you can choose any combination of these options. For other types, only count is permitted. If you choose more than one option, you get more than one result. For example, selecting average and max gives you one result with average values, and another one holding the max values.



Aggregate

Columns to aggregate:

Group–By columns:

- cylinders
- cubicinches
- horsepower
- weightlbs
- time_to_sixty
- brand
- origin
- mpg_bin
- year_bin

Columns to remove:

For columns to aggregate, choose indexes and/or distribution columns:

Index by: *mpg_bin*

Index2 by: *year_bin*

Distribute by: *none*

For selected aggregation columns, choose 1 or more aggregation operation(s

□ Sum □ Min

□ Average □ Max

□ Count

Figure 3-11 Aggregate Dialog Box

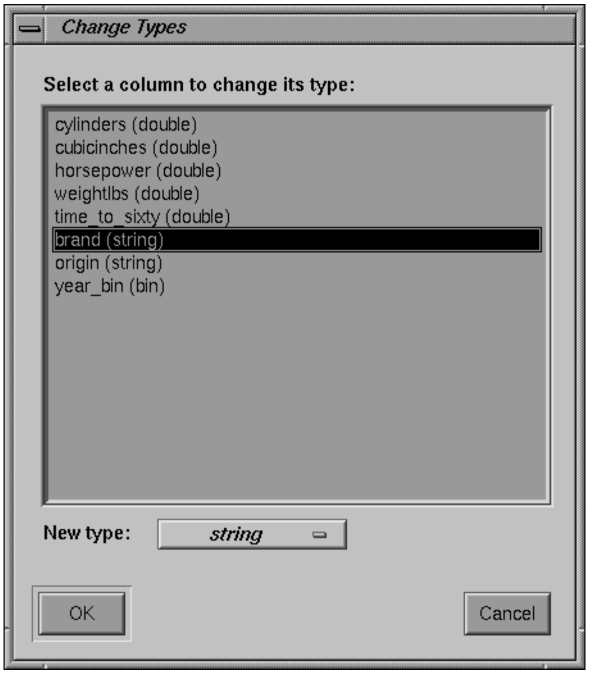
The three lists of column names are given below:

- *Columns to aggregate*.
- *Group-By columns* (the default); this keeps the columns unchanged throughout the operation. For each set of records with the same combination of values in the Group By columns, only one record is output in the resulting table, with values in the aggregated columns summed, averaged, minned, maxed, or counted (depending on the checkboxes at the bottom of the panel).
- *Columns to remove*, as can be seen with the *Sex_bin* column in Table 3-5

After you have finished with the additional aggregate criteria dialog box, the Current Columns text box in the Table Processing window shows the new column names that result from applying these criteria.

The Change Types Button

Some databases store numerical values as strings. Oracle stores all numbers (both integers and real numbers) in a single format, which defaults to the data type *double* in the Tool Manager. You can use the *Change Types* button to ensure that these values are processed correctly. To change the types of one or more columns, click the *Change Types* button. A new dialog box appears (see Figure 3-12). This dialog box contains a window with a list of column headings and their respective types.



Change Types

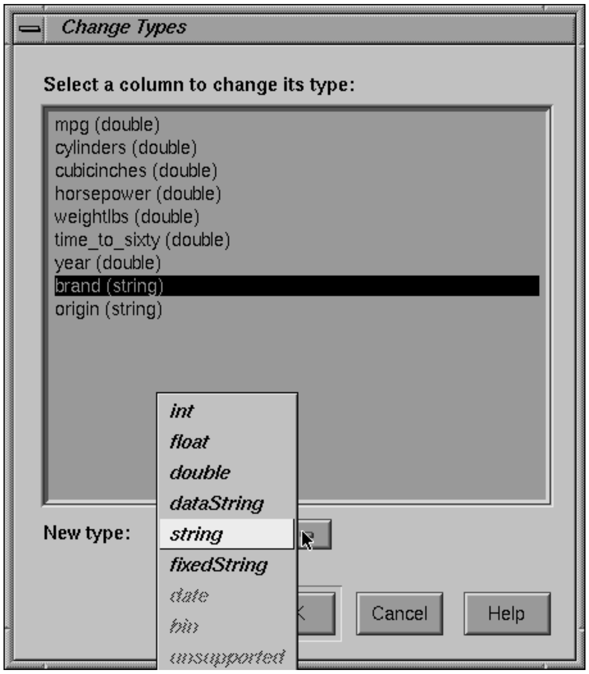
Select a column to change its type:

cylinders (double)
cubicinches (double)
horsepower (double)
weightlbs (double)
time_to_sixty (double)
brand (string)
origin (string)
year_bin (bin)

New type: string

Figure 3-12 Change Types Dialog Box

First select a column heading in the window. Then click the *Column type* button. This produces a popup list of the possible types (invalid types are grayed out), as shown in Figure 3-13.



Change Types

Select a column to change its type:

mpg (double)
cylinders (double)
cubicinches (double)
horsepower (double)
weightlbs (double)
time_to_sixty (double)
year (double)
brand (string)
origin (string)

New type: string

Figure 3-13 Types Popup List

- *int*—represents a 32-bit signed integer.
- *float*—represents a single-precision floating-point number. The decimal point is optional when representing a floating-point number.
- *double*—represents a double-precision floating-point number. The decimal point is optional when representing a floating-point number.
- *dataString*—represents a string that is unlikely to appear multiple times. If it appears multiple times, several copies are made. A *dataString* can be used to store an address. Addresses are unlikely to be compared, and each record can have a different address.

- *string*—represents a string of characters that can appear multiple times in the data file. Unlike a *dataString*, only a single copy of a given string is stored in memory, no matter how many times it appears in the data. This saves memory for strings appearing many times.

  Comparing *strings* is also much quicker than comparing *dataString*s. However, reading in strings can be slower than reading in *dataString*s because it is necessary to look for duplications. An example of *string* use would be for a division name that appears once for each department in the division. If you are unsure whether to use a *string* or a *dataString*, use a *string*.
- *fixedString*—represents a string that is stored internally as a fixed-length array of characters. These are useful when all the elements in a data column are known to be approximately the same length. They also are the most efficient way of encoding very short strings, such as State abbreviations.
- *date*—represents the date type from the database.
- *bin*—represents a column created by a binning operation.
- *unsupported*—represents a database type not supported by MineSet (for example., images).

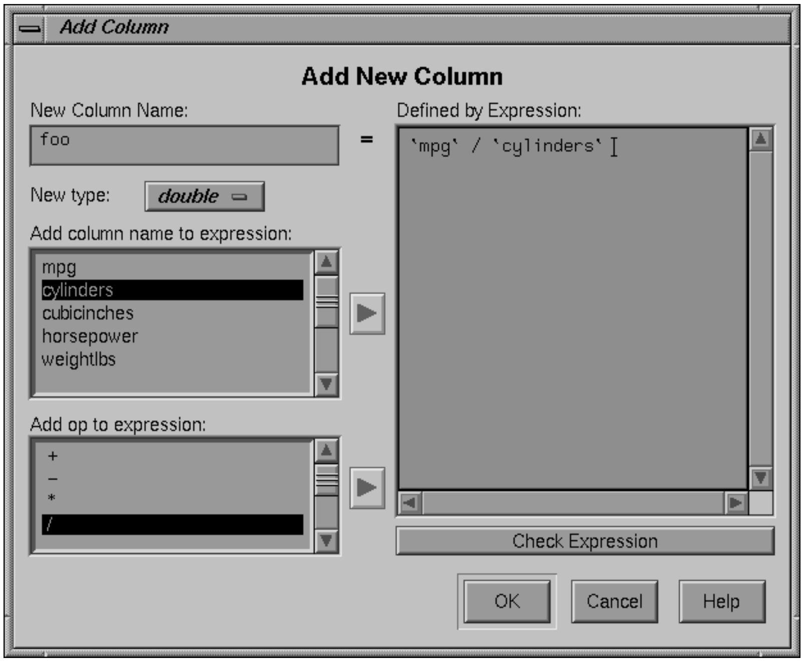
After selecting a new type, click the *OK* button to have the change take effect.

Note that if you try to convert an inappropriate field (such as a name) to a number, the resulting values are all zeroes.

Note: When the data source is an existing file, there are fewer possibilities available for changing any given column.

The Add Column Button

You can use the *Add Column* button to create a new column whose values are computed based on a mathematical expression. For example, you could add a new column whose values are the ratio of values from two existing columns. Click *Add Column* to get a dialog box that lets you specify the new column name and expression (Figure 3-14).



New Column Name: foo

New type: double

Defined by Expression: `mpg` / `cylinders`

Figure 3-14 The Add Column Dialog Box

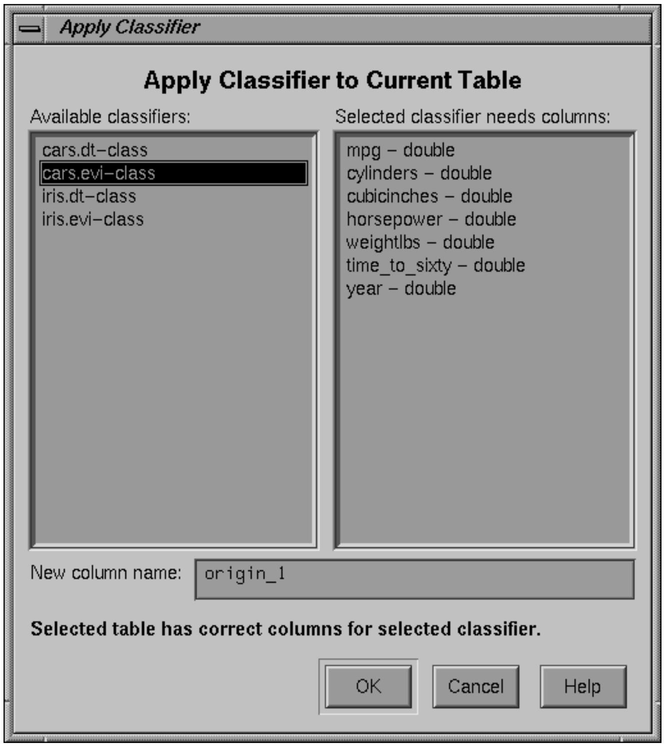
In the upper left of this dialog box is a field for entering the new column's name. Below this is a popup menu that lets you specify the column type (integer, string, floating point, and so on).

The right-hand side of the dialog contains a large text entry area where you can type in a definition of the expression (for a complete description of the expression definition language, see “The Configuration File” in Appendix A). As a shortcut to typing column names and operators, scrolled lists in the lower left of the dialog display all columns in the current table and all possible operators. To insert a column name or operator into the expression, either double-click it in its scrolled list, or select it and click the arrow button to the right of the scrolled list.

To check the syntax of the expression you have created, click the *Check Expression* button. If there is an error, a dialog box appears, indicating what the error is and where it occurred. When you click *OK*, the expression is automatically checked, and the dialog box is not removed unless the expression is correct.

The Apply Classifier Button

The *Apply Classifier* button lets you use a previously created classifier (see the “Using Mining Tools” section) to label records in the current table. For example, if you created a classifier to determine the edibility of mushrooms based on their shape, size, smell, and so on, you could use *Apply Classifier* to determine edibility of mushrooms in a new table. Click *Apply Classifier* to access a dialog box that lets you apply previously created classifiers to the current table (Figure 3-15).



Apply Classifier

Apply Classifier to Current Table

Available classifiers:

- cars.dt–class
- cars.evi–class
- iris.dt–class
- iris.evi–class

Selected classifier needs columns:

- mpg – double
- cylinders – double
- cubicinches – double
- horsepower – double
- weightlbs – double
- time_to_sixty – double
- year – double

New column name: origin_1

Selected table has correct columns for selected classifier.

Figure 3-15 The Apply Classifier Dialog Box

On the left of this dialog box is the list of all classifiers currently available on the server. If you select a classifier, the right-hand list is filled in with the column names and types required by that classifier. If these requirements match the current table, a message at the bottom states this, and the *OK* button is activated. If the current table does not have all the columns required for the selected classifier, the message at the bottom states this, the columns that are missing are selected in the list on the right, and the OK button is deactivated. Finally, a text field at the bottom lets you name the new column to be created by the classifier.

The Table History Buttons

Table processing is a series of operations performed by using the buttons described above. To allow you to see this series of steps, and go back if you made a mistake, there are two *Table History* buttons at the bottom of the Table Processing panel (Figure 3-16). When you click the left arrow button, the columns window shows the table as it appeared at an earlier step. Clicking the right arrow button brings the table forward to its current state.

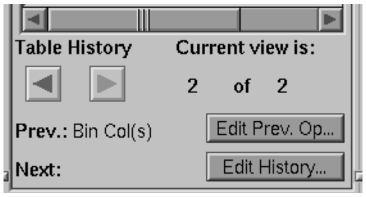


Table History

Current view is:

2 of 2

Prev.: Bin Col(s)

Next:

Figure 3-16 Table History Buttons

The Current view is Field

To the right of the history buttons is the information field *Current view is*, which counts the changes you've made and indicates which step you are viewing. The two integers in this field indicate which table view you're looking at, out of the total number of table views that exist. For example, if you've made two changes, you can view the original table (*1 of 3*), the table after the first change (*2 of 3*), or the table after the second change (*3 of 3*).

The Prev and Next Buttons

As you go back and forth using the *Table History* buttons to view earlier versions of the table, the *Prev* and *Next* fields (under the arrow buttons) help you keep track of where you are in the history of the table. For any table you view, the *Prev:* field tells you what the previous change was, and the *Next:* field tells you the next change.

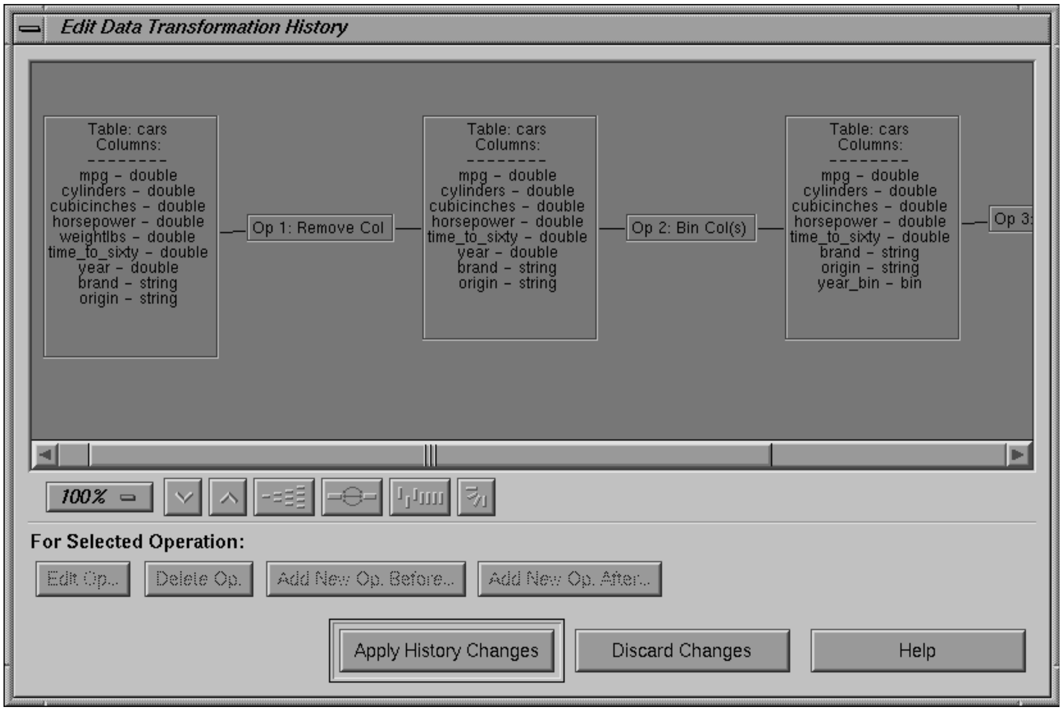
The Edit Prev. Op Button

The *Edit Prev. Op.* button allows you to edit the operation shown in the *Prev.* field. (This button does not work when *Current view is: 1 of n*, because that is the original table, with no previous changes.) When you click the *Edit Prev. Op.* button, the dialog box for the previous operation comes up, and you can make changes to that operation. For example, if the previous operation was binning columns, when you click *Edit Prev. Op.*, the *Bin Columns* dialog box appears.

Note that by changing a previous operation, you could affect operations you set up subsequent to the current one. For example, if you delete a column that you used in a subsequent binning operation, that binning operation becomes invalid. The *Edit History* button can help you avoid problems.

The Edit History Button

When you click the *Edit History* button, the Table History dialog box appears and shows you the complete history of the *Data Transformation* table (Figure 3-17). Each version of the table appears as a box containing a list of the columns, linked by a smaller box (indicating the operation performed on the table) to the next version of it.



```
Table: cars
Columns:
--------
mpg - double
cylinders - double
cubicinches - double
horsepower - double
weightlbs - double
time_to_sixty - double
year - double
brand - string
origin - string
```

```
Table: cars
Columns:
--------
mpg - double
cylinders - double
cubicinches - double
horsepower - double
time_to_sixty - double
year - double
brand - string
origin - string
```

```
Table: cars
Columns:
--------
mpg - double
cylinders - double
cubicinches - double
horsepower - double
time_to_sixty - double
brand - string
origin - string
year_bin - bin
```

Figure 3-17 Table History Dialog Box

As with *Edit Prev. Op*, changing one operation usually affects (sometimes invalidates) subsequent operations in the history.

Zoom Buttons

Under the window displaying this flow chart are the zoom buttons that allow you to view the flow chart closer up or farther away. You can choose the zoom by using the button indicating the percentage, or by clicking the arrow buttons to increase or decrease the size. The increments of change are the same whether you use the percentage button or the arrow buttons.

Vertical/Horizontal View Button

Next to the zoom buttons is a toggle button that allows you to view the flow chart vertically or horizontally. Clicking the button switches you back and forth between the two points of view.

For Selected Operation

Under the indicator *For Selected Operation* is a row of buttons that becomes active if you click one of the operations in the flow chart. Once you select an operation, you can alter it. The *Edit Op* button brings up the dialog box for the selected operation, so you can make changes to it. The *Delete Op* button will remove the operation from the table history, and the elements that follow in the flow chart move over when it disappears. The *Add New Op. Before* and *Add New Op. After* buttons let you insert a new operation into the table history.

Applying or Discarding the Changes

If you decide not to carry out these changes, click *Discard Changes*; the changes you made are ignored, and you return to the *Data Transformation* panel. You might choose *Discard Changes* if, for example, you delete a column that was used in a subsequent binning operation, and the binning operation and linked table also disappear. If that consequence was not what you wanted, *Discard Changes* allows you to undo your choice.

If the changes you’ve made in *Table History* are what you want, click *Apply History Changes* to implement the changes and return to the *Data Transformation* panel.

Investigating the Data

The *Data Destination* panel (Figure 3-18) lets you direct your processed data to one of the SGI_MineSet visualization or mining tools or to a data file.

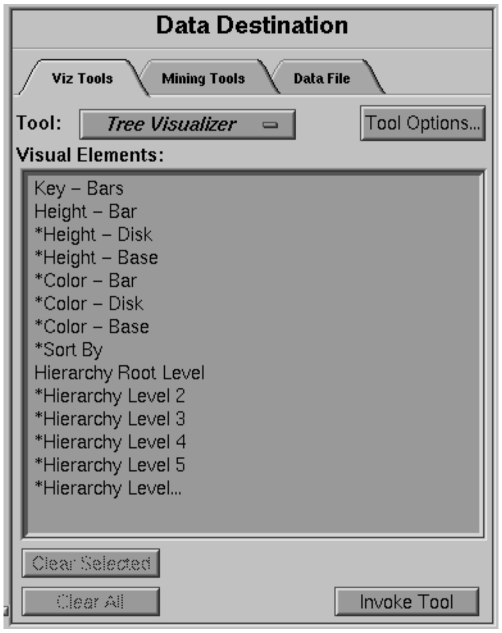
There are three tabs at the top of this panel:

- *Viz Tools*
- *Mining Tools*
- *Data Files*

These are the three possible destinations for your data. They are discussed in greater detail in later chapters dealing with the Data Destination tools.

Using Visualization Tools

If you choose the *Viz Tool* tab, the visualization tool panel appears under Data Destination.



Tool: Tree Visualizer

Key – Bars
Height – Bar
*Height – Disk
*Height – Base
*Color – Bar
*Color – Disk
*Color – Base
*Sort By
Hierarchy Root Level
*Hierarchy Level 2
*Hierarchy Level 3
*Hierarchy Level 4
*Hierarchy Level 5
*Hierarchy Level...

Figure 3-18 Data Destination Panel

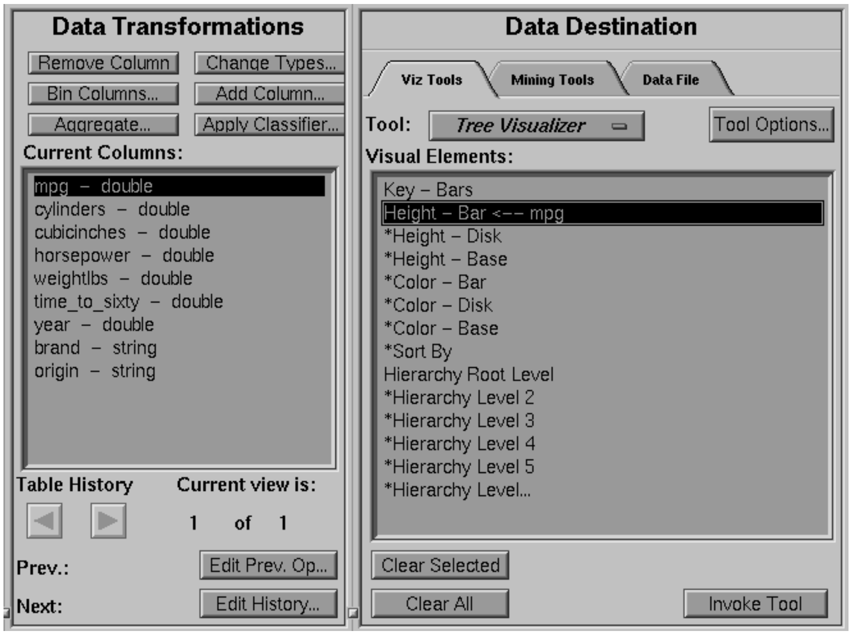
Viz Tool is a popup menu that lets you choose among *Tree Visualizer, Map Visualizer, Scatter Visualizer,* and *Text Editor* to determine the type of visual representation you want for your data. The first three tools are described in their respective chapters. The Text Editor tool launches a text editor in a separate window for viewing the raw data file. The Text Editor has no requirements and no options. The specific editor that is launched defaults to jot. If the user has the WINEDITOR environment variable, that editor is used. If WINEDITOR has not been set, but EDITOR has, that editor is used.

- *Tool Options*—lets you further specify options you want to set in the specified tool's configuration file.
- *Clear Selected*—lets you undo the mapping to a selected Visual Element.
- *Clear All*—clears all mappings.
- *Invoke Tool*—lets you start the tool you specified (via the top button) using the configuration file named in the *Saved as* text field.

Each tool’s requirements are listed individually in the *Visual Elements* pane. This pane lets you map a table column to a requirement. To do this,

1. Select a column by clicking its name in the Current Columns pane.
2. Select the requirement which you want to map the column by clicking on that requirement in the *Visual Elements* pane.

The *Viz Tool* panel now shows the Visual Element and the column to which it has been mapped (see Figure 3-19).



Current Columns:

mpg – double
cylinders – double
cubicinches – double
horsepower – double
weightlbs – double
time_to_sixty – double
year – double
brand – string
origin – string

Current view is: 1 of 1

Tool: Tree Visualizer

Visual Elements:

Key – Bars
Height – Bar <-- mpg
*Height – Disk
*Height – Base
*Color – Bar
*Color – Disk
*Color – Base
*Sort By
Hierarchy Root Level
*Hierarchy Level 2
*Hierarchy Level 3
*Hierarchy Level 4
*Hierarchy Level 5
*Hierarchy Level...

Figure 3-19 Columns Mapped to Requirements

You can clear the mapping at any time by selecting the requirement that has the mapping you want to change, then clicking the *Clear Selected* button. You can clear all mappings using the *Clear All* button.

If you want to specify other details to fine-tune your mappings or to change the settings so that the data representations more clearly reflect your intentions, click the *Tool Options* button. A dialog box specific to each MineSet tool appears, where you can manually specify the options to use.

Note: For details on a specific tool's options, see that tool's chapter.

Using Mining Tools

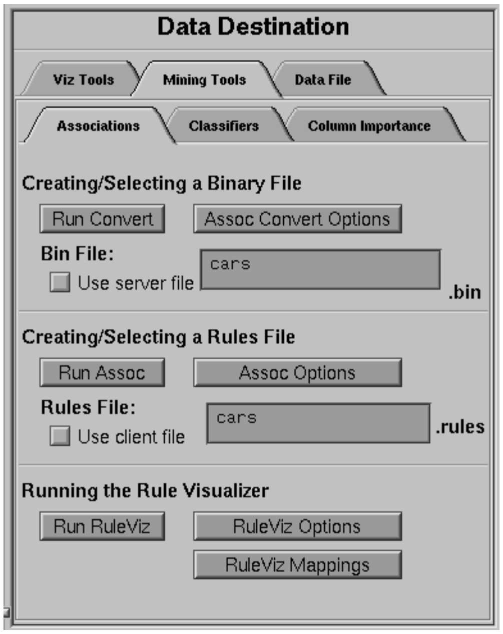
The MineSet Classifiers are described in Chapter 8, "MineSet Inducers and Classifiers," Chapter 9, "Inducing and Visualizing the Decision Tree Classifier," and Chapter 10, "Inducing and Visualizing the Evidence Classifier." Column importance is described in Chapter 11.

Creating Associations for the Rule Visualizer

If you click the *Associations* tab, the panel lets you take the data file you created in Data Transformations and proceed to the *Rule Visualizer.* Each step of the process is shown in the boxes in the panel:

- *Creating/Selecting a Binary File*—creates a binary file from your data file
- *Creating/Selecting a Rules File*—runs the Assoc program on the binary file
- *Running the Rule Visualizer*—runs the Rule Visualizer

If you don’t want to go through this process manually, click the *Run Rule Viz* button, and the computer will perform the process using defaults.



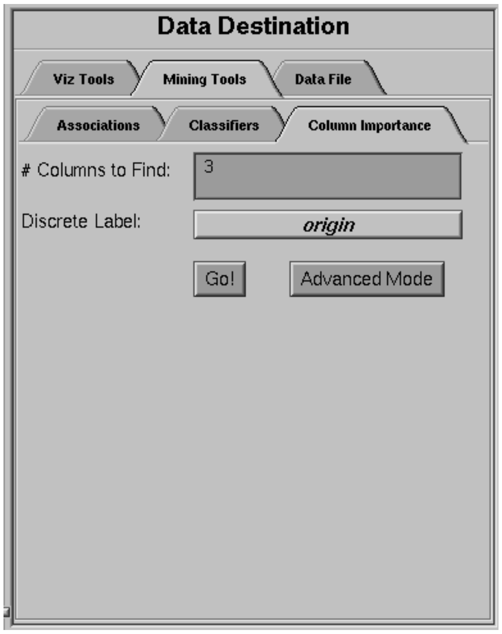
Bin File: cars .bin

Rules File: cars .rules

Figure 3-20 The Associations Tab

Finding Important Columns

Column Importance (Figure 3-21) determines how important various columns are in discriminating the different values of the label column you choose. You might, for example, want to find out the best three columns for discriminating the label *good credit risk* so you can choose them for the Scatter Visualizer. When you select the label and click *Go!*, a popup window appears with the three columns that are the best three discriminators. A measure called "purity" (a number from 0 to 100) informs you how well the columns discriminate the different labels. Adding more columns can only increase the purity.



Columns to Find: 3

Discrete Label: origin

Figure 3-21 The Column Importance Tab

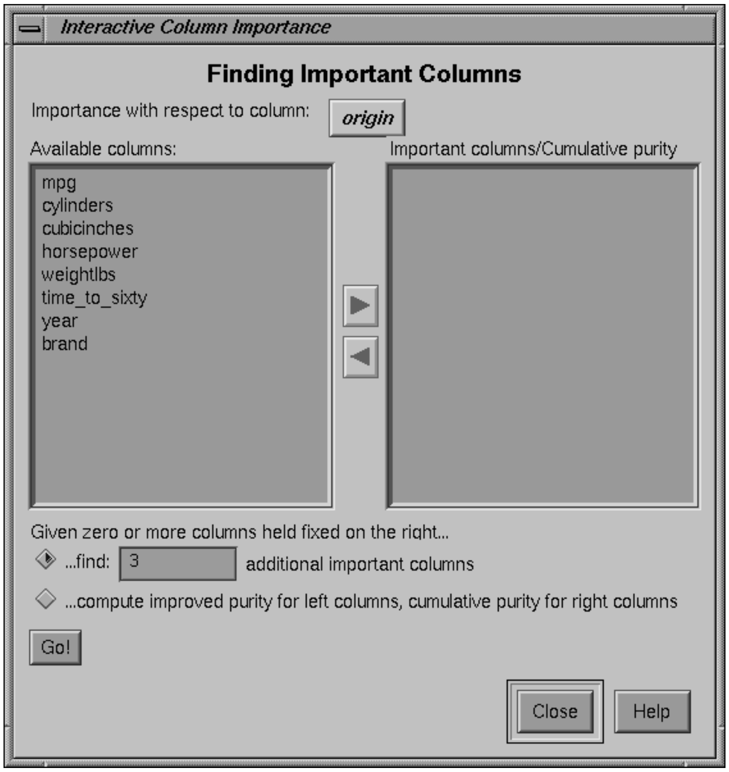
There are two modes of column importance:

- Simple Mode

  To invoke the simple mode, choose a discrete label from the popup menu, and specify the number of columns you want to see.

- Advanced Mode

  Advanced mode lets you control the choice of columns. To enter advanced mode, click *Advanced* in the Column Importance panel. A dialog box appears, as shown in Figure 3-22. The dialog box contains two lists of column names: the left list contains available attributes, and the right list contains attributes chosen as important (by either the user or the column importance algorithm).



Finding Important Columns

Importance with respect to column: origin

Available columns:

mpg
cylinders
cubicinches
horsepower
weightlbs
time_to_sixty
year
brand

Important columns/Cumulative purity

Given zero or more columns held fixed on the right...

● ...find: 3 additional important columns

○ ...compute improved purity for left columns, cumulative purity for right columns

Figure 3-22 Advanced Mode of Column Importance

Advanced mode can work two different ways: finding several new important attributes, or ranking available attributes.

- Finding Several Important Attributes

  To enter this sub-mode, click the first of the two radio buttons at the bottom of the dialog (*...find [number] additional important attributes*). If you click *Go!* with no further changes, the effect is the same as if you were in Simple Mode, finding the specified number of important columns and automatically moving them to the right column. Near each column, the cumulative purity is given (that is, the purity of all the columns up to and including the one on the line). More attributes can only increase the purity.

  Alternatively, by moving columns names from the left list to the right list, you can pre-specify columns that you want included and let the system add more. For example, to select the age column and let the system find three more columns, click the age column name, then click the right arrow.

  Clicking *Go!* lets you see the cumulative purity of each column, together with the previous ones in the list. A purity of 100 means that using the given columns, you can perfectly discriminate the different label values.

- Ranking Available Attributes

  Advanced Mode also lets you compute the change in purity that each column would add to all those that were already selected. For example, you might choose *age*, and then ask the system to compute the incremental improvement in purity that each column would yield.

  To enter this sub-mode, click the second of the two radio buttons at the bottom of the dialog (*...compute improved purity for attributes on the left.*). This sub-mode permits fine control over the process. If two columns are ranked very closely, you might prefer one over the other (for example, cheaper to gather, more reliable, easier to understand).

Column Importance Notes

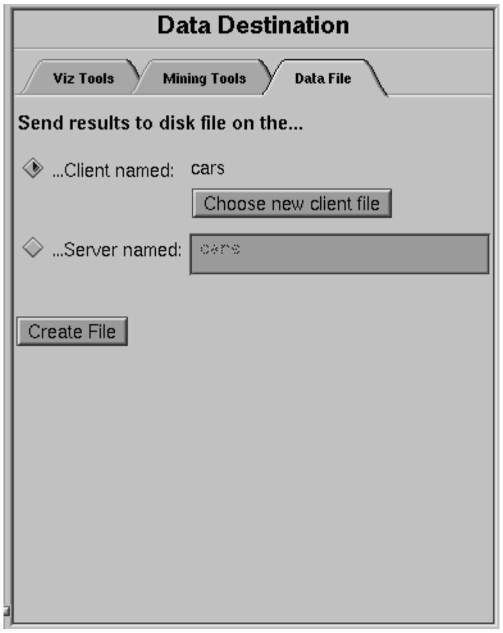
Note that with other columns, the importance of features varies from their ranking alone. For example, while *net-income* might be a good column individually, it might not be as important together with *salary* because they are likely to be highly correlated. The best set of three columns is not necessarily composed of the columns that rank highest individually. If two columns give the income in dollars and in another currency, they are ranked equally alone; however, once one of them is chosen, the other adds no discriminatory power to the set of best features.

Column selection is useful for finding the best three axes for the Scatter Visualizer, as well as for finding a good discriminatory hierarchy for the Tree Visualizer.

All floating point values (double or float) are pre-discretized using the automatic discretization. If a column has no value given to it in the left list, the algorithm did not consider it, because it either had a single value (for example, when it is discretized into one interval), or the number of records that it would separate are not statistically significant.

Using Data Files

The Tool Manager lets you save the manipulated table for future use in a data file on the client or server. If you click the *Data Files* tab, the panel shown in Figure 3-23 appears.



Data Destination

Send results to disk file on the...

◆ ...Client named: cars

◇ ...Server named: cars

Figure 3-23 The Data Files Panel

The two toggle buttons in this panel let you specify whether the file is to be saved on the server or your client machine. The selected name for the client file appears next to the Client checkbox. If you select *Client*, the *Choose new client file* button brings up a dialog for you to choose the name for the client file. If you select *Server*, you can type the server filename directly into the adjacent text field.

Note: Pathnames are not permitted for server files; all server files are stored in the Datamove cache directory.

Session Files

The Tool Manager can save a description of your work to a "session file" for future use. A session file contains a description of the data source you selected, all the transformations on the data, and the mining or visualization of the data. Each session file can hold descriptions of only one data source and one data destination; thus, if you change the destination visual tool or source data table, the session file loses its links to any previous data source or destination.

Session files can be saved at any time through the entries in the File menu, described below. The name of the current session appears in the window's title bar. The Tool Manager also keeps a parallel session file, called *.latest.mineset*, in your home directory. It always has a record of your most recent actions in the Tool Manager. Whenever you start the Tool Manager without a session file, it reads the contents of the *.latest.mineset* file to return you to the state when you last ran MineSet.

Session files also can be used for running the Tool Manager in batch mode, by issuing this command at the UNIX shell prompt:

```
mineset_batch [-s serverPassword -d databasePassword]
configFile
```

The **-s** and **-d** options allow you to specify the password for logging into the server and database respectively. If you do not specify these options, *mineset_batch* will ask you to type in the passwords, thus these options are useful when running *mineset_batch* from a shell script. To specify that there is no password for either the server or database, use **-s** or **-d** followed by two double quotes, that is,

```
mineset_batch -s "" -d "" foo.mineset
```

If you specify one of the two passwords, you must specify both.

In batch mode, the Tool Manager does not bring up tools or windows; however, it creates files for tools. For example, if the session file includes the Tree Visualizer as the data destination, running the Tool Manager in batch mode produces files for running the Tree Visualizer, but the Tool Manager does not invoke it.

Pulldown Menus

At the top of the screen are four pulldown menus:

- File
- Options
- Visual Tools
- Help

The following sections, describe each of these menus.

The File Menu

The File menu lets you choose what to do with your current session, which is one complete session with a tool. This includes choosing the server, data source and table, all the table manipulations, and the mapping or classifying of the data.

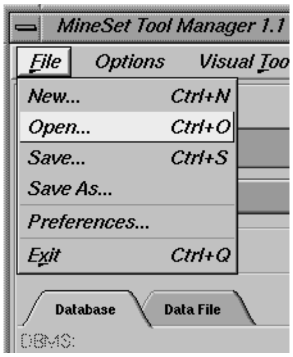


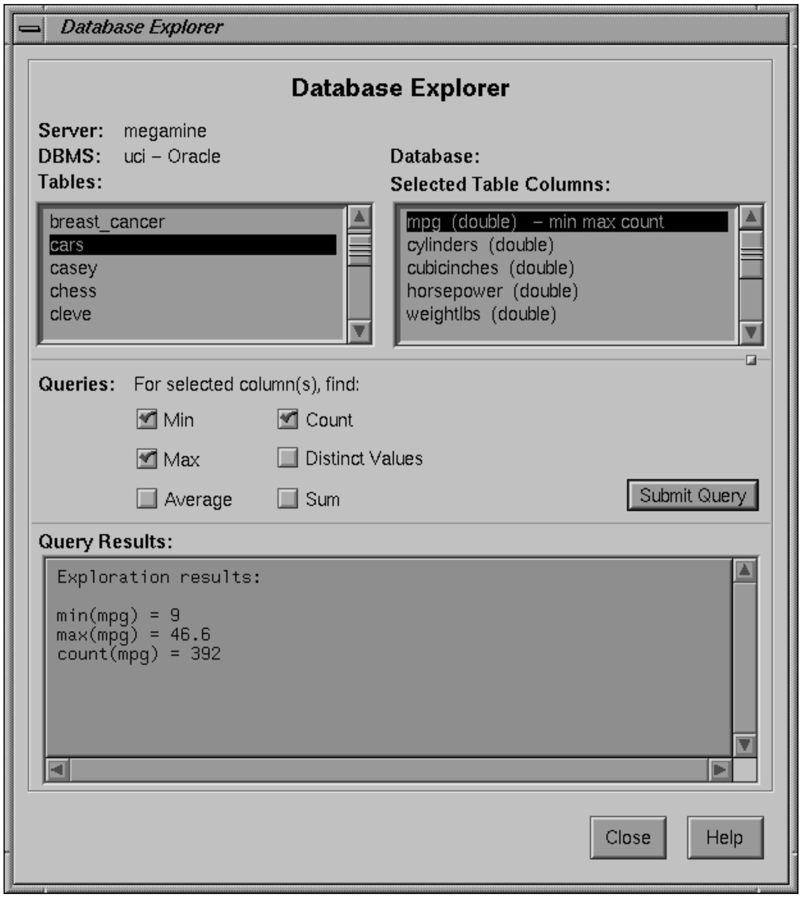
Figure 3-24 File Menu

The File menu provides five functions:

- New—Lets you start a new session, by setting your session name to "untitled."
- Open—Lets you restore a session that has been saved previously.
- Save—Saves the current session.
- Save As—Lets you save the current session under a new name.
- Preferences—Lets you set two global Tool Manager options. When you choose this menu item, a dialog box appears with two toggles. The first, Include NULLs in aggregation arrays, specifies whether arrays and distributions created using the Aggregation dialog (see "Aggregation" on page 37) have a slot for null values. The second, Automatically restore session on startup, determines whether the Tool Manager tries to return you to the same place the next time you start up.
- Exit—Exits the Tool Manager.

The Options Menu

The Options menu has only one item: Database Explorer. Choosing this item brings up the Database Explorer (Figure 3-25), a tool to help you find out about the information in database tables. For example, it can be used to find maximums, minimums, averages, sums, and distinct values.



Database Explorer

Server: megamine

DBMS: uci – Oracle

Database:

Tables:

breast_cancer
cars
casey
chess
cleve

Selected Table Columns:

mpg (double) – min max count
cylinders (double)
cubicinches (double)
horsepower (double)
weightlbs (double)

Queries: For selected column(s), find:

☑ Min ☑ Count

☑ Max □ Distinct Values

□ Average □ Sum

Query Results:

```
Exploration results:

min(mpg) = 9
max(mpg) = 46.6
count(mpg) = 392
```

Figure 3-25 The Database Explorer

The Database Explorer has three panels: the top lets you select the table and columns, the middle provides summary information, and the bottom provides summary results. The upper left shows a list of all the tables in the current database. Selecting any of these tables causes the columns of that table to appear in the list to the right. To get summary information, select one or more column names, and click the desired operations in the center panel. When you click the *Submit Query* button in the middle panel, the summary information is computed for the specified columns (this can take a long time for large tables). Results are output at the bottom. Note that *min*, *max*, *average*, and *sum* can be applied to number-valued columns only.

Note: The count option in the Database Explorer produces a count of all rows in the table, not just those with non-null values.

The Visual Tools Menu

The Visual Tool menu allows you to invoke any of the visual tools directly:

- Evidence Visualizer
- Map Visualizer
- Rule Visualizer
- Scatter Visualizer
- Tree Visualizer

If you've created a file that runs within one of these tools and you want to go back to it, click the tool. From within the tool, use File Open to open the data file.

The Help Menu

The Help menu provides information about the elements of the Tool Manager and how they work:

- Click for Help—Gives help information about a particular item if you press *Shift-F1*, then click the item for which you want help.
- Overview—Gives an overview of the online help and how to use it.
- Index—Provides an index of the complete help system. This option is currently disabled.
- Keys & Shortcuts—Provides the keyboard shortcuts for all of the Tree Visualizer's functions that have accelerator keys.
- Product Information—Indicates what version of the Tool Manager you are using.
- *MineSet User's Guide*—Invokes the IRIS Insight viewer with the online version of this manual.

Color Options for the MineSet Visualizers

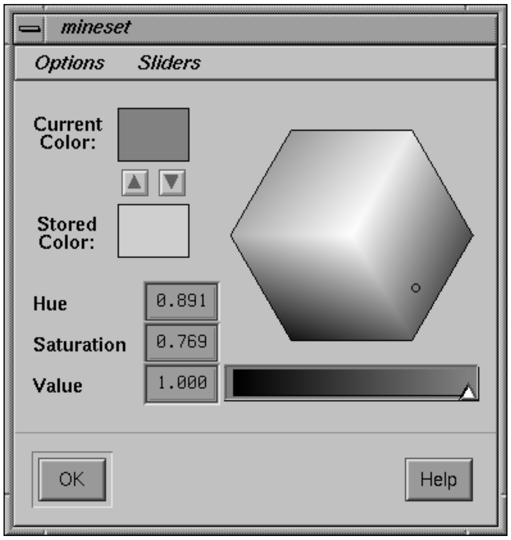
Many of the tool option dialogs have options for choosing colors. MineSet 1.1 has a color list chooser that uses color swatches. This section describes how to choose, apply, and change color options for the MineSet Visualizers.

Choosing Colors

If only one color is to be chosen (for example a grid color), a single color swatch appears (Figure 3-26).

Figure 3-26 Configuration Option With a Single Color Swatch

Clicking the swatch brings up a Color Browser that lets you change the color of that swatch (Figure 3-27). The Color Browser is described in more detail in the "Using the Color Browser" section, shown in Figure 3-27.



Hue 0.891
Saturation 0.769
Value 1.000

Figure 3-27 Color Browser

If a list of color swatches is to be chosen, the list of swatches appears (these can be empty initially), as shown in Figure 3-28.

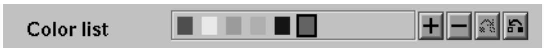


Figure 3-28 Multiple Colors Swatches

To edit the color, click a swatch with the left mouse button. This also selects the swatch for making changes to the colors with the buttons. If you click with the swatch with the middle mouse button, the swatch is selected, but the color chooser does not appear.

Next to the list of swatches are four buttons. First is the odd button, labeled with a plus sign (+*)*, adds a new color at the end of the list. A swatch is added, and the color chooser appears, where you can select the color of that swatch. The add button is disabled if the maximum number of colors is already in the list.

Next to the add button is a delete button, labeled with a minus sign (-*)*. This button deletes the selected color. It is disabled if no swatch is selected, or if the list already has the minimum number of colors.

Next to the delete button are two buttons to shift the selected color right and left. These buttons are disabled if no swatch is selected, or if the swatch is already at the end of the list.

If there are more colors in the list than room to display them, scroll arrows are added at each end of the list (Figure 3-29).

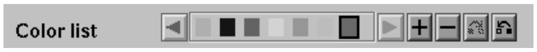


Figure 3-29 Scroll Arrows on Color Browser

If the hardware runs out of colors, the color swatches are replaced with text labels showing the color in X notation (Figure 3-30).

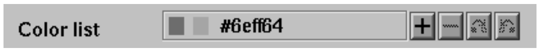


Figure 3-30 Color Browser Out of Colors

Using the Color Browser

The Color Browser (Figure 3-27) appears when you click a color swatch or the add button in the Colors panel of the visualizer's Configuration Options panel.

To select a color using the Color Browser:

1. Move your mouse cursor on top of the small circle in the colored hexagon.
2. Press the left mouse button, and move your mouse around the hexagon. The color beneath the small circle appears in the rectangle next to the *Current Color* label. This rectangle acts as your color palette while you choose a color.
3. Release the mouse button when the small circle is on top of a color you want. The selected swatch immediately takes on the chosen color.

   You can edit several colors without dismissing the Color Browser; clicking any color in the options panel lets you edit that color in the already posted Color Browser.
4. Click the *OK* button when you decide on a color. The Color Browser window closes.

Chapter 4

Using the Tree Visualizer

This chapter discusses the features and capabilities of the Tree Visualizer. It provides an overview of this database visualization tool, discusses ways of invoking it, then explains the Tree Visualizer's functionality when working with the following elements.

- main window
- external controls
- pulldown menus
- overview window

Finally, this chapter lists and describes the sample files provided for this tool.

Overview of Tree Visualizer

The Tree Visualizer is a graphical interface that displays data as a three-dimensional "landscape." It presents your data as clustered, hierarchical blocks (nodes) and bars through which you can dynamically navigate, viewing part, or all, of the dataset.

As shown in Figure 4-1, the Tree Visualizer displays quantitative and relational characteristics of your data by showing them as hierarchically connected nodes. Each node contains bars whose height and color correspond to aggregations of data values. The lines connecting nodes show the relationship of one set of data to its subsets.

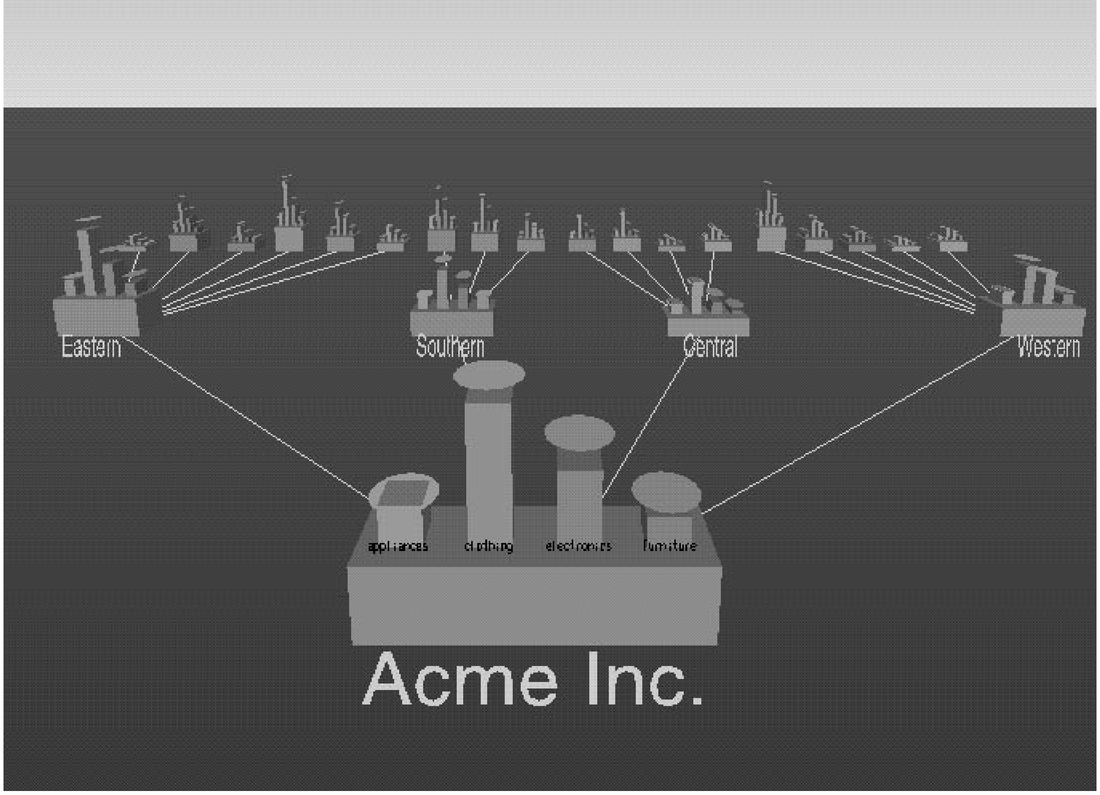


Figure 4-1 Example Display in the Tree Visualizer's Main Window

Values in subgroups can be summed and displayed automatically in the next higher level. The base under the bars can provide information about the aggregate value of all the bars. Bars representing negative values are shown below the top of the base. You can see negative value bars more clearly by disabling the base height (see "The Display Menu" on page 108, or the "Base Height Statements" section in Appendix A, "Creating Data and Configuration Files for the Tree Visualizer").

File Requirements

The Tree Visualizer requires the following files:

- A **data file** consisting of rows and tab-separated fields. This file is easily created using the Tool Manager (see Chapter 3). If you are generating this file yourself, see Appendix A, “Creating Data and Configuration Files for the Tree Visualizer” for the required file format.

  Data files are generated by extracting data from a source (such as an Oracle, INFORMIX, or Sybase database) and formatting it specifically for use by the Tree Visualizer. Data files have user-defined extensions (the sample files provided with the Tree Visualizer have a *.data* extension).

- A **configuration file** describing the format of the input data and how these are converted to a hierarchy. This file also is easily created using the Tools Manager (see Chapter 3). You also can use an editor (such as jot, vi, or Emacs) to produce this file (see Appendix A, “Creating Data and Configuration Files for the Tree Visualizer”).

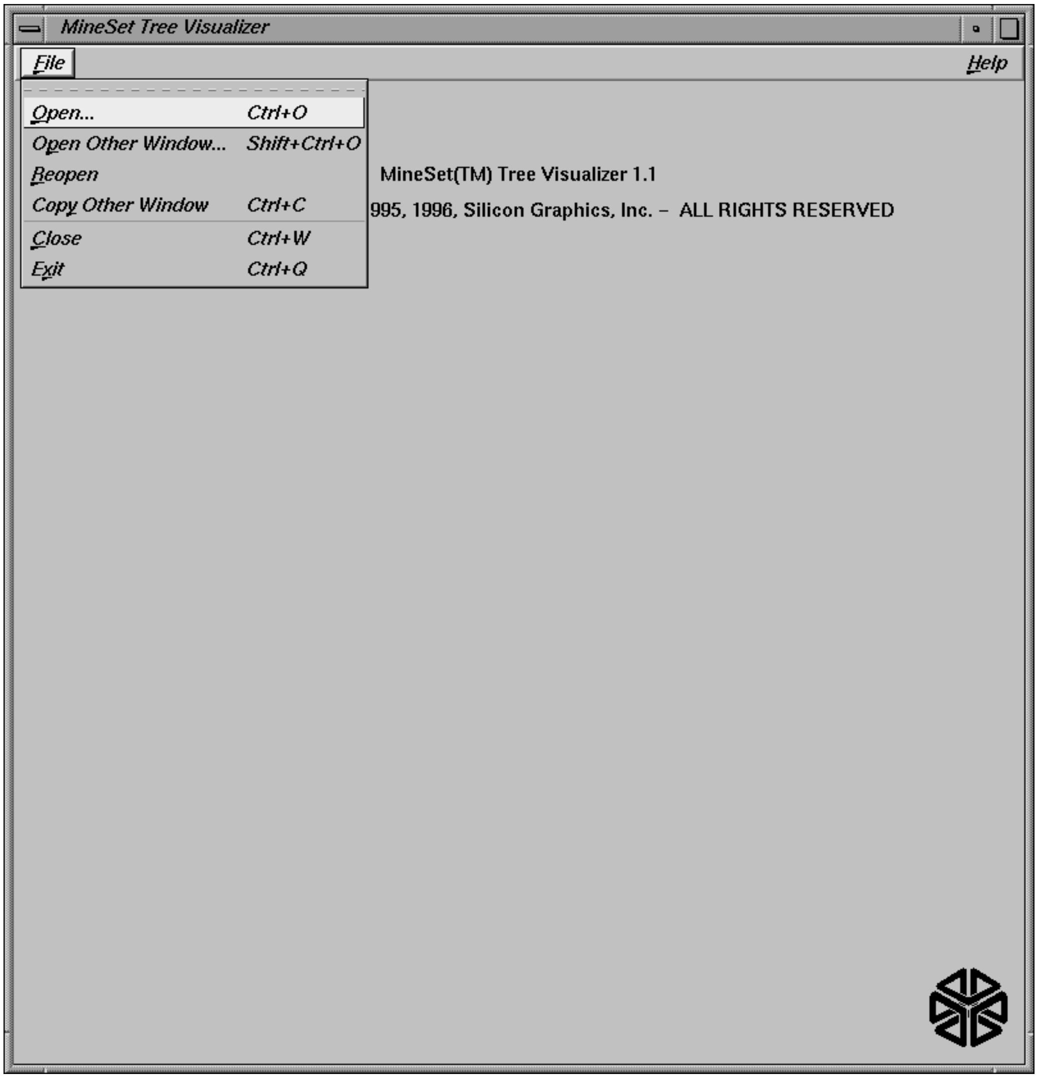
  Configuration files must have a *.treeviz* extension. When starting the Tree Visualizer, or when opening a file, specify the configuration file, not the data file.

Starting the Tree Visualizer

There are five ways to start the Tree Visualizer:

- Use the Tool Manager to configure and start the Tree Visualizer. (See Chapter 3 first for details on most of the Tool Manager’s functionality, which is common to all MineSet tools; see below for details about using the Tool Manager in conjunction with the Tree Visualizer.)
- Double-click the Tree Visualizer icon, which is in the MineSet page of the icon catalog. The icon is labeled *treeviz*. Since no configuration file is specified, the start-up screen requires you to select one by using File > Open.

Starting the Tree Visualizer without specifying a configuration file causes the main window to show the copyright notice for this tool. Only the File and Help pulldown menus can be used. For the main window to be fully functional, open a configuration file by selecting File > Open (Figure 4-2).



MineSet(TM) Tree Visualizer 1.1

995, 1996, Silicon Graphics, Inc. – ALL RIGHTS RESERVED

Figure 4-2 Tree Visualizer’s Startup Screen, File Pulldown Menu Selected

- If you know what configuration file you want to use, double-click the icon for that file. This starts the Tree Visualizer and automatically loads the file you specified. This only works if the filename ends in *.treeviz* (which is always the case for configuration files created for the Tree Visualizer via the Tool Manager).
- Drag the configuration file icon onto the Tree Visualizer icon. This starts the Tree Visualizer and automatically loads the file you specified. This works even if the filename does not end in *.treeviz*.
- Start the Tree Visualizer from the UNIX shell command line by entering this command at the prompt:

  ```
  treeviz [ configFile ]
  ```

  where *configFile* is optional and specifies the name of the configuration file to use. If you don't specify a configuration file, you must use File > Open to specify one (see Figure 4-2).

Configuring the Tree Visualizer Using the Tool Manager

This section describes how the Tree Visualizer can be configured using the Tool Manager. Although the Tool Manager greatly simplifies the task of configuring the Tree Visualizer, you can construct a configuration file manually for this tool using an editor (see Appendix A, "Creating Data and Configuration Files for the Tree Visualizer").

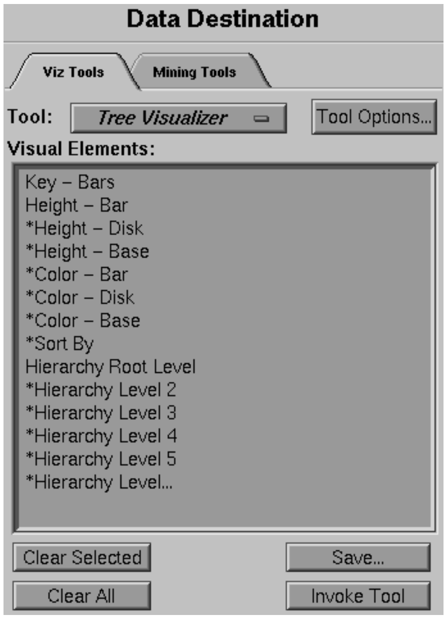
For the Tree Visualizer, the Tool Manager does not support the following:

- Non-aggregated hierarchies where the data is displayed directly without aggregating it.
- The execute command.
- Anumber of very rarely used options (skip missing, overview, shrinkage, root label, speed, climb speed, leaf leaf margin, root leaf margin, leaf edge margin, initial position, initial angle, bar label size, base label size, and lod). See Appendix A.
- Variable-length arrays.

Note that the steps required to connect to a data source are described in Chapter 3.

Selecting the Tree Visualizer Tool

Select the *Viz Tools* tab in the Data Destination panel of the Tool Manager's main screen (Figure 4-3). From the popup list of tools, select Tree Visualizer. The mapping requirements for the Tree Visualizer are displayed in the window on the right side of this panel. Items in the Visual Elements: list that are preceded by an asterisk are optional.



Tool: *Tree Visualizer*

Visual Elements:

Key – Bars
Height – Bar
*Height – Disk
*Height – Base
*Color – Bar
*Color – Disk
*Color – Base
*Sort By
Hierarchy Root Level
*Hierarchy Level 2
*Hierarchy Level 3
*Hierarchy Level 4
*Hierarchy Level 5
*Hierarchy Level...

Figure 4-3 Data Destination Panel of Tool Manager With Tree Visualizer Selected

Key - Bars— Lets you define what the bars show in the Tree Visualizer main window represent. For example, in a table representing the budget of the 50 United States, the keys could be state names. If the first key is associated with Alabama, the first bar represents the values for Alabama.

Height - Bar— Lets you specify what the bar heights represent. Typically, the higher the bar, the greater the value represented.

Sort By— Lets you specify a column, the values of which are used to sort the layout of the nodes. The sort order defaults to ascending from left to right.

Hierarchy Root Level—Lets you specify how the table from your data source is converted into a hierarchy. The Requirements: list defaults to six hierarchical levels. If you specify a sixth hierarchy level, the Tree Visualizer automatically adds a seventh. With every next level you specify, the Tree Visualizer adds another one. You can specify as many hierarchy levels as necessary.

Height - Disk—Lets you specify what the heights represent for optional disks placed at the same location as the bar. If no mapping is specified, no disks are displayed.

Height - Base—Lets you specify what the base heights represent. If no mapping is specified, the bar height mapping is used.

Color - Bar—Lets you specify what the bar colors represent. The specific colors must be assigned via the Tool Manager’s Tool Options panel (see “Color Options for the MineSet Visualizers” in Chapter 3).

Color - Disk—Lets you specify what the disk colors represent. This option has an effect only if the disk height is specified (see “Color Options for the MineSet Visualizers” in Chapter 3).

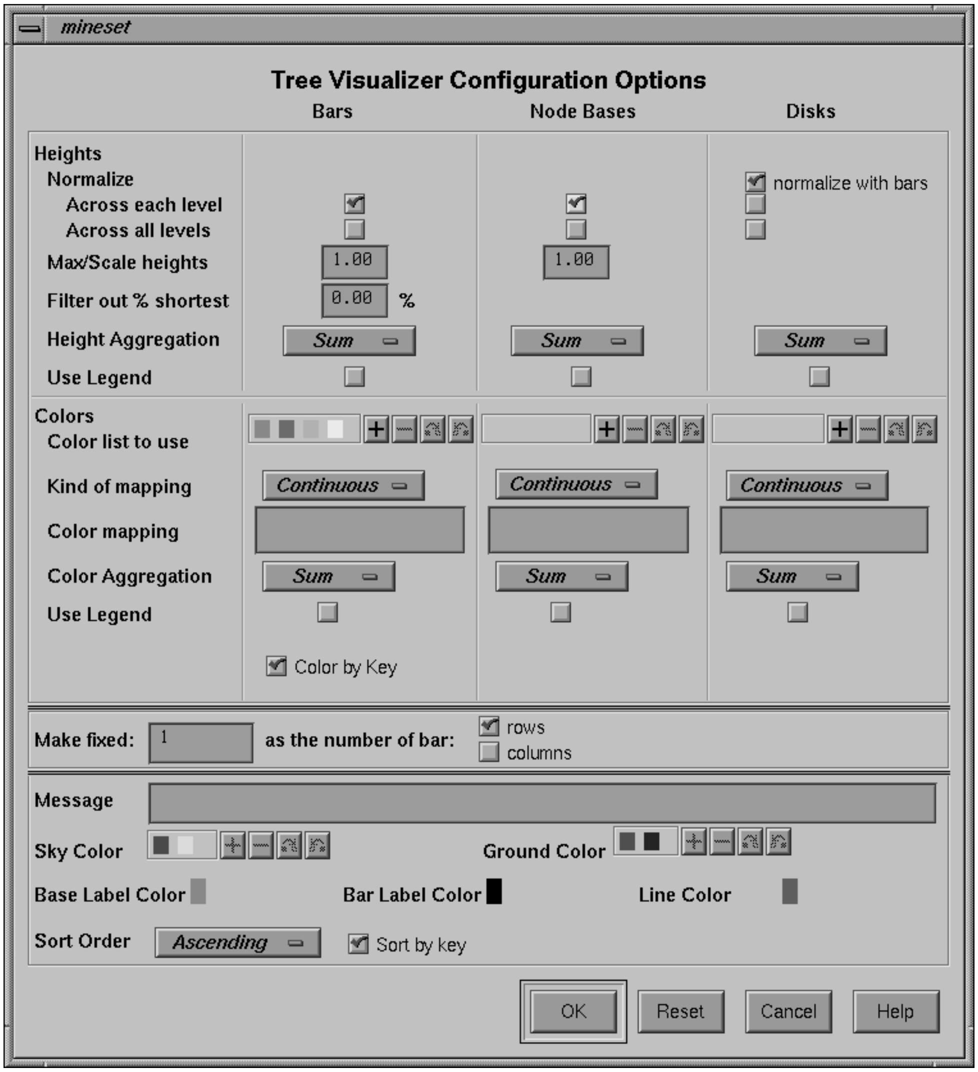
Color - Base—Lets you specify what the base colors represent. If no mapping is specified, the bar color mapping is used (see “Color Options for the MineSet Visualizers” in Chapter 3).

Undoing Mappings

To undo any mapping, select that mapping in the Requirements: window, then click the *Clear Selected* button. To undo all mappings, click the *Clear All* button.

Specifying Tool Options

Clicking the *Tool Options* button causes a new dialog box to be displayed (Figure 4-4). This lets you change some of the Tree Visualizer options from their default values.



Tree Visualizer Configuration Options

	Bars	Node Bases	Disks
Heights			
Normalize			☑ normalize with bars
Across each level	☑	☑	□
Across all levels	□	□	□
Max/Scale heights	1.00	1.00	
Filter out % shortest	0.00 %		
Height Aggregation	Sum	Sum	Sum
Use Legend	□	□	□
Colors			
Color list to use			
Kind of mapping	Continuous	Continuous	Continuous
Color mapping			
Color Aggregation	Sum	Sum	Sum
Use Legend	□	□	□
	☑ Color by Key		

Make fixed: 1 as the number of bar: ☑ rows □ columns

Message

Sky Color Ground Color

Base Label Color Bar Label Color Line Color

Sort Order Ascending ☑ Sort by key

Figure 4-4 Tree Visualizer's Configuration Options Dialog Box

The top of the dialog box has three columns: *Bars*, *Node Bases*, and *Disks*.

Normalize Heights

This option lets you normalize heights across each level of the hierarchy (or across all levels) of bars, node bases, and disks. Normalizing the heights determines the maximum value of the height variable; it normalizes all values relative to that height. Thus, if the maximum value is 30.0, and the maximum bar height was set to 1.0 (in arbitrary units), a value of 15.0 would be mapped to a value of 0.5.

Normalizing across each level independently normalizes each level of the hierarchy. This option is most useful if data has been summed up the hierarchy, and prevents the top level of the hierarchy from dwarfing items at the lowest level. Normalizing across all levels normalizes everything together, regardless of the level in the hierarchy. If neither box is checked for bars, no normalization takes place.

Node Bases are normalized independently of Bars. If no boxes are checked, the same normalization method used for bars is used for node bases, although the values are normalized independently.

If disks are present and *normalize with bars* is checked, the disks are normalized in conjunction with the bars: a disk and a bar representing the same value have the same height. If one of the other normalize boxes is checked in the Disks column, disks are normalized independently of the bars: the highest disk and the tallest bar have the same height, regardless of the actual values represented by them.

Max/Scale Heights

This option lets you specify the height of the tallest bars and node bases. The default is 1.0 (in arbitrary units). If after looking at the view, you see that the heights are too low or too high, use this field to adjust them. For example, entering `2` in the field causes all bars to be doubled in height; entering `.5` makes all bars half as big.

If normalization was specified, this value represents the height of the tallest bar or base. If normalization was not specified, all values are scaled by this amount. The latter can be useful when comparing views of two different datasets.

Filter Out % Shortest

This option lets you filter out nodes containing only short bars. First, the tallest bar in the scene is calculated (if heights are normalized by level, then the tallest bar in each level). Then only those nodes that contain at least one bar that is the appropriate percentage of the tallest bar are shown. For example, if you enter 5% in this field, then only those nodes containing at least one bar that is at least 5% of the height of the tallest bar are shown. (Also shown are ancestors of such bars). This option is intended as a coarse way to filter out small, uninteresting nodes. It is not intended as an exact mechanism of identifying specific nodes of a certain value. Use of this option can accelerate the rendering of slow, complex scenes, or reduce clutter resulting from many bars near zero height.

Although small nodes are filtered out, they are nonetheless counted in any cumulation up the hierarchy.

Height Aggregation

By default, the height of the bars of the parent node is the sum of the height of all the bars of the children; however, these heights can be average, max, min, count, or any. This aggregation can be used for the values of the bar heights, base heights, and disk heights.

Colors

This set of options lets you

- specify the list of colors to use
- specify the kind of mapping
- map colors to bars, node bases, and disks

To use these Colors options, you must have mapped a column to the *Color - Bar, *Color - Disk, or *Color - Base requirements of the Data Destination panel. See “Color Options for the MineSet Visualizers” in Chapter 3 for a more detailed explanation of how to choose and change colors.

Color list to use—You can specify the color list using the + button next to the color list label. This brings up a color editor that lets you specify a color to be added to the list.

Kind of mapping—You can specify whether the color change that is shown in the graphic display is *Continuous* or *Discrete*. If you choose Continuous, the color values (of the bars, node bases, or disks) shift gradually between the colors entered in the Color list to use field as a function of the values that are mapped to those colors in the *Color mapping* field. If you choose Discrete, the colors change only at the specified boundaries.

Color mapping—These fields let you specify a value to which the colors are mapped.

Example 1:

If you

- used the Color Browser to apply red and green to bars
- selected Discrete for the Kind of mapping
- entered the values `0 100`

then the display shows all bars (or node bases or disks) with values of less than 100 in red, and all those with values greater than or equal to 100 in green.

Example 2:

If you

- used the Color Browser to apply red and green to bars
- selected Continuous for the Kind of mapping
- entered the values `0 100`

then the display shows all bars (or node bases or disks) with values less than or equal to 0 as completely red, those as greater than or equal to 100 as completely green, and those between 0 and 100 as shadings from red to green.

Color Aggregation

By default, the values of the color of the bars of the parent node are the sum of the values of all the bars of the children; however, these colors can be average, max, min, or any. This aggregation can be used for the values of the bar colors, base node colors, and disk colors.

Color by Key

This option lets you automatically color the bars by their key value. This option is ignored if another coloring was specified. If you specify no color list, or specify insufficient colors, additional colors are chosen at random. If extra colors are specified, they are ignored.

Make Fixed

By default, this option places all bars across one row. This option allows changing the number of rows or columns. If neither rows nor columns are selected, or the number is set to 0, then neither rows nor columns are fixed, and the closest approximation to a square is displayed.

Message

This option lets you type in any message you want. The message statement specifies the message displayed when the pointer is moved over an object or when an object is selected. By default, the same message is used for the base as for the bars. If no message is specified, a default message containing the names and values of all the columns is used.

The format of the message must match the type of data being used:

- Strings must use %s.
- Ints must use integer formats (like %d).
- Floats and doubles must used floating-point formats (like %f).

For a detailed description of the message field, see “Message Statements” in Appendix A.

Sky Color

You can specify either one or two colors. If only one color is specified, the sky is solid. If two colors are specified, the sky is shaded between the colors. When specifying two colors, the first color is for the top of the sky, the second for the bottom.

Ground Color

You can specify either one or two colors. If only one color is specified, the ground is solid. If two colors are specified, the ground is shaded between the colors. For the ground, the first color is for the far horizon, the second is for the near ground.

Base Label Color

You can specify the color of the labels on the front of the bases.

Bar Label Color

You can specify the color of the labels on the front of the bars.

Line Color

You can specify the color of the lines connecting the bases.

Sort Order

If you select the *Sort by Key* checkbox, the nodes in the display are in sorted order. The menu next to the checkbox lets you specify whether to sort in ascending or descending order.

Resetting the Tool Options

If, after you have made changes to the Tool Options dialog box, you want to reset the values of all options to their default values, click the *Reset Options* button.

Saving the New Tool Options

Once you have finished making changes to the Tool Options dialog box, click *OK* to return to the Tool Manager's main screen.

To have the changes you made to any part of the Tool Options dialog box take effect, you must save the configuration file again (by clicking the *Save Config File* button, described below). Note that if you give the name of a previously saved configuration file, the new file overwrites the saved file. After you have saved your new configuration file, click the *Invoke Tool* button again to see the results of your changes.

Saving Tree Visualizer Settings

The Tool Manager stores information for the Tree Visualizer in several files, all sharing the same prefix:

- *<prefix>.treeviz.data* contains data.
- *<prefix>.treeviz.schema* describes the data file.
- *<prefix>.treeviz* contains information needed by the Tree Visualizer.
- *<prefix>.mineset* contains all the information needed to create the other files.

To specify a prefix, use the Save ... menu option in the File menu of the Tool Manager's main window. If you do not specify a prefix, the default *untitled* is used.

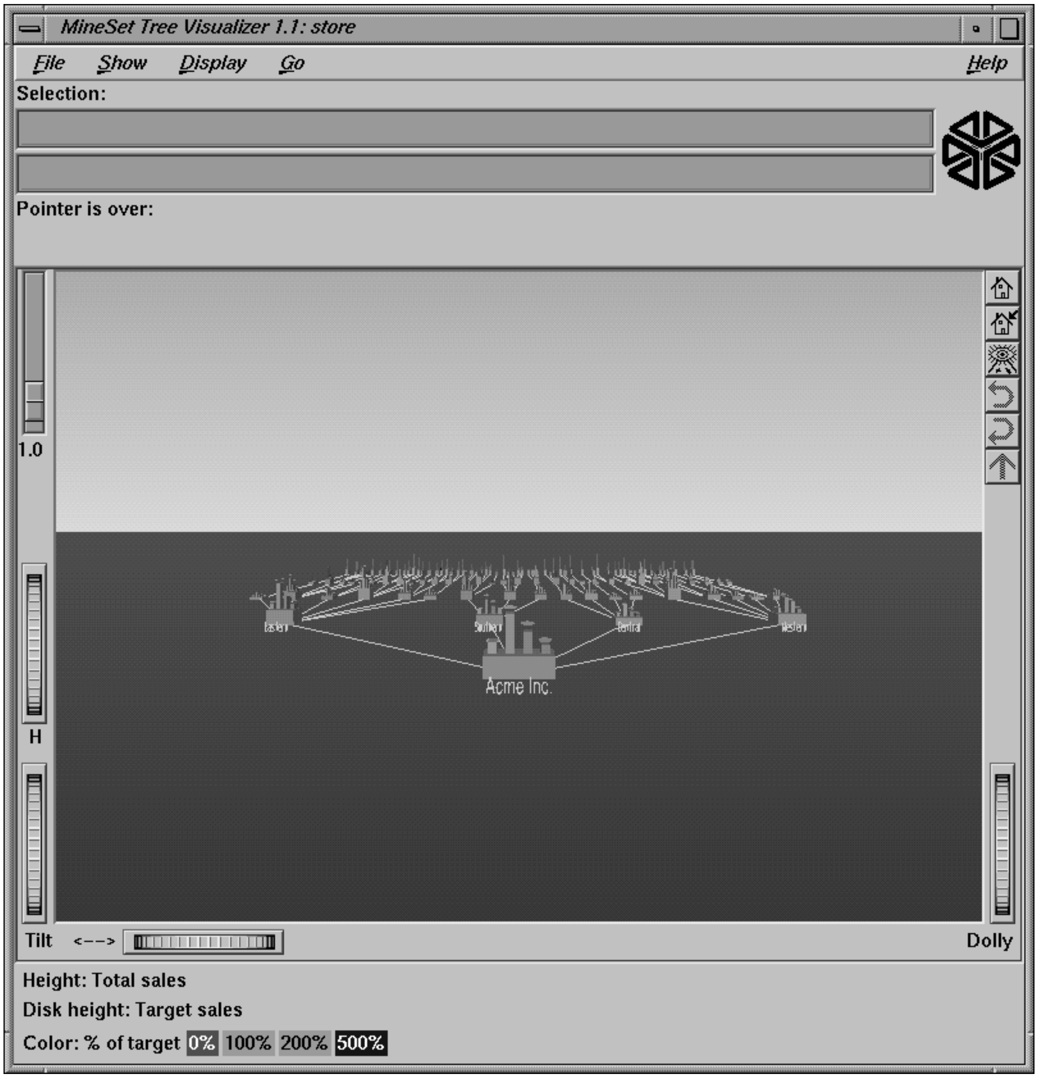
When you use the *Invoke Tool* button, the *.data*, *.schema*, and *.treeviz* files are updated, if necessary.

Invoking the Tree Visualizer

To see the Tree Visualizer graphically represent your data, click the *Invoke Tool* button at the bottom of the Data Destination panel.

Working in the Tree Visualizer’s Main Window

A file’s hierarchy is visible only after a valid configuration file is specified. For example, specifying *store.treeviz* results in Figure 4-5.



Selection:

Pointer is over:

Height: Total sales

Disk height: Target sales

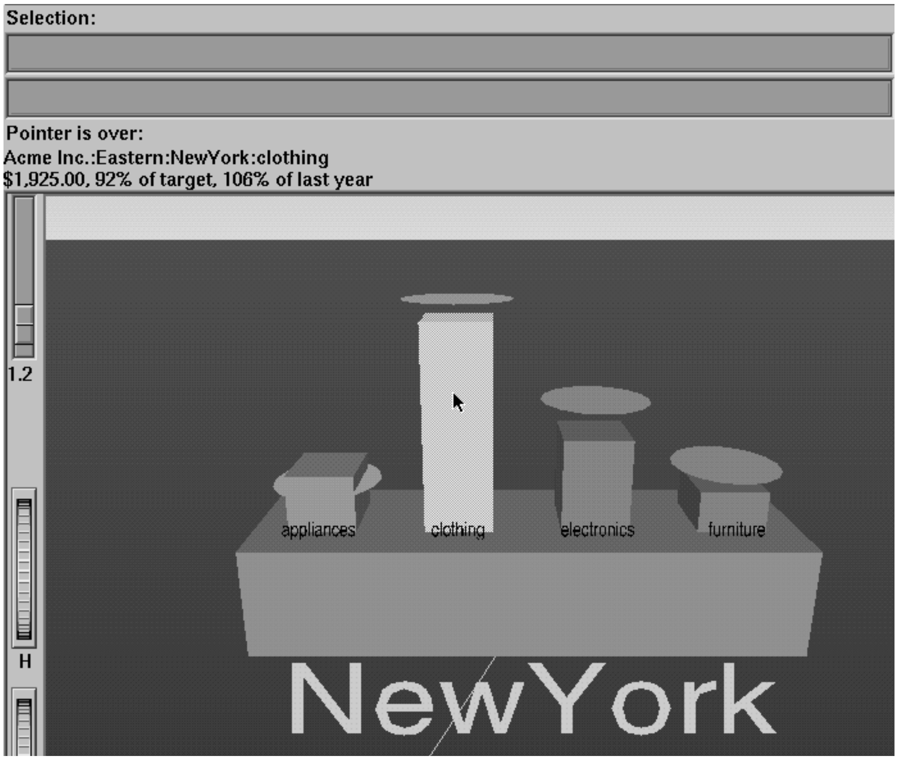
Color: % of target 0% 100% 200% 500%

Figure 4-5 Tree Visualizer’s Initial View When Specifying store.treeviz

The root node of the hierarchy is at the front of the scene, near the bottom of the Tree Visualizer’s main window. In back of the root node are its descendents; each one consists of a base with bars on it. You can change what the heights and colors of the bars represent by specifying it in the Tool Manager or manually changing the *.treeviz* configuration file; usually, the base represents the aggregate of all the bars. Bases are connected with lines representing the connection of the nodes to their descendents.

Highlighting an Object or Node

To highlight an object, move the mouse over that object (either a base or a bar). This causes information about that object to appear over the top left of the view area, under the “Pointer is over:” label (Figure 4-6). To highlight a node and obtain information about that node, place the pointer over a line leading to that node. This information appears in the same place as that for an object.



Acme Inc.:Eastern:NewYork:clothing
$1,925.00, 92% of target, 106% of last year

Figure 4-6 A Highlighted Object and the Information It Represents

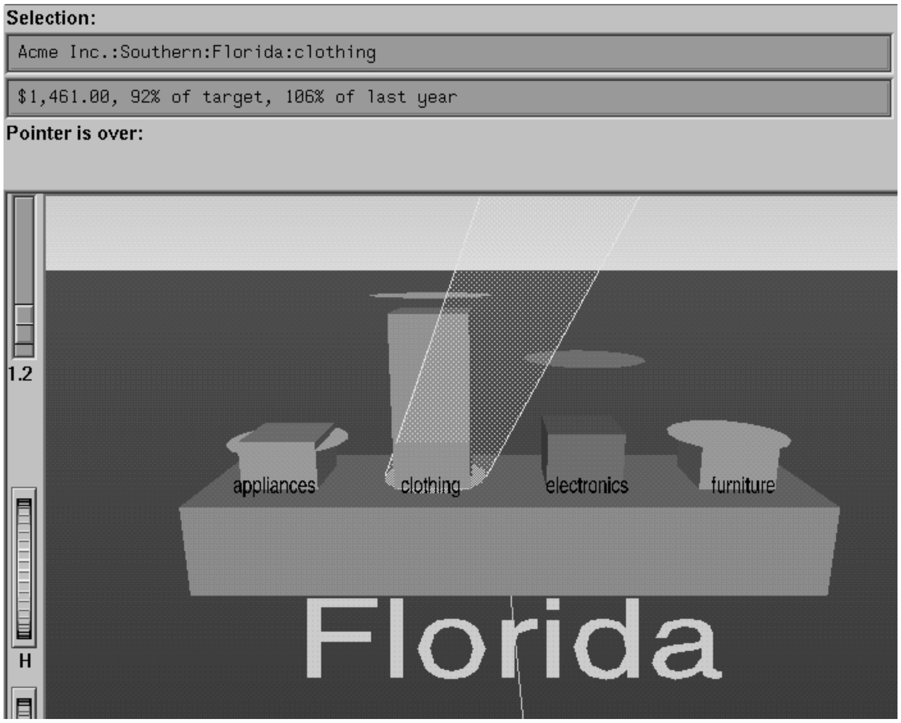
Selecting an Object

To select an object and zoom to it, left-click the mouse on that object. Hold the Shift key down while clicking to select the object without zooming to it. At the top of the window, under the label "Selection:", you see information about a selected object. The information is the same as that shown when highlighting an object. As long as the object is selected, the information is displayed. This lets you compare information about two objects by selecting one, then highlighting the other. Using the mouse, you can cut and paste selection information into other applications, such as reports or databases. Note that when you select a bar, you might not see the label of that bar's base.

Spotlighting an Object

When you select an object, a white spotlight appears on it (Figure 4-7). A yellow spotlight appears when you are searching (see “The Search Panel” on page 98). A spotlight’s point of origin is above the root node; thus, when it is shining on columns or bases to the left or right of the root node, the beam is angled accordingly. Spotlights are visible even if the selected object is a descendent node in the far background.

The edges of spotlights are surrogates for an object: when you move the pointer over the edge of a spotlight, the associated object is highlighted, and information about that object appears above the top left of the view. Left-click the edge of a spotlight to select the associated object and (if the Shift key is not held down) to zoom to it. The spotlight is active only on the solid lines along the edges, not the translucent section in the center. This allows selection of objects behind the spotlight.



Selection:

Acme Inc.:Southern:Florida:clothing

$1,461.00, 92% of target, 106% of last year

Pointer is over:

Figure 4-7 Example of a Selected (Spotlighted) Object

Navigating With the Middle Mouse Button

To navigate over the scene in the main window, use the middle mouse button. You also can use external controls to perform all middle mouse button functions (see the “External Controls” on page 91).

To move through the main window, click the middle mouse button. A small square appears (see Figure 4-8). Move the cursor out of this square while pressing the mouse to move your point of reference dynamically through the 3D landscape. The farther the cursor is from the square, the faster your viewpoint moves. To move the viewpoint forward, move the mouse up. To move the viewpoint back, move the mouse down. Moving the mouse left and right causes the viewpoint to shift accordingly. You can move in any direction as long as a part of your data is visible.

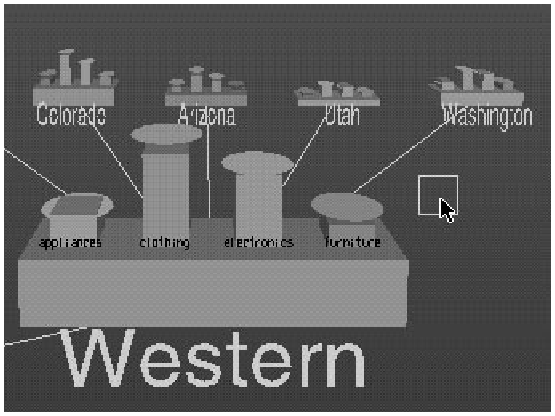


Figure 4-8 Example of the Square as Navigational Base

To move the viewpoint up and down, hold the Shift key down when pressing the middle mouse button. To move the viewpoint up, move the mouse up. To move the viewpoint down, move the mouse down. You cannot move below ground level.

To combine horizontal and vertical motion (that is, to move the viewpoint back and forth, as well as up and down), hold the Alt key down when pressing the middle mouse button. Note that while moving forward, the viewpoint also moves down, based on the current tilt. Similarly, while moving backward, the viewpoint moves up, based on the tilt.

Note: You cannot turn from side to side. Tilting the viewpoint requires using external controls.

External Controls

Several external controls surround the graphics window. These consist of buttons and thumbwheels.

Buttons

At the top right of the image area are six buttons as shown in Figure 4-9.

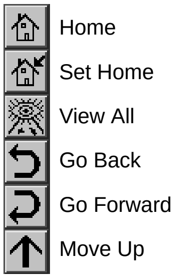


Figure 4-9 Tree Visualizer's External Button Controls

- *Home* takes you to a designated location. Initially, this location is the first viewpoint shown after invoking the Tree Visualizer and specifying a configuration file. If you have been working with the Tree Visualizer and have clicked the *Set Home* button, then clicking *Home* returns you to the viewpoint that was current when you last clicked *Set Home*.

- *Set Home* makes your current location the Home location. Clicking the *Home* button returns you to the last location where you clicked *Set Home*.
- *View All* lets you view the whole hierarchy, keeping the tilt of the camera. To get an overhead view of the scene, tilt the camera to point straight down, then click the *View All* button. To tilt the camera, see the description of the Tilt thumbwheel (see “Thumbwheels” on page 93).
- *Go Back* lets you return to the previous location. If you have just started the Tree Visualizer and have not moved from the home view, this button is grayed out.
- *Go Forward* lets you proceed to the location from which you clicked the *Go Back* button. If you have not clicked the *Go Back* button, the *Go Forward* button is grayed out.
- *Move Up* is active only when you have an object selected. If a bar is selected, clicking this button selects the base containing the bar. If a base is selected, clicking this button moves up the hierarchy to the parent node. Once the root node has been reached (highest level of the hierarchy), the *Move Up* button is grayed out. Note that when using *Move Up*, the selected node is changed to the parent of the previously selected one.

You also can perform these functions using the Go menu (see “The Go Menu” on page 109.)

Thumbwheels

Four thumbwheels appear around the lower part of the graphics window border (see Figure 4-10). They let you dynamically move the viewpoint.

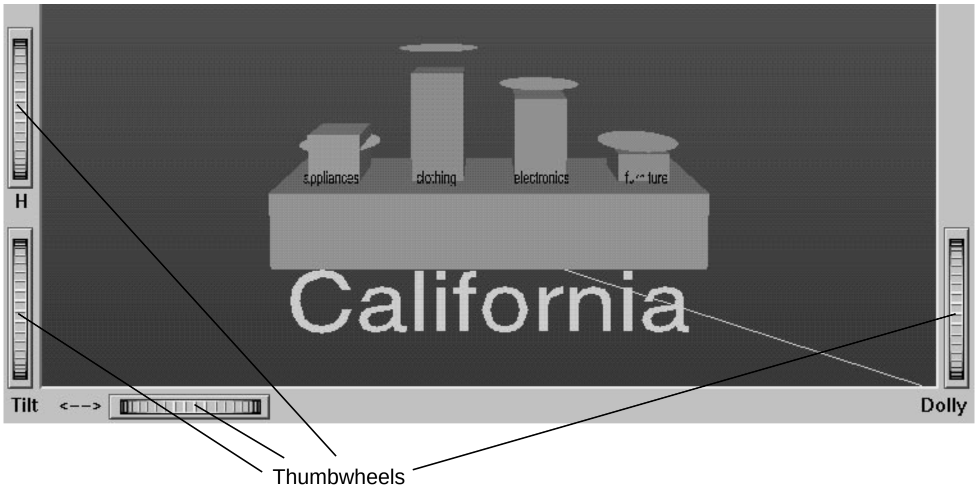


Figure 4-10 Tree Visualizer's Thumbwheels

- The vertical H (height) thumbwheel, on the left, moves the camera up and down. You cannot move the viewpoint below ground level.
- The vertical Tilt thumbwheel, at the bottom left, tilts the camera. You can tilt the viewpoint to any position from straight ahead and straight down. You cannot tilt the viewpoint to look up.
- The horizontal <--> (pan) thumbwheel, at the bottom left, moves the viewpoint from left to right and back. You cannot rotate the viewpoint.
- The vertical Dolly thumbwheel, on the right, moves the viewpoint forward and backward.

Height Slider

A slider to the top left of the main window (Figure 4-11) lets you rescale all objects in the window. Pushing the slider up to a value of 2.0 doubles the size of all objects in the main window. Pulling the slider back down to a value of 1.0 returns the objects in the window to their original heights.

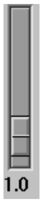


Figure 4-11 Tree Visualizer's Height Slider

Pulldown Menus

You also can access all of the Tree Visualizer's functions via five pulldown menus. These are labeled File, Show, Display, Go, and Help.

If you start the Tree Visualizer without specifying a configuration file, only the File and the Help menus are available. The Show, Display, and Go menus are available after a graph is loaded.

The File Menu

The File menu (Figure 4-12) contains six options. The options are described below.

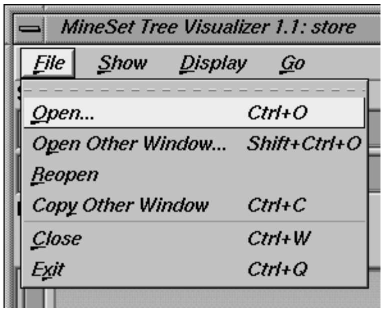


Figure 4-12 Tree Visualizer's File Pulldown Menu With Options

- Open loads and opens a configuration file, displaying it in the main window. Previously displayed data is discarded. Use *Open* to view a new dataset, or to view the same dataset after changing its configuration.
- Open Other Window opens a configuration file, but displays its results in a different window. The current dataset remains open.
- Reopen reopens the currently opened file. This can be used after the configuration or data file has been updated.
- Copy Other Window opens a new window that displays the same view of the current dataset. You can interact with these windows independently.
- Close closes the current window (and all panels associated with it). If no other windows are open, *Close* exits the application.
- Exit closes all windows and exits the application.

The Show Menu

The Show menu (Figure 4-13) contains four options:

- Overview
- Search Panel
- Filter Panel
- Marks

Each of these options brings up another dialog box for interacting with the data.

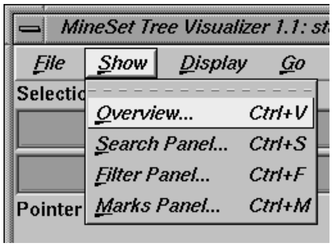


Figure 4-13 Tree Visualizer's Show Pulldown Menu With Options

The Overview Window

Select Overview in the Show menu to bring up a new window with an overhead view of the complete hierarchy (Figure 4-14). If you want the Overview to be brought up automatically each time the scene is viewed, set the Overview option in the configuration file (see "Overview" on page 366).

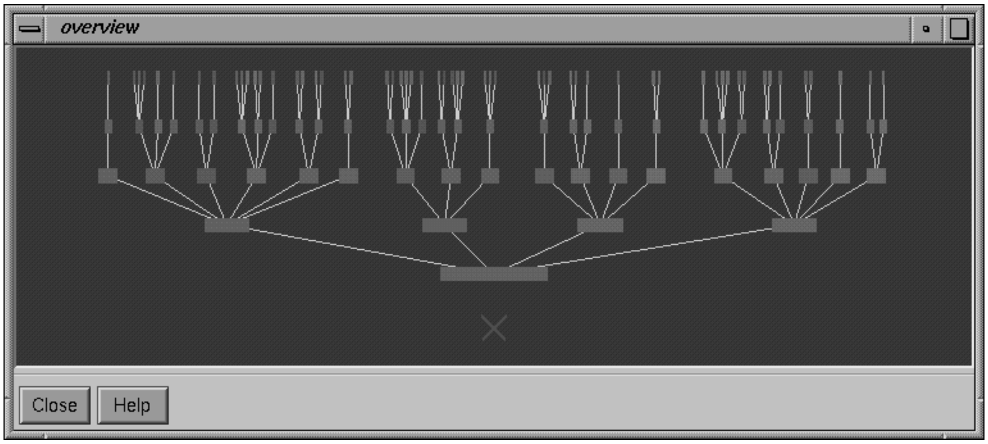


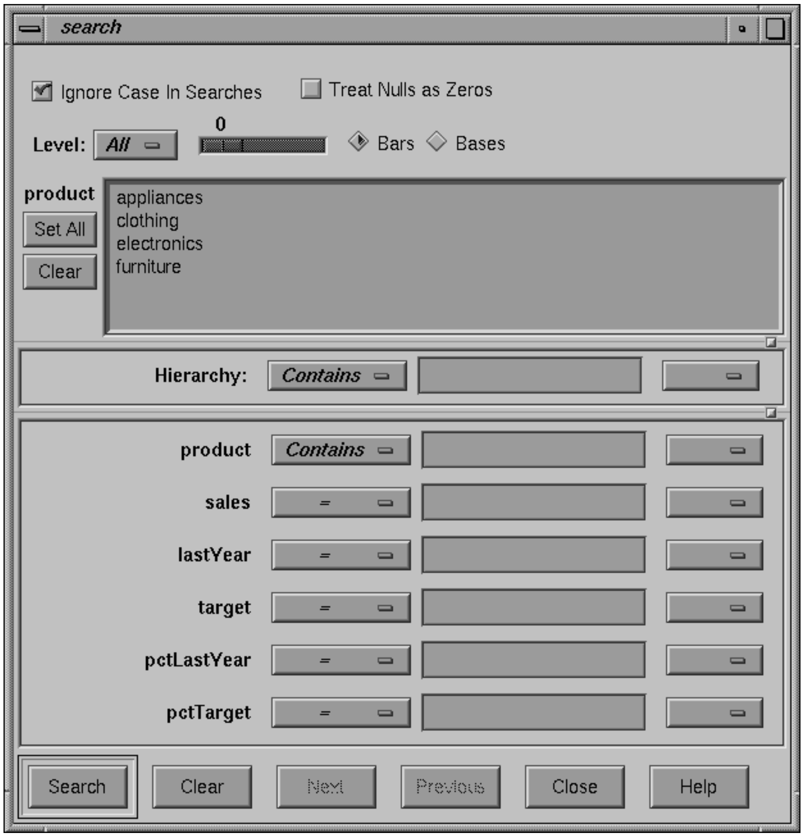
Figure 4-14 Tree Visualizer’s Overview Window

The “X” in the Overview window shows your current location. The Overview helps you keep track of your location and viewpoint in the entire scene. It can also help you quickly go to a specific node.

To select an object in the Overview and have the main view zoom to it, left-click that object. This is similar to left-clicking the object in the main view. Middle-clicking anywhere in the overview zooms your viewpoint to that location, even if no object is at that point.

The Search Panel

Select *Search* in the Show menu to bring up a dialog box that lets you specify criteria to search for objects (Figure 4-15).



☑ Ignore Case In Searches

□ Treat Nulls as Zeros

Level: All

0

◆ Bars ◇ Bases

product

appliances
clothing
electronics
furniture

Hierarchy: Contains

product Contains

sales =

lastYear =

target =

pctLastYear =

pctTarget =

Figure 4-15 Tree Visualizer's Search Dialog Box

Once the search is complete, yellow spotlights highlight objects matching the search criteria (see Figure 4-16). To display information about an object under a yellow spotlight, move the pointer over that spotlight; the information appears in the upper left corner, under the label "Pointer is over." To select and zoom to an object under a yellow spotlight, left-click the spotlight; if you press the Shift key while clicking, zooming does not occur.

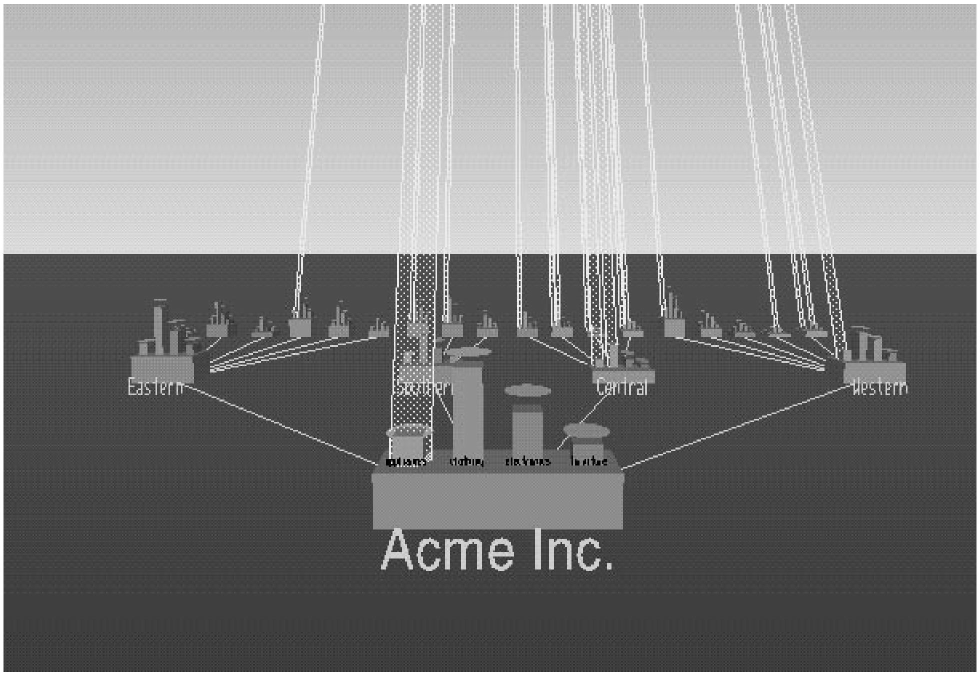


Figure 4-16 Sample Results of a Search in the Tree Visualizer

Items in the Search Panel

To specify whether a search is case-sensitive, click the *Ignore Case In Searches* checkbox, at the top of the Search panel. For example, if this toggle is on (a check mark appears on that button), the string “hello” is the same as “HellO.”

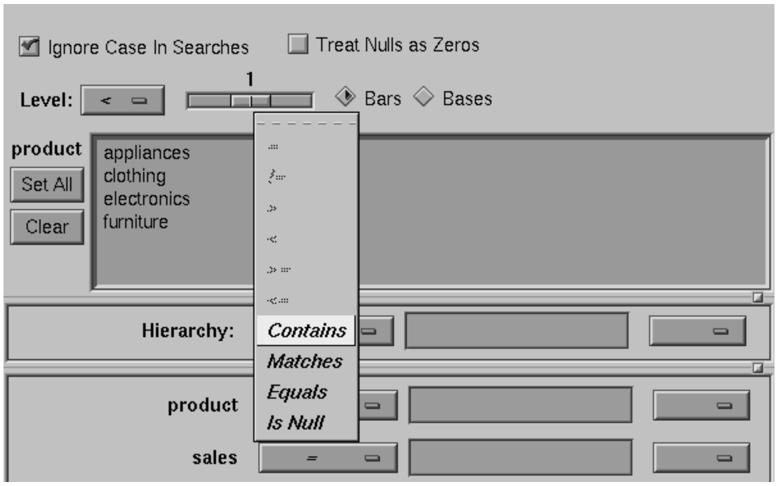
To the right of the case sensitivity checkbox is another, labeled *Treat Nulls as Zeros*. If this checkbox is off (the default), comparisons involving nulls cannot return TRUE in a search. If the it is on, nulls are treated as equal to zero.

Below the case-sensitivity checkbox are controls that let you specify the parts of the hierarchy to be searched. By default, the whole hierarchy is searched. To limit the levels searched, select a relational operator (such as <=) from the option menu that lets you specify the operand for the level. Then use the slider to select the level to be searched. Level 0 is the root of the hierarchy, level 1 is the level below that, and so forth. To search the root and the two levels below that, for example, choose <= 2.

Checkboxes also let you choose whether to search the bars or the bases.

When searching through bars, the default is that all bars are searched. To search only a specific list of bars, you must select them. The *Set All* button turns on all bars; this is useful if most of the bars are to be searched, and only a few are to be turned off. The *Clear* button turns off all bars. If no bar is selected, the bar list is ignored, and all bars are searched.

Below the panel for bar labels is a Hierarchy field that lets you specify nodes to search (Figure 4-17). Below the Hierarchy field are fields that let you specify search criteria for individual columns (defined in the Current Columns: window of the Tool Manager’s Table Processing pane, see “Selecting the Tree Visualizer Tool” on page 76).



☑ Ignore Case In Searches □ Treat Nulls as Zeros

Level: < 1 ◆ Bars ◇ Bases

Figure 4-17 Detail of the Tree Visualizer’s Search Dialog Box

To search for numeric values, enter the value, and select a relational operation (=, !=, >, <, >=, <=). To search for alphanumeric values, enter the string for which you want to search. You can use any of three types of string comparisons:

- “Contains” indicates that it contains the appropriate string. For example, California contains the strings Cal and forn.
- “Equals” requires the strings to match exactly.
- “Matches” allows wildcards:
  - An asterisk (*) represents any number of characters.
  - A question mark (?) represents one character.
  - Square braces ([ ]) enclose a list of characters to match.

  For example, California matches Cal*, Cal?fornia, and Cal[a-z]fornia.

In some cases (usually associated with binning in the Tool Manager), an option menu of values appears, instead of a text field. To ignore that variable, select Ignored in the Option menu. You can use relational operators (such as >=) with these options. This means that the specified value as well as subsequent ones are selected.

In addition to numeric and string comparison operations, you can specify `Is Null`, which will be true if the value is null.

To the right of each search field is an additional option menu that lets you specify "And" or "Or" options. For example, you could specify "sales > 20 And < 40." You can have any number of And or Or clauses for a given column, but cannot mix And and Or in a single column.

Note that if different levels of the hierarchy are keyed by different types of data (for example, the top level is selected by strings, while the second level is selected by integers), then the "Hierarchy" search field is treated as a string and provides string operations, not number operations.

If the *Ignore Case In Searches* checkbox is checked, the comparisons of all string searches are case-insensitive.

Six buttons are placed across the bottom of the Search panel:

- *Search* causes the search to be started. This button is automatically activated if the Enter key is pressed and the panel is active.
- *Clear* turns off all search spotlights and erases the values from the search fields.
- *Next* selects and zooms to the next matched object, in left-to-right order. After the last matched object is selected, clicking *Next* returns the view to the Home position. *Next* is valid only after a search that has found matches.
- *Previous* selects and zooms in the opposite order from that of the *Next* button.
- *Close* closes the search window and turns off the search spotlights. If the Search panel is reopened, it is in the same state as it was before the last *Close*; clicking *Search* again repeats the last search.

The Filter Panel

The Filter panel filters out selected information, thus fine-tuning the displayed hierarchy. You can use the Filter panel to emphasize specific information, or to shrink the amount of data for better performance.

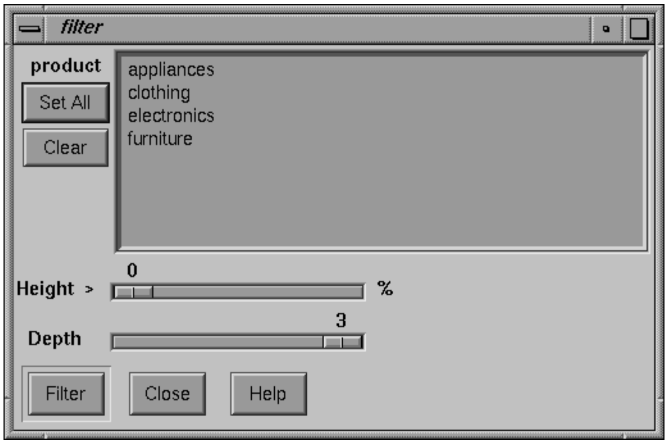
The buttons at the top left of the panel (Figure 4-18) let you filter the information to be displayed based on specific bars. Select the bars to be filtered by clicking them. The *Set All* button turns on all bars; this is useful if most of the bars are to be searched, and only a few are to be turned off. The *Clear* button turns off all bars. If no bar is selected, the bar list is ignored, and nothing is filtered.

Filtering bars does not affect the information in the base, which continues to include the summary of all bars.

Below the bar label window of the Filter panel is the Height Filter slider. This slider lets you filter out those nodes containing only short bars. The size of a value is shown as a percentage of the maximum height. First, the tallest bar in the scene is calculated (if heights are normalized by level, then the tallest bar in each level). Then only those nodes that contain at least one bar that is the appropriate percentage of the tallest bar are shown.

For example, if you enter 5% in this field, then only those nodes containing at least on bar that is at least 5% of the height of the tallest bar are shown. (Also shown are ancestors of such bars). This option is intended as a coarse way to filter out small, uninteresting nodes. It is not intended as an exact mechanism of identifying specific nodes of a certain value; the search panel should be used for that purpose. Use of this option can accelerate the rendering of slow, complex scenes, or reduce clutter resulting from many bars near zero height. You can also set this filtering option in the configuration file by using the Height Filter command.

Although small nodes are filtered out, they are nonetheless counted in any cumulation up the hierarchy.



appliances
clothing
electronics
furniture

Height > 0 %

Depth 3

Figure 4-18 Tree Visualizer’s Filter Dialog Box

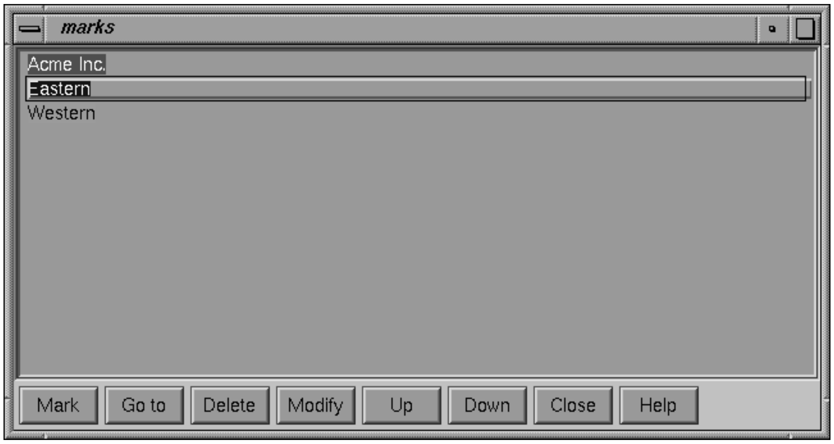
The Depth slider, which is under the Height Filter slider, lets you display the hierarchy so that only a given number of levels are displayed at any given time. When you are at the top of the hierarchy, only the number of hierarchical levels specified by the slider is seen. The nodes in the rows are arranged to optimize their visibility. When navigating to nodes lower in the hierarchy, additional rows are made visible automatically. The nodes above them automatically adjust their locations to accommodate the newly added nodes; thus, some nodes might seem to move. Note that the overview shows all nodes in the hierarchy, not just the top nodes; thus, the layout of the overview might not match the layout of the main view. The X in the overview approximates the corresponding location in the main view; there is no exact mapping between the two layouts.

- Click the *Filter* button to start filtering. If the *Enter* key is pressed while the panel is active, filtering automatically starts.
- Click the *Close* button to close the panel.

The Marks Panel

The Marks panel lets you name and store important locations (viewpoints) so that you can easily and quickly return to them (see Figure 4-19). The location is stored relative to the currently selected object. If no object is selected, the absolute location is recorded.

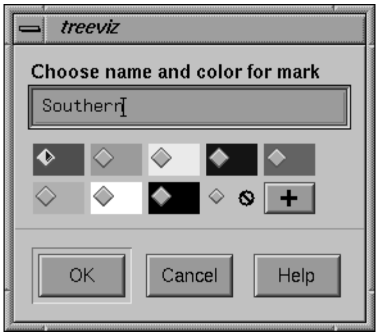
All marks can be indicated by colored flags in the main view. If the mark represents a selected object, the flag is placed on that object. If it represents an absolute position, the flag is placed at that position. To go to the mark, click the flag. All flags can be turned on and off using the Mark Flags menu entry in the Display menu. (See *Mark Flags* in “The Display Menu” on page 108).



Acme Inc.
Eastern
Western

Figure 4-19 Tree Visualizer's Marks Panel

- Click the *Mark* button to mark the current location. Another dialog box appears (in Figure 4-20) to prompt you for the name and color of the mark. The default name is that of the currently selected object. The color controls the color of the flag appearing in the main window and represents the mark. If you do not want a flag to represent the mark, click the button with the "Not" symbol (slash through a circle). To add another color to the palette, click the button with the plus symbol (+) to bring up a color chooser.



Choose name and color for mark

Southern

Figure 4-20 Window Resulting From Clicking Mark Button

Figure 4-21 shows a sample main window with flags representing the created marks.

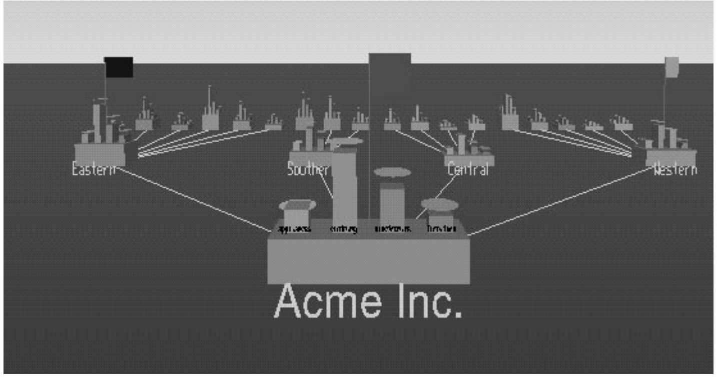


Figure 4-21 Main Window With Flags Representing Marks

- Click the *Go to* button to go to the current location associated with the selected mark in the panel. Double-clicking a mark has the same effect. If the object selected by that mark no longer exists (because it was filtered out, or the data was changed since the mark was created), the location shown is close to where the object would have been.
- Click the *Delete* button to delete the selected mark in the panel.
- Click the *Modify* button to change the name or color of the selected mark in the panel.
- Click the *Up* button to move the selected mark in the panel up the listing order.
- Click the *Down* button to move the selected mark in the panel down the listing order.
- Click the *Close* button to exit the marks panel.

The file storing the marks information has the same name as the configuration file, with a *.marks* suffix appended. Whenever a mark is changed, all marks are saved to that file. If all marks are deleted, the *.marks* file is removed. If mark changes cannot be saved (because of a permission error, for instance), a warning appears; this warning is not repeated when subsequent mark changes are attempted.

The Display Menu

The Display menu lets you control several display parameters.

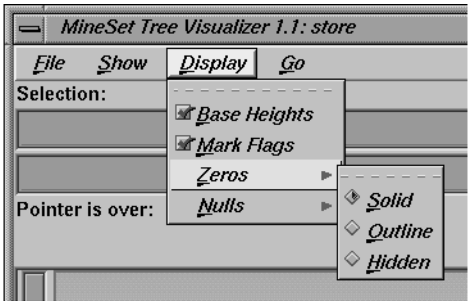


Figure 4-22 Tree Visualizer's Display Menu

Base Heights is a checkbox that lets you turn the heights of the bases on and off. To see negative numbers, or to make it easier to compare the bar heights, turn this option off. Turning it on provides summary information about all the bars. The initial value of this toggle can be changed with the "base height" statement in the configuration file.

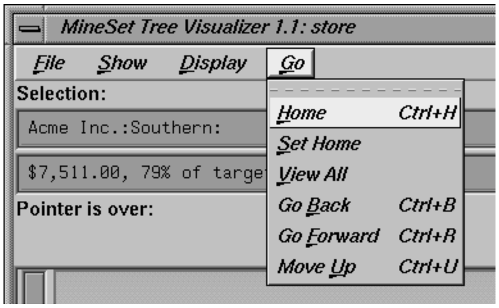
Mark Flags is a toggle option that lets you turn on or off the flags representing marks (also see "The Marks Panel").

Zeros is a submenu that controls how objects with zero height are displayed. By default, they are shown like other objects: a solid cube of height zero (a plane). The submenu lets you specify them to be displayed as outlines (appearing as a hollow square), or to be hidden completely (not drawn). The initial value of this of this can be changed using the "zero" option in the configuration file (see "Zero" in Appendix A).

Nulls is a submenu that controls how objects of null height are displayed. It has the same options as the zero menu; however, the default for null options is to display the objects as an outline. The initial value can be changed using the "null" option in the configuration file (see "Null" in Appendix A).

The Go Menu

The Go menu duplicates the functions of the buttons on the upper right-hand side of the main window (see Figure 4-23). It also identifies keyboard shortcuts for some functions.



Acme Inc.:Southern:

$7,511.00, 79% of targe

Figure 4-23 Tree Visualizer's Go Pulldown Menu

- Home takes you to a designated location. By default, this location is the initial view point of the scene. Initially, this location is the first viewpoint shown after invoking the Tree Visualizer and specifying a configuration file. If you have been working with the Tree Visualizer and have clicked the *Set Home* menu item, then clicking *Home* returns you to the viewpoint that was current when you last clicked *Set Home*. The keyboard shortcut for this function is Ctrl+H.
- Set Home changes the Home location to your current location. Clicking the *Home* menu item then returns you to the viewpoint that was current when you last clicked *Set Home*.
- View All shows the whole hierarchy, keeping the tilt of the camera. To get an overhead view of the scene, tilt the camera to point straight down, then click the *View All* menu item. (To tilt the camera, see the description of the Tilt thumbwheel in "Thumbwheels" on page 93.)
- Go Back lets you return to the previous location. If you have just started the Tree Visualizer and have not moved from the home view, this menu item is grayed out. The keyboard shortcut for this function is Ctrl+B.
- Go Forward lets you proceed to the location from which you clicked the *Go Back* menu item. If you have not clicked the *Go Back* menu item, the *Go Forward* menu item is grayed out. The keyboard shortcut for this function is Ctrl+R.

- Move Up is active only when an object is selected. If a bar is selected, clicking this menu item selects the base containing the bar. If a base is selected, clicking this menu item moves up the hierarchy to the parent node. Once the root node has been reached (highest level of the hierarchy), the *Move Up* menu is grayed out. The keyboard shortcut for this function is Ctrl+U.

The Help Menu

The Help menu (see Figure 4-24) provides access to five help functions.

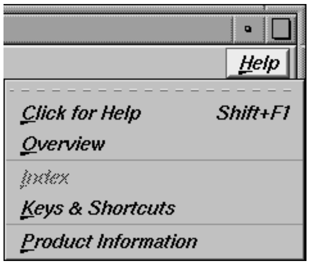


Figure 4-24 Tree Visualizer’s Help Pulldown Menu

- Click for Help turns the cursor into a question mark. Placing this cursor over an object in the main window and clicking the mouse causes a help screen to appear; this screen contains information about that object. Closing the help window restores the cursor to its arrow form and deselects the help function. The keyboard shortcut for this function is Shift+F1. (Note that it also is possible to place the arrow cursor over an object and press the F1 function key to access a help screen about that object.)
- Overview provides a brief summary of the major functions of this tool, including how to open a file and how to interact with the resulting view.
- Index provides an index of the complete help system. This option is currently disabled.
- Keys & Shortcuts provides the keyboard shortcuts for all of the Tree Visualizer’s functions that have accelerator keys.

- Product Information brings up a screen with the version number and copyright notice for the Tree Visualizer.
- *MineSet User's Guide* invokes the IRIS Insight viewer with the online version of this manual.

Null Handling in the Tree Visualizer

Nulls represent unknown data (see Appendix G, "Nulls in MineSet").

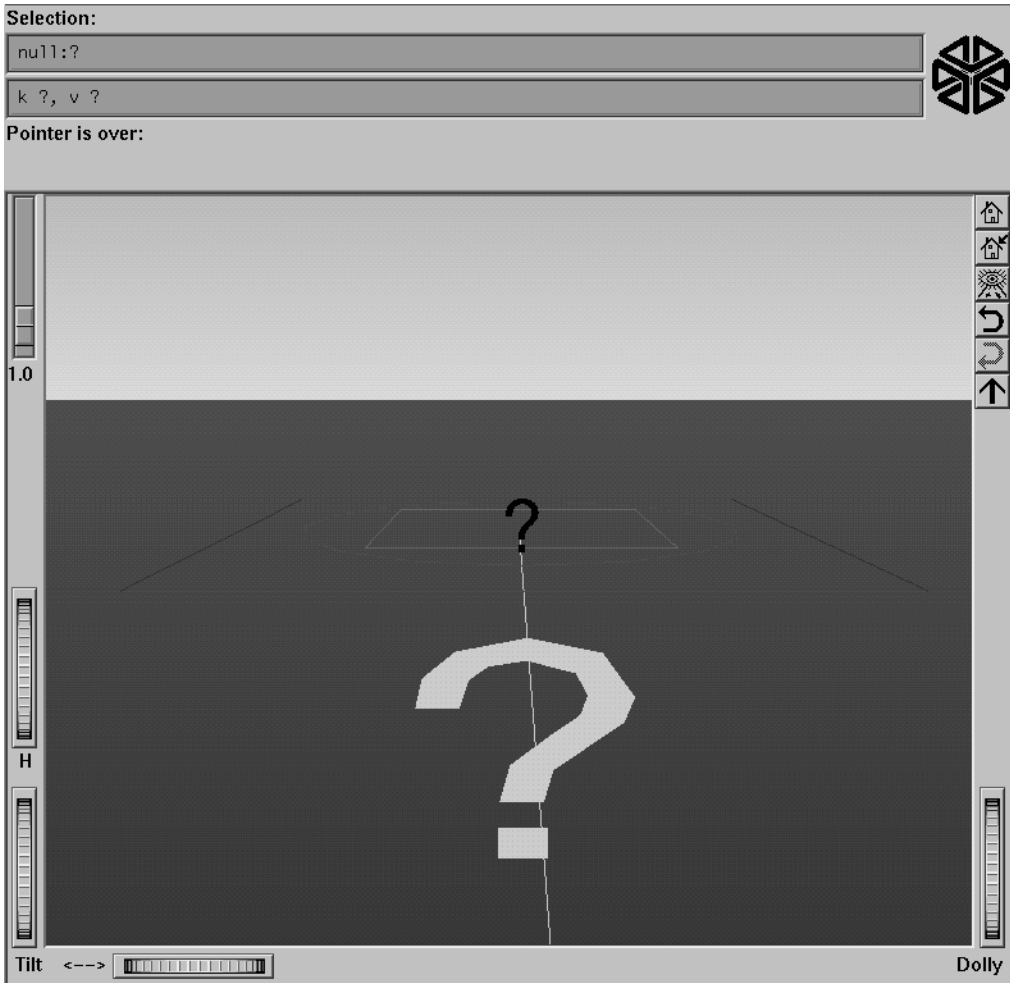
In the Tree Visualizer, nulls can occur in the following cases:

- The database or data file contains a null value.
- The skipMissing option is not present in the configuration file (see skipMissing in Appendix A), and data is present for the key value in one node of the hierarchy, but not in another. For example, in a representation of state budgets, if there is no record for state income tax for Texas, Texas would have an income tax of null. This is different for the case where there is a record showing 0 as the income tax for Texas, in which case it would show a tax of 0.
- When the Tool Manager is used to make an array based on bins and no data falls into a specific bin, the value for that bin is null. For example, if there is no data for 30-40 year olds, that bin is null.
- When making an array in the Tool Manager and the null enum option is specified, an extra array entry, corresponding to the first bar in each bar chart, is created to represent the aggregation of all the values where the bin value is null (see "Bins and Arrays With Nulls" in Appendix G). This bar is labeled with a question mark (?), representing null. If there is no data for that null bin, the values associated with it are null as well.

  Note: if all values throughout the data associated with the null bin are null, the Tree Visualizer ignores the null bin and does not display it.
- Expressions and aggregations of nulls can generate nulls (see Appendix G).

When a null value is mapped to a visual attribute, special representations are used in the Tree Visualizer. If null is mapped to height, the object is normally drawn in outline mode (although this is configurable through the Display menu (see the "The Display Menu" section) or the configuration file (see

"Null" in Appendix A). For a bar or a base, this looks like an empty square. (It does not look like a cube, since it has no height.) For a disk, it looks like a circle. If a null value is mapped to a color, it is drawn in a dark grey (see Figure 4-25).



Selection:

```
null:?
k ?, v ?
```

Pointer is over:

Figure 4-25 Representation of a Null Value Mapped to Height, Color, Disk, and Label

When selecting an object with a null value, it is shown as a question mark (?) in the selection field.

Sample Configuration and Data Files

The provided sample configuration and data files demonstrate the Tree Visualizer's features and capabilities. The following files are in the directory */usr/lib/MineSet/treeviz/examples*:

- *store.data* and **sto***re.treevi***z**
  When graphically displayed, these files show hypothetical sales data for a store chain. The hierarchy includes the entire chain, regions, states, cities, and individual stores. Four products are shown for each level in the hierarchy. In this configuration, heights represent sales in dollars; colors represent the percentage of the target dollar amount.
- *stateRevenue.data* and *stateRevenue.treeviz*
  When graphically displayed, these files show the revenue components of the every state's budgets for 1992, as obtained from the United States Census Bureau (from http://www.census.gov/govs/state/stfin92.dat). Heights represent the dollar amounts in taxes. The descendent nodes in the background show the contribution of various taxes to the total revenues shown in the root node.
- *beer.data* and *beer2.data*, and *beer.treeviz* and *beer2.treeviz*
  When graphically displayed, these files show fictitious data based on consumer research of beer purchases. The hierarchy contains three levels:

  1. The first is category (for example, beer or ale).
  2. The second level is brand codes (randomly assigned).
  3. The third is the individual product codes; for example, twelve-pack versus six-pack (randomly assigned).

  Each chart contains seven bars, representing seven age groups. Bar height represents the total dollars spent by that age group. Colors represent the percentage of dollars spent by males and females. Brands, products, and data used in these files are samples only.

Both *beer.treeviz* and *beer2.treeviz* produce the same graphical output, but they have been constructed differently. In *beer.treeviz*, each type of beer is represented by a single record, with values for males and for female consumption; these values are stored in an enumerated array (explained in Appendix A, “Creating Data and Configuration Files for the Tree Visualizer”).

In *beer2.treeviz*, there are seven records for each beer, with each record representing one age group. Note that in the *beer* file, the age groups are represented in the configuration file; in the *beer2* file, they are included in the data file.

The *beer* file requires less storage space than the *beer2* file; however, the configuration file is a little more complicated. In some cases, it might be easier to produce data in the form used by the *beer2* file.

Additional examples of the Tree Visualizer to visualize a Decision tree are provided in Chapter 9.

Chapter 5

Using the Map Visualizer

This chapter discusses the features and capabilities of the Map Visualizer. It provides an overview of this database visualization tool, then explains the Map Visualizer's functionality when working with the follwing elements:

- main window
- viewing modes
- external controls
- pulldown menus

Finally, it lists and describes the sample files provided for this tool.

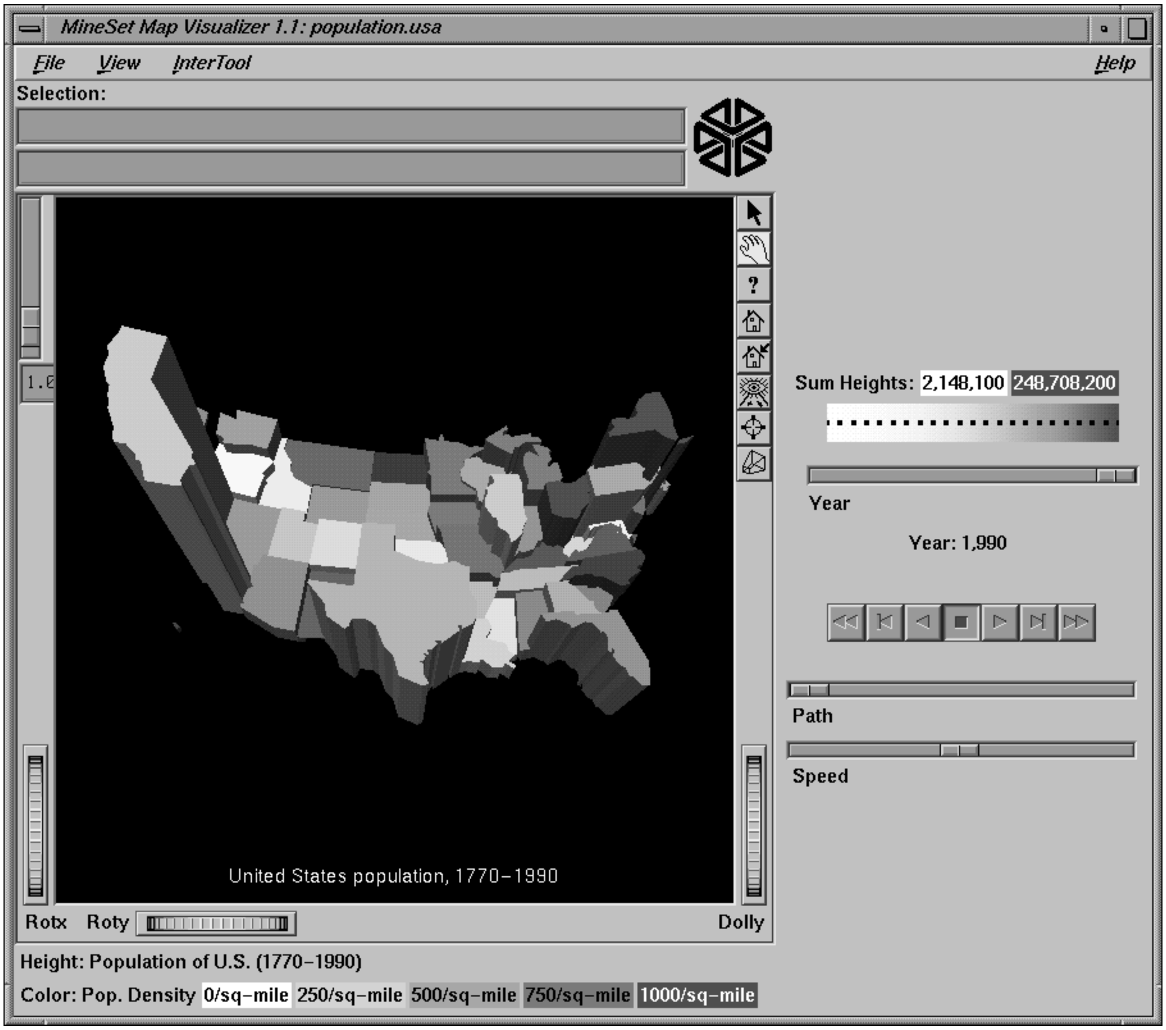
Overview of Map Visualizer

The Map Visualizer is a graphical interface that displays data as a three-dimensional "landscape" of arbitrarily specified and positioned "bar chart" shapes. This tool displays quantitative and relational characteristics of your geographically oriented data.

Data items are associated with graphical "bar chart" objects in the visual landscape. However, the objects have recognizable geographical shapes and positions. The landscape can consist of a collection of these geographical objects, each with individual heights and colors (see Figure 5-1). You can dynamically navigate through this landscape by

- panning
- rotating
- zooming to more clearly see areas of interest
- drilling down to see increased granularity of geographic details

- drilling up to aggregate data into coarser-grained graphical objects
- using animation to see how the data changes across one or two independent dimensions.


MineSet Map Visualizer 1.1: population.usa
File
View
InterTool
Help
Selection:
1.0
United States population, 1770–1990
Rotx
Roty
Dolly
Year
Path
Speed

Sum Heights: 2,148,100 248,708,200

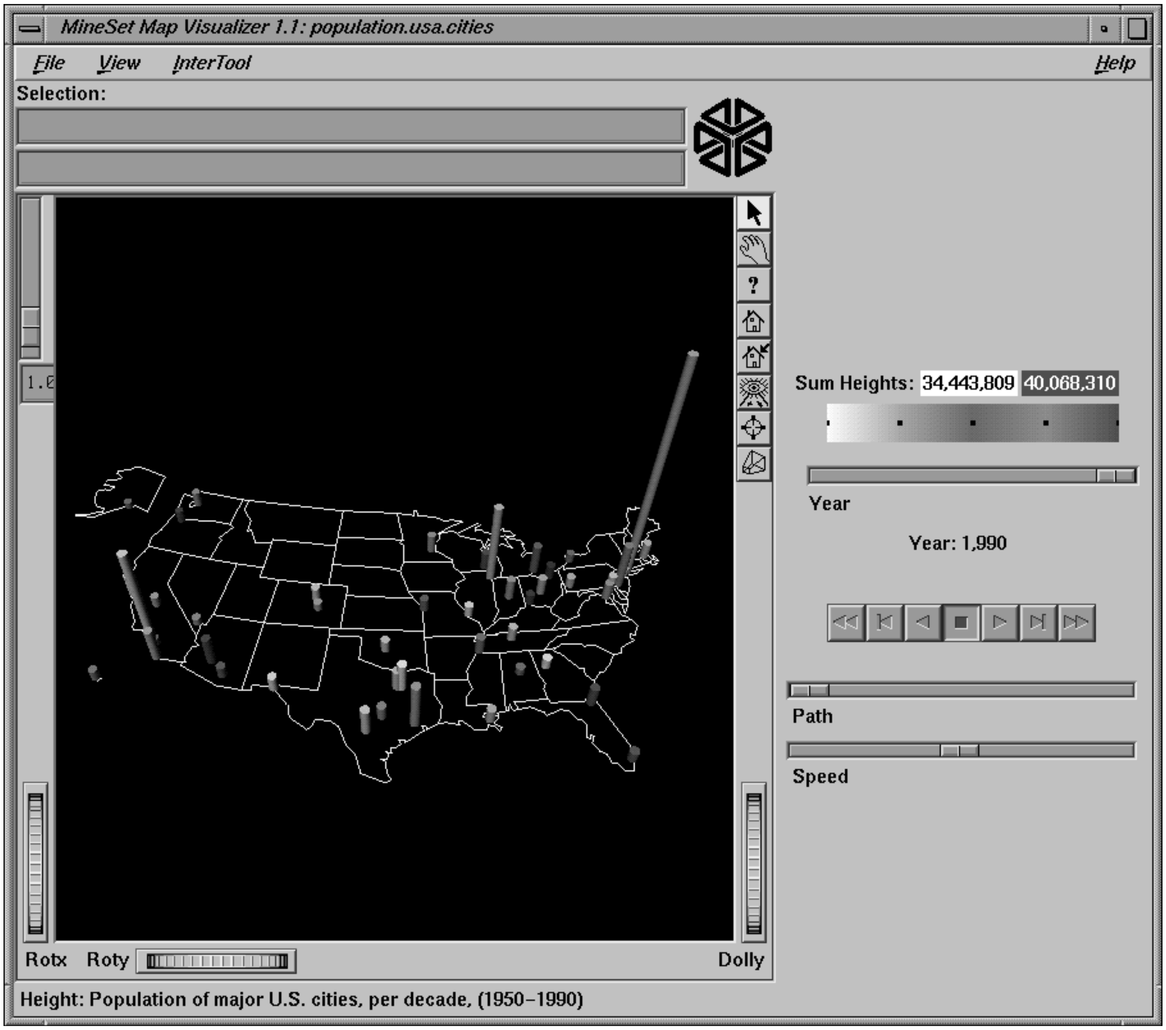
Year: 1,990

Height: Population of U.S. (1770–1990)

Color: Pop. Density 0/sq–mile 250/sq–mile 500/sq–mile 750/sq–mile 1000/sq–mile

Figure 5-1 Sample Map Visualizer Screen Showing 1990 U.S. Population

The landscape can also consist of a flat plane of these geographical objects drawn as simple outlines, with “bar chart” cylinders placed at specific locations (see Figure 5-2).



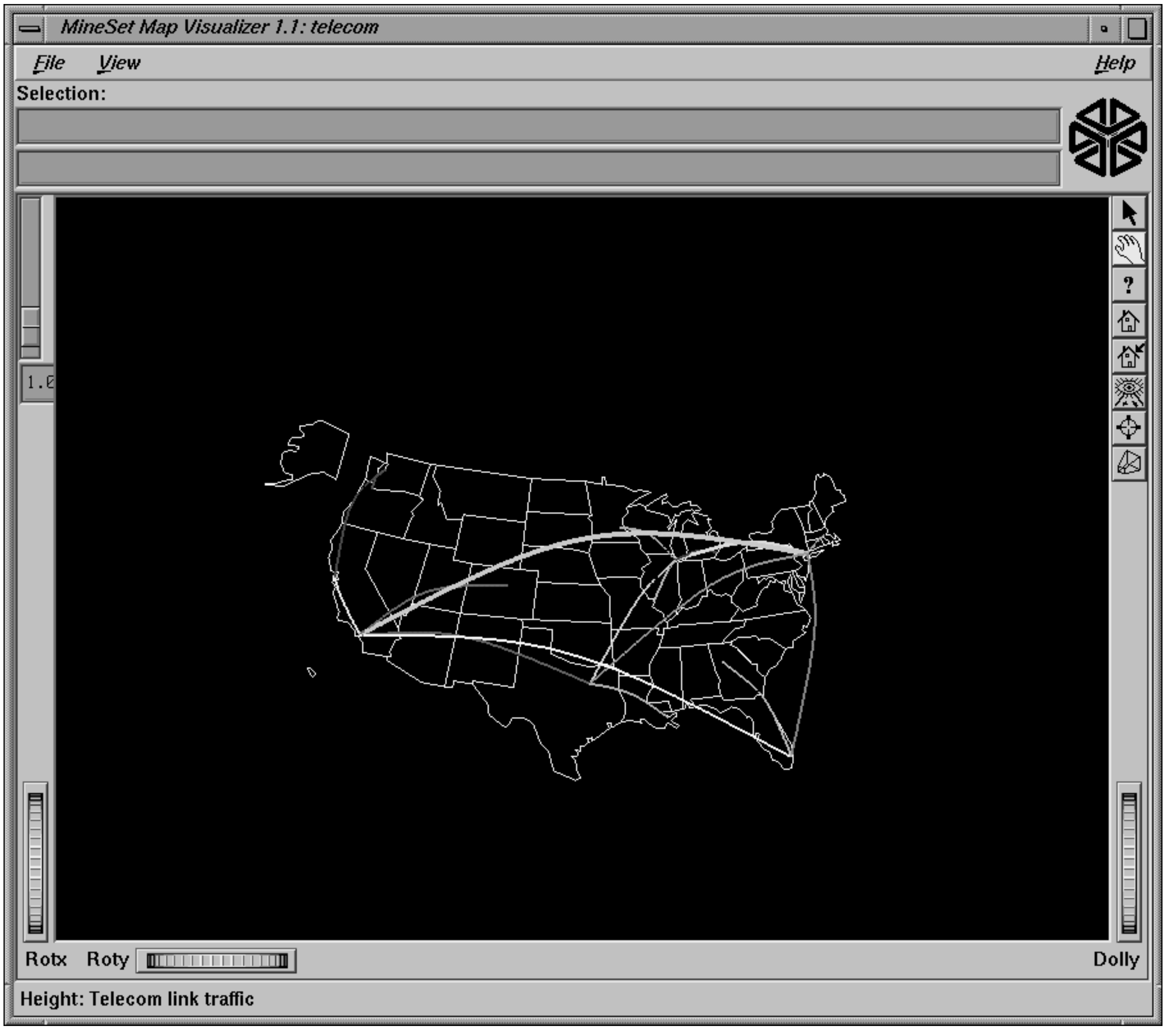
Sum Heights: 34,443,809 40,068,310

Year: 1,990

Height: Population of major U.S. cities, per decade, (1950–1990)

Figure 5-2 Sample Map Visualizer Screen Showing Relative Population of Major U.S. Cities

Another landscape possibility is lines with endpoints at specific point locations, all with individual widths and colors (see Figure 5-3). Lines have width and color properties, instead of the height and color properties of the arbitrarily shaped objects and cylinders.



Height: Telecom link traffic

Figure 5-3 Sample Map Visualizer Screen Showing the United States With Specific Endpoints

File Requirements

The Map Visualizer requires the following files:

- A data file consisting of rows and tab-separated fields. Typically, the Tool Manager creates this file (see Chapter 3). You can also generate this file without using the Tool Manager (for the required file format, see Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer”).

  Data files are the result of extracting raw data from a source (such as an Oracle, INFORMIX, or Sybase database) and formatting it specifically for use by the Map Visualizer. Data files have user-defined extensions (the sample files provided with the Map Visualizer have a *.data* extension).

- A gfx file consisting of a description of the shapes and locations of the 1-, 2-, or 3-dimensional objects to be displayed.

  Gfx files must have a *.gfx* extension. MineSet includes various *.gfx* files, including the United States to the granularity of counties, telephone area codes, and postal zip codes, as well as Canada to the granularity of provinces. You can also manually generate *.gfx* files (see Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer” for the required file format).

- A hierarchy file consisting of a description of
  - the column names of the various graphical objects to be displayed
  - the filenames of the *.gfx* files that describe the locations and shapes of the graphical objects
  - an optional description of the hierarchical relationship of the graphical objects, which is used for the drill-down and drill-up functions.

Hierarchy files enable drill down and drill up. This means that information associated with objects at one level can be aggregated (or, conversely, shown in greater detail) and displayed at a different level. For example, a hierarchy file defining the relationships between states and regions comprising multiple states allows values such as population levels to be displayed at both the individual state level as well as at regional levels. The *gfx_files/usa.states.gfx* file, for example, describes the shapes of the 50 United States; the *gfx_files/usa.states.hierarchy* file describes the hierarchy grouping individual states into regions, regions into East-West areas, and the East-West areas into an aggregated United States.

For more information, see Appendix B, "Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer."

- A configuration file describing the format of the input data and how these are to be displayed. Typically, this file is created using the Tool Manager (see Chapter 3). You also can use an editor (such as jot, vi, or Emacs) to produce this file without using the Tool Manager (see Appendix B, "Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer").

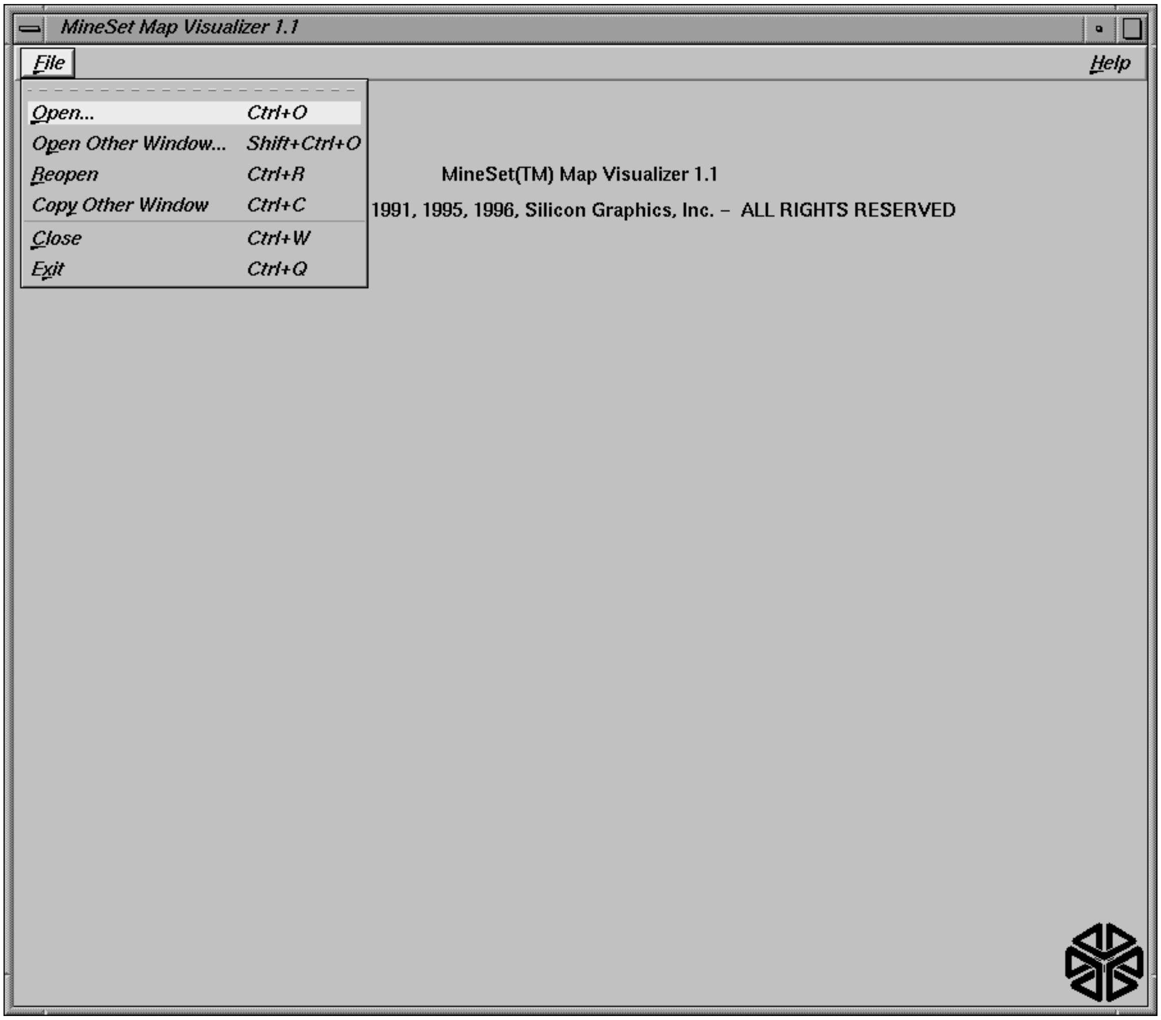
  Configuration files should have a *.mapviz* extension. If they do not, they are not listed when selecting the Open option from the File pulldown menu. When starting the Map Visualizer, or when opening a file, specify the configuration file, not the data file.

Starting the Map Visualizer

There are five ways to start the Map Visualizer:

- Use the Tool Manager to configure and start the Map Visualizer. See Chapter 3 first for details on most of the Tool Manager's functionality, which is common to all MineSet tools; see below for details about using the Tool Manager in conjunction with the Map Visualizer.

- Double-click the Map Visualizer icon, which is in the MineSet page of the icon catalog. The icon is labeled *mapviz*. Since no configuration file is specified, the start-up screen requires you to select one by using File > Open.



MineSet(TM) Map Visualizer 1.1

1991, 1995, 1996, Silicon Graphics, Inc. – ALL RIGHTS RESERVED

Figure 5-4 Map Visualizer’s Startup Screen, With File Pulldown Menu Selected

Starting the Map Visualizer without specifying a configuration file causes the main window to show the copyright notice for this tool. Only the File and Help pulldown menus can be used. For the main window to be fully functional, open a configuration file by selecting File > Open (Figure 5-4).

- If you know what configuration file you want to use, double-click the icon for that configuration file. This starts the Map Visualizer and automatically loads the configuration file you specified. This only works if the configuration filename ends in *.mapviz* (which is always the case for configuration files created for the Map Visualizer using the Tool Manager).
- Drag the configuration file icon onto the Map Visualizer icon. This starts the Map Visualizer and automatically loads the configuration file you specified. This works even if the configuration filename does not end in *.mapviz*.
- Start the Map Visualizer from the UNIX shell command line by entering this command at the prompt:

  ```
  mapviz [ configFile ]
  ```

  where *configFile* is optional and specifies the name of the configuration file to use. If you don’t specify a configuration file, you must use File > Open to specify one (see Figure 5-4).

Configuring the Map Visualizer Using the Tool Manager

This section describes how the Map Visualizer can be configured using the Tool Manager. Although the Tool Manager greatly simplifies the task of configuring the Map Visualizer, you can construct a configuration file manually for this tool using a text editor (see Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer”).

Note that the steps required to connect to a data source are described in Chapter 3.

Generating .gfx and .hierarchy Files

To use the Map Visualizer, you must provide the application with two files that define the graphical objects to be displayed:

- One or more *.gfx* files, which define the shapes of the graphical objects displayed.
- A.*hierarchy* file, which describes the relationship of multiple, interrelated map (*.gfx*) files.

These files are not created by the Tool Manager; they must already exist as part of MineSet (residing in the */usr/lib/MineSet/mapviz/gfx_files* directory), or they must be created by the user. For instructions on their creation, see Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer.”

The *.gfx* and *.hierarchy* files that are part of the MineSet package include

- the individual states of the United States
- the individual counties of the United States
- the individual five-digit ZIP codesof the United States
- the telephone area codes of the United States
- the individual provinces and territories of Canada
- the individual states of Mexico
- the individual states and territories of Australia
- the individual countries of Western and Central Europe
- regional subdivisions of both France and The Netherlands

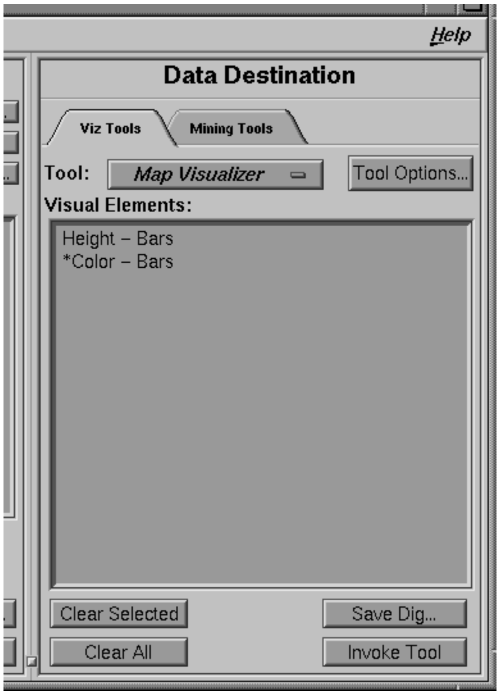
The Map Visualizer requires a data file with

- One column indicating geographical objects (for example, states). Each row in this column must indicate a unique geographical object (staying with the example, this means one row for each state).
- At least one column with numeric values mapped (using arithmetic expressions) to the heights and / or colors of each geographic bar. These columns can be scalar, a 1D array, or a 2D array. If the column is an array, a slider must be used to select specific data points for this mapping to heights and colors.

If both heights and colors are mapped to 1D or 2D arrays, the arrays must have the same indexes (see Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer”).

Selecting the Map Visualizer Tool

Select the *Viz Tools* tab in the Data Destination panel of the Tool Manager's main screen (Figure 5-5). From the popup list of tools, select *Map Visualizer*. The mapping requirements for the Map Visualizer are displayed in the window on the right side of this panel. Items in the Visual Elements list that are preceded by an asterisk are optional.



Tool: Map Visualizer

Visual Elements:

Height – Bars
*Color – Bars

Figure 5-5 Data Destination Panel, With Map Visualizer Selected

Height - Bars—This key lets you specify the heights of the geographic bars on the map.

*Color - Bars—This optional key lets you assign the colors of the geographic bars. See “Color Options for the MineSet Visualizers” in Chapter 3 for a more detailed explanation of how to choose and change colors.

Mapping Columns to Visual Elements

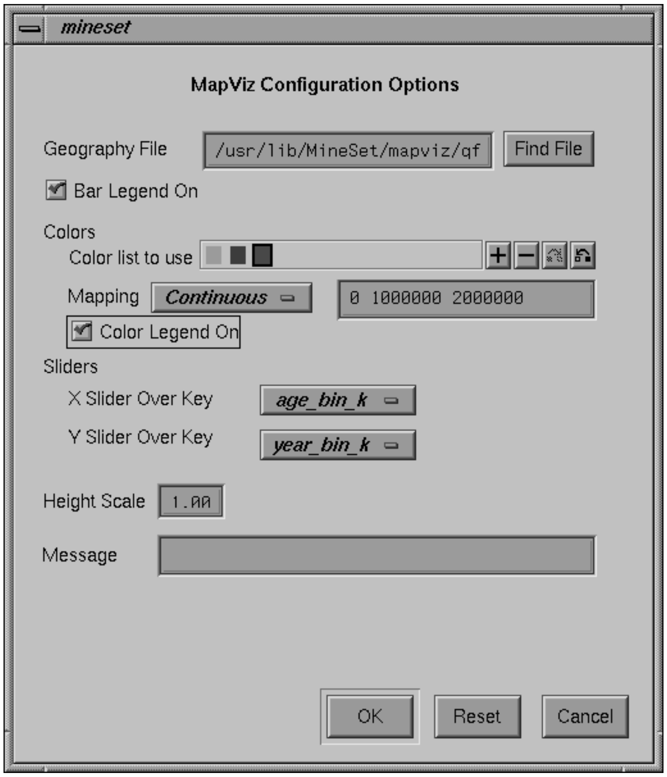
A column in the Current Columns window should be mapped to the Visual Element Height - Bars by clicking the column first, then Height - Bars. Optionally, another column (perhaps even the same column) can be mapped to the Visual Element *Color - Bars.

Undoing Mappings

To undo a mapping, select the mapping in the Requirements: window, then click the *Clear Selected* button. To undo all mappings, click the *Clear All* button.

Specifying Tool Options

Clicking the *Tool Options* button causes a new dialog box to be displayed (Figure 5-6). This lets you change some of the Map Visualizer options from their default values.



MapViz Configuration Options

Geography File /usr/lib/MineSet/mapviz/qf

☑ Bar Legend On

Colors

Color list to use

Mapping Continuous 0 1000000 2000000

☑ Color Legend On

Sliders

X Slider Over Key *age_bin_k*

Y Slider Over Key *year_bin_k*

Height Scale 1.00

Message

Figure 5-6 Map Visualizer's Options Dialog Box

The following sections describe the buttons and fields of the Map Visualizer's Options dialog box.

Geography File

This option lets you specify a *.hierarchy* file to be used for the representation of the geographical objects in the Map Visualizer's main window.

The *Find File* button lets you browse your files to find the *.hierarchy* file to be used.

Note that the Geography File field is optional. If not supplied, the Map Visualizer expects the first column in the Current Columns window to be a string column, and presumes that that column contains the names of various geographical objects. Since no specific description of the size and shape of these geographical objects is identified, the Map Visualizer displays these objects as simple rectangles, arbitrarily sized and placed in the main viewing window.

Bar Legend On Button

The *Bar Legend On* button lets you determine whether a legend that describes the graphical object heights is displayed or hidden.

Colors Options Field

To use these Colors options, you must have mapped a column to the **Color - Bars* requirement of the Data Destination panel. See "Color Options for the MineSet Visualizers" in Chapter 3 for a more detailed explanation of how to choose and change colors.

Color list to use**—**You can specify the color list using the + button next to the color list label. This brings up a color editor that lets you specify a color to be added to the list.

Mapping**—**You can specify whether the color change that is shown in the graphic display is Continuous or Discrete. If you choose Continuous, the color values shift gradually between the colors entered in the "Color list to use" field as a function of the values that are mapped to those colors in the "Mapping" field.

The field to the right of the popup button lets you enter specific values to which the colors are mapped. You must have the same number of values in this field as there are colors entered in the “Color list to use” field.

If you

- used the Color Browser to choose gray and red
- selected Discrete for the Mapping
- entered the values `0 150000`

then the display shows the population of the United States across the time period 1770-1990. States with more than 150,000 square miles are shown in red, the rest are in gray.

If you

- used the Color Browser to choose gray and red
- selected *Continuous* for the *Mapping*
- entered the values **`0 300000`**

then the display shows the population of the United States across the same time period. The states’ colors vary from gray to red, depending on their size; the largest states are shown with the greatest density of red.

You can enter as many colors into this field as you think are necessary for your display. If the number of values in the column that maps to *Color - Bars exceeds the number of distinct colors you have chosen, the Map Visualizer adds an appropriate number of randomly chosen colors at runtime.

Color Legend On Button

The *Color Legend On* button lets you determine whether a color legend is displayed or hidden.

Sliders Options Fields

If an array of values are mapped to the height or color requirements, you must specify an X or Y slider. If the array is a 2D array, you must specify a Y slider. The popup buttons next to these options provide a list of available array keys over which the sliders operate. If both the X and Y sliders have *None* selected as their key, the resulting display does not include animation.

Height Scale Field

This field lets you enter a value by which the height values are scaled in the display. Normally, the height variable is mapped directly to the height of the graphical objects, so that the tallest object (with the largest numeric value) rises towards the top of the view window. Entering a scale value in this field causes all objects in the display to be multiplied by that value.

Message Field

This lets you specify the message displayed when an entity is selected. For a listing and description of format types that can be entered in this field, see the “Message Statement” section in Appendix B, “Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer”

Resetting the Tool Options

If, after making changes to the Tool Options dialog box, you want to reset the values of all options to their default values, click the *Reset Options* button.

Accepting the Tool Options

Once you have finished making changes to the Tool Options dialog box, click *OK* to return the Tool Manager’s main screen.

Saving Map Visualizer Settings

The Tool Manager stores information for the Map Visualizer in several files, all sharing the same prefix:

- <*prefix*>*.mapviz.data* contains data.
- <*prefix*>*.mapviz.schema* describes the data file.
- <*prefix*>*.mapviz* contains information needed by the Map Visualizer.
- <*prefix*>*.mineset* contains all the information needed to create the other files.

To specify a prefix, use the *Save ...* menu option in the File menu of the Tool Manager's main window. If you do not specify a prefix, the default *untitled* is used.

When you use the *Invoke Tool* button, the *.data*, *.schema*, and *.mapviz* files are updated, if necessary.

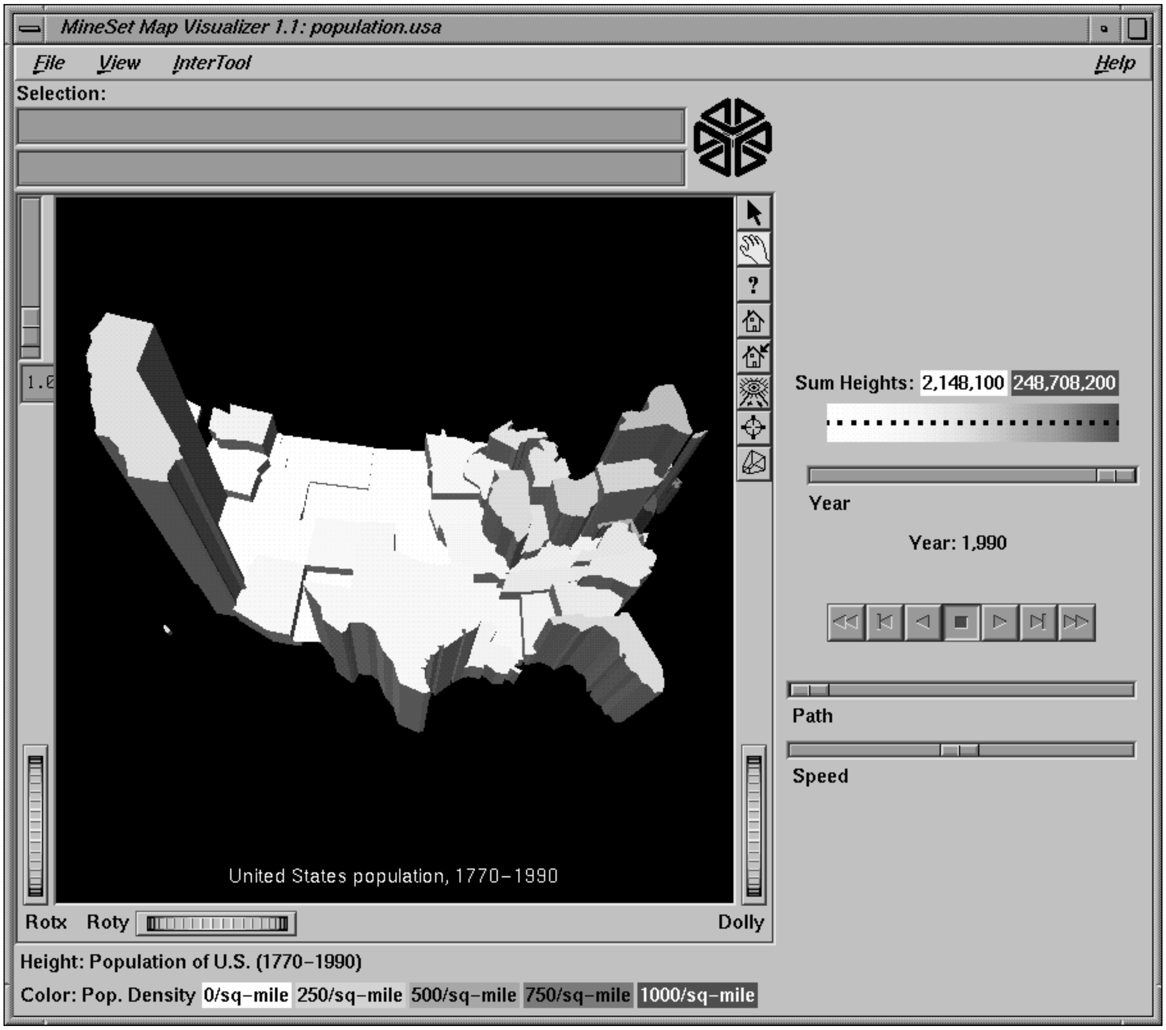
Invoking the Map Visualizer

To see the Map Visualizer graphically represent your data, click the *Invoke Tool* button at the bottom of the Data Destination panel.

Working in the Map Visualizer's Main Window

If you started the Map Visualizer without specifying a configuration file, the main window shows the copyright notice for the Map Visualizer. Only the File and Help pulldown menus can be used. For the main window to show all menus and controls, open a configuration file. Use File > Open (Figure 5-4) to see a list of configuration files.

When a valid configuration file has been specified, its geographical landscape is visible. For example, Figure 5-7 shows the results of specifying *population.usa.mapviz* and moving the Year slider to the far right.


MineSet Map Visualizer 1.1: population.usa
File
View
InterTool
Help
Selection:
1.0
United States population, 1770–1990
Rotx
Roty
Dolly
Year
Path
Speed

Sum Heights: 2,148,100 248,708,200

Year: 1,990

Height: Population of U.S. (1770–1990)

Color: Pop. Density 0/sq–mile 250/sq–mile 500/sq–mile 750/sq–mile 1000/sq–mile

Figure 5-7 Population.usa.mapviz Example With the Slider Moved to 1990

This shows the population and population density for each state of the United States. The population of each state is represented by the height of the state's graphical shape. Heights are relative to each other across the entire range of the animation controls.

Viewing Modes

The two modes of viewing are *grasp* and *select*. To toggle between these modes, move the cursor into the main window, and press the Esc key. You can also change from one mode to the other by clicking the appropriate button: to enter select mode, left-click the arrow button (to the top-right of the main window); to enter grasp mode, left-click the hand button (immediately below the arrow button, near the top right of the main window).

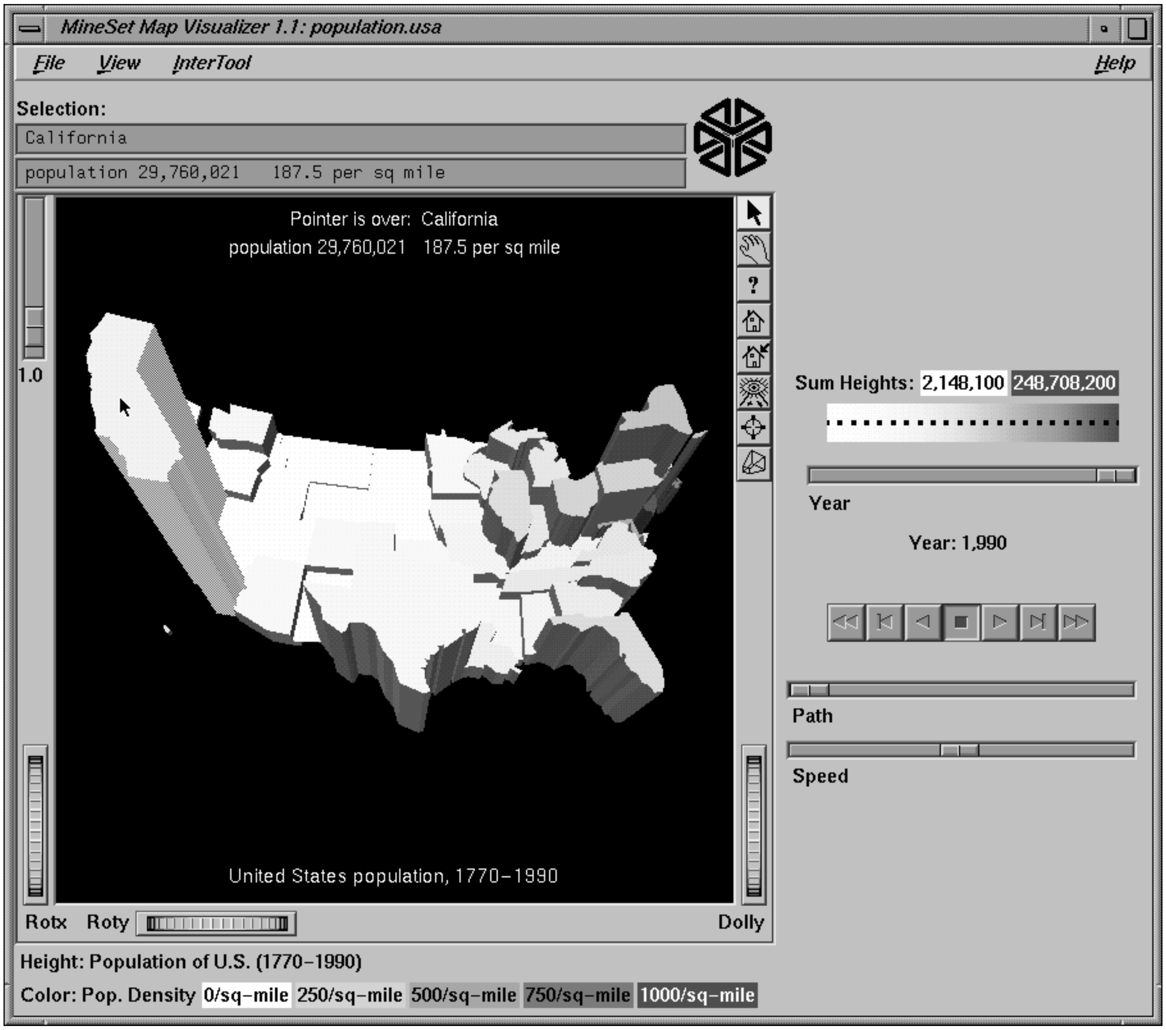
Grasp Mode

In grasp mode, the cursor appears as a hand. This mode supports panning, rotating, and scaling the scene's size in the main window.

- To pan the display, press the middle mouse button and drag it in the direction you want the display panned.
- To rotate the display, press the left mouse button and move the mouse in the direction you want to rotate.
- To move the viewpoint forward, press the left and middle mouse buttons simultaneously and move the mouse downwards. To move the viewpoint backward, press the left and middle mouse buttons simultaneously and move the mouse upwards. This is equivalent to the functions provided by the Dolly thumbwheel.

Select Mode

In select mode, you can highlight an object by positioning the cursor over that object. Information about that object then appears at the top of the view area. This information remains visible in the window only as long as the pointer cursor remains over the object. If you position the pointer cursor over an object and click the left mouse button, the same information appears in the Selection Window, which is above the main window, under the "Selection" label (Figure 5-8).



Selection:
California
population 29,760,021   187.5 per sq mile

Pointer is over: California
population 29,760,021   187.5 per sq mile

United States population, 1770–1990

Sum Heights: 2,148,100 248,708,200

Year: 1,990

Height: Population of U.S. (1770–1990)

Color: Pop. Density 0/sq–mile 250/sq–mile 500/sq–mile 750/sq–mile 1000/sq–mile

Figure 5-8 Example of a Highlighted (Information in the Viewing Window) and Selected (Information in the Selection: Window) Object

This Selection information remains visible until you select another object or click the black background. Using the mouse, you can cut and paste this text into other applications, such as reports or databases.

Drill down and drill up functionality—To view a finer level of geographical granularity for an object (if the *.data* and *.hierarchy* files support it), click the right mouse button while the cursor is over that object. This is called "drilling down." You can repeat this down to the finest level of granularity supported by the data. If the cursor is positioned over a specific object when drilling down, only the more detailed sub-objects of that object appear. If, instead, the cursor is positioned on the black background at the time of the mouse click, then the more detailed sub-objects of the entire set of objects appear. This might produce a display with a large number of individual objects. The greater the number of objects, the longer the Map Visualizer takes to construct the scene, and the slower the performance when moving the animation controls.

To move up one level and view a coarser geographical granularity ("drill up"), click the middle mouse button. If the cursor is positioned on the black background when you click, all the higher-level objects appear. If the cursor is positioned on a specific object in the scene, then the scene "returns" to the group of higher-level objects visible when you last drilled down with the right mouse button.

Note: By default, the Map Visualizer initially displays objects at the lowest level of detail; thus, initially, only drill-up (to coarser granularity) is active.

External Main Window Controls

Several external controls surround the graphics window. These consist of buttons, sliders, and a summary window. Each of these controls is described in this section.

Buttons

At the top right of the image area are eight buttons, each of which is selectable with the left mouse button, as shown in Figure 5-9.

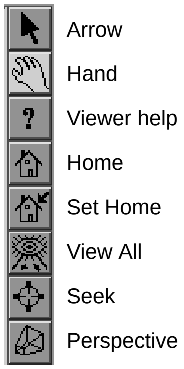


Figure 5-9 Top Right Buttons

- *Arrow* puts you in select mode. When in this mode, the cursor shape is an arrow. Select mode lets you highlight graphical objects in the main window, as well as drill down or drill up to different levels of geographical granularity.
- *Hand* puts you in grasp mode. When in this mode, the cursor shape is a hand. Grasp mode lets you rotate, zoom, and pan the display in the main window.
- *Viewer help* (symbolized by a question mark) brings up a help window describing the viewer itself.

- *Home* takes you to a designated location. Initially, this location is the first viewpoint shown after invoking the Map Visualizer and specifying a configuration file. If you have been working with the Map Visualizer and have clicked the *Set Home* button, then clicking *Home* returns you to the viewpoint that was current when you last clicked *Set Home*.
- *Set Home* makes your current location the Home location. Clicking the *Home* button returns you to the last location where you clicked *Set Home*.
- *View All* lets you view the entire Map Visualizer display, keeping the angle of view. To get an overhead view of the scene, rotate the camera so that you are looking directly down on the display, then click the *View All* button.
- *Seek* takes you to the point or object you click after selecting this button. This changes the perspective and angle of your viewpoint.
- *Perspective* lets you view the scene in 3D perspective (closer objects appear larger, farther object appear smaller). Clicking this button toggles 3D perspective on (default setting) or off.

  Note: If *Perspective* is off, the Dolly thumbwheel becomes the Zoom thumbwheel.

Height-Adjust Slider and Label

To the left of the Map Visualizer's main window is a vertical height adjust slider and, below it, a label containing a numeric value between 0.1 and 100. This slider lets you change the absolute heights of all the graphical objects in the main window. Moving the slider up increases the heights of the objects; moving it down decreases their heights. The numeric value in the label changes accordingly. This value indicates the height multiplier, the default value of which is 1.0. The height adjust slider is useful for accentuating relative height differences between objects in the view window.

Thumbwheels

Three thumbwheels appear around the lower part of the main window border (see Figure 5-10). They let you dynamically move the viewpoint.

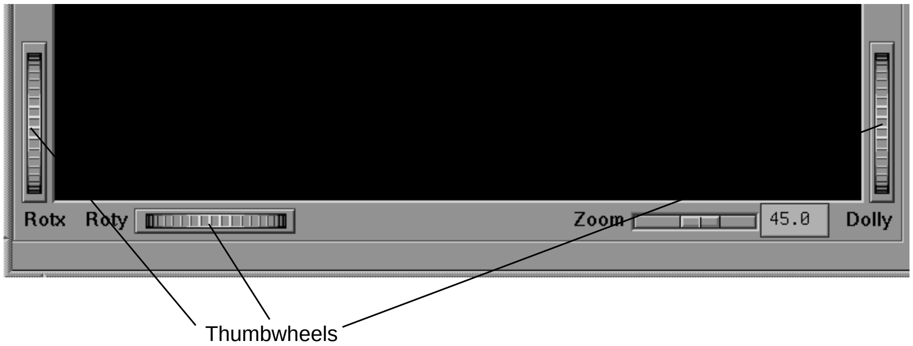


Figure 5-10 Lower Half of Window With Thumbwheels

- The vertical thumbwheel *Rotx* (rotate about the x axis), on the left, rotates the display up and down.
- The horizontal thumbwheel *Roty* (rotate about the y axis), at the bottom left, rotates the scene in the main window around its centerpoint left and right.
- The vertical *Dolly* thumbwheel, on the right, moves the viewpoint forward and backward. Note that as you use the Dolly thumbwheel to magnify the scene in the main window, additional detail can appear. This is not the case with the Zoom slider, which merely enlarges the scene without adding detail.

  Note: If *Perspective* is off, the Dolly thumbwheel becomes the Zoom thumbwheel, and the Zoom slider and Zoom factor box disappear.

The Animation Control Panel

To the right of the Map Visualizer's main window are several external controls, depending on the type of data being displayed (see Figure 5-11). These controls can include

- sliders for independent dimensions
- a summary window containing a color density profile.
- a color legend showing the color density value limits
- buttons and sliders for animation

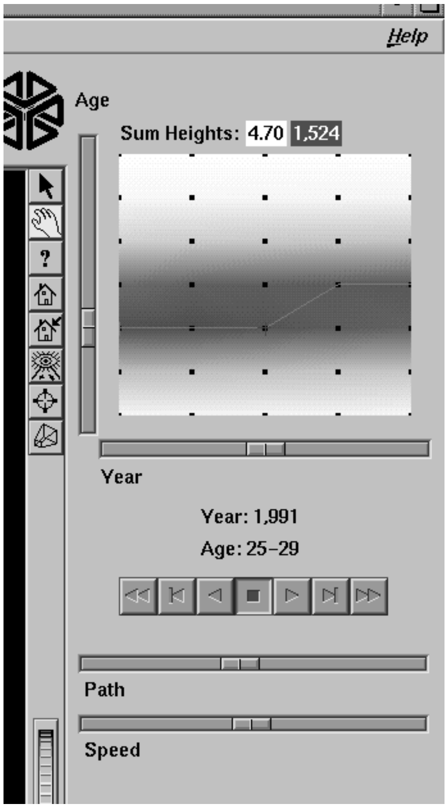


Figure 5-11 Map Visualizer's Summary Window With Slider and Animation Controls

Sliders Controlling Independent Dimensions

The number of sliders appearing adjacent to the summary window is dependent on the dataset displayed in the Map Visualizer's main window. Datasets can have two, one, or no independent dimensions.

Datasets With Two Independent Dimensions

If the dataset has two dimensions of independently varying data (such as *nl.births.mapviz*), the animation control panel to the right of the main graphics window becomes visible (as in Figure 5-11).

Within this animation control panel are the 2D summary window and two sliders. The summary window has a horizontal slider below it for selecting data points of the first independent dimension, and a vertical slider to the left for selecting data points of the second independent dimension. The horizontal slider's dimension is identified by a label below it. The vertical slider's dimension is identified by a label above it.

Datasets With One Independent Dimension

For datasets with one independent dimension (such as *population.usa.mapviz*), only the slider below the summary window appears, and the summary window is compressed (see Figure 5-12). This slider's dimension is identified by a label below it.

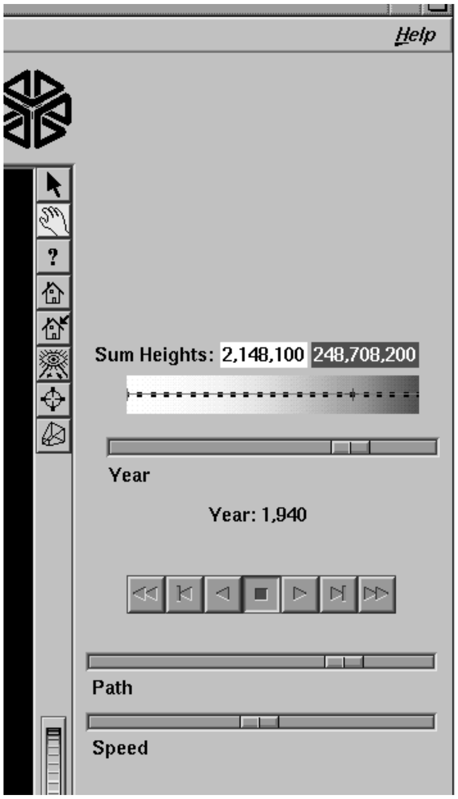
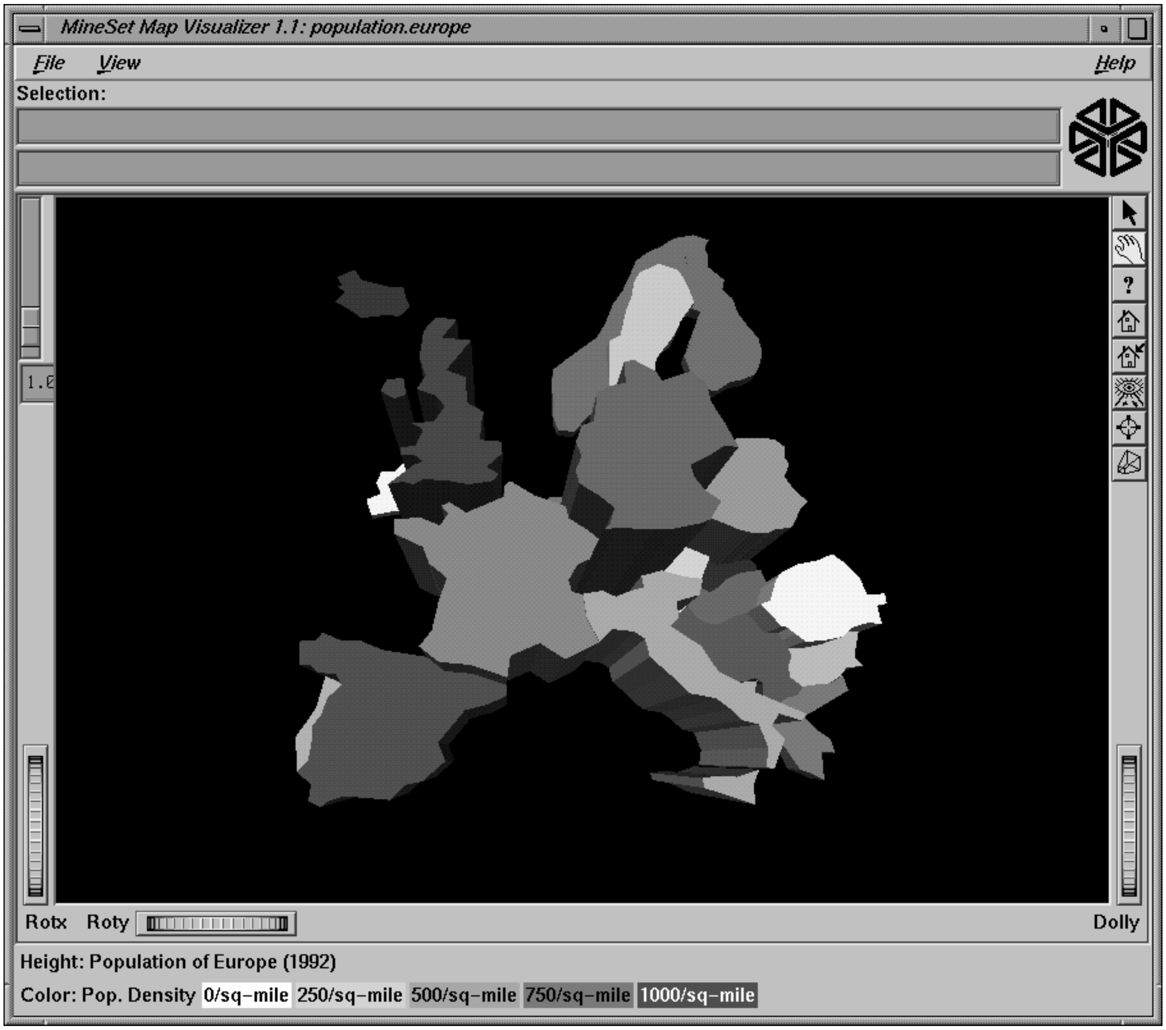


Figure 5-12 Map Visualizer's Summary Window With One Slider and Animation Controls

Datasets With No Independent Dimension

For datasets with no independent dimensions (such as *population.europe.mapviz*), no animation control panel appears (see Figure 5-13).



Height: Population of Europe (1992)

Color: Pop. Density 0/sq–mile 250/sq–mile 500/sq–mile 750/sq–mile 1000/sq–mile

Figure 5-13 If There Are No Independent Dimensions, No Animation Control Panel Appears

The Summary Window

The summary window provides a 2D representation of the aggregation of values that the main window displays in 3D. Above this window is a label, Sum Heights, followed by two rectangles: the first white, the second red. Within the rectangles are numbers; each is the respective value for the maximum density of that color. This summary color legend provides a visual and numeric comparison to the densities in the summary window.

The whiter the areas of the summary window, the lower the total values represented by the heights of the objects in the main window. The greater the density of red shown in areas of the summary window, the higher the total of those values. The density of these colors in the summary window provides a summary of the data across the one or two independent dimensions in the dataset, which is useful for guiding your exploration through the data.

By default, the summary window also contains a set of black dots, evenly spaced across the one or two dimensions of data. These dots indicate the precise positions of the discrete datapoints of the data. You can turn off the dots using the View > Show Data Points menu option.

Color Density Examples in the Summary Window

After opening the *population.usa.mapviz* file, for example, the 2D summary window shows a color range from white (on the left) to red (on the right). White corresponds to the low aggregate population in the early years of the United States; red represents the higher aggregate population in later years. In this example, the greater the density of red, the higher the total population of United States.

For a more complex example, open *perhouse.perage.mapviz*. This dataset has two independent dimensions: time and age. The summary window displays these dimensions as a complex pattern of colors. Place the cursor on the horizontal lines with the greatest density of red, which runs horizontally across the summary window (this means the age group making the greatest number of purchases). Click the left mouse button. The information displayed in the field below the horizontal slider shows that this represents purchases made by 30- to 39-year-olds.

Now place the cursor at the junction of the densest red horizontal (age group) and vertical (time frame) parts of the summary window, and click the left mouse button. The information displayed in the field below the horizontal slider shows that most purchases were made by 30- to 39-year-olds in May-June 1989 and May-June 1990.

Creating a Path in the Summary Window

If the dataset loaded into the Map Visualizer has at least one independent dimension, it is possible to view all or any part of that dataset via animation. This is done by first creating a path in the summary window, then activating the animation controls described in the next section.

The three ways to draw a path in the summary window are as follows:

- Define a starting point by clicking and holding down the left mouse button, then draw an arbitrary path by dragging the cursor over the window. End the path by releasing the left mouse button.
- Define a starting point by clicking the left mouse button, then define an endpoint by moving the cursor to another part of the window and clicking the middle mouse button. A path appears between those two endpoints, passing through the intermediate discrete data point(s) that are closest to the hypothetical straight line between the endpoints. To add more line segments, continue with repeated middle mouse clicks.
- Define a starting point by clicking the left mouse button, then drag one of the independent dimension sliders to draw a straight line along this dimension. If there are two sliders, then using the second slider will continue to draw a straight line along the axis controlled by this second slider.

The path you draw can only go through the well-defined discrete data points, identified by the black dots in the summary window.

Animation Buttons and Sliders

Use the seven VCR-like buttons and two sliders (*Path* and *Speed*) below the 2D summary window to control animation.

Animation Buttons

Once a path is drawn in the summary window (see “Creating a Path in the Summary Window,” above), you can use the VCR-like buttons to control animation along this path. The middle *Stop* button is highlighted in blue to indicate an initial state. Use the adjacent *Play Forward* button (to the right of *Stop*) or *Play Reverse* (to the left) to begin simple movement along the drawn path in a forward or reverse direction. *Forward* and *Reverse* are defined by the sequence in which the path was drawn, not by a sense of left-to-right or right-to-left movement.

To stop and restart the animation, click the *Stop* button, then use the *Play Forward* or *Reverse* button. When you use the *Stop* button, the animation continues in the current direction until the position falls on a discrete data point.

Adjacent to the *Play* buttons are the *Single-Step* buttons, also *Forward* and *Reverse*. Clicking one of these buttons causes the current path position to change to the next discrete data point.

On the outside are the *Fast Forward* and *Fast Reverse* buttons. Clicking one of these *Fast* buttons while in *Stop* state changes the path position to the end (for *Forward*) or to the beginning (for *Reverse*) of the path. Clicking a *Fast* button when in *Play* state increases the animation speed.

Animation Sliders

While animation is stopped, you can move the *Path* slider to reset the position along the path. Note that when you use the Path slider, the cursor in the summary window moves across the drawn path, and the 1D sliders (below and to the left of the drawing area) move consistently with the cursor position. Then use the *Play* or *Reverse* button to restart the animation from the newly specified point.

You can drag the *Path* slider to an arbitrary position on the path between discrete data points; however, when you release the slider, the path position changes to a stop at the nearest discrete data point.

Use the *Speed* slider to adjust the speed of the animation along the path.

Data Points and Interpolation

As animation proceeds, the variables mapped to height and color in the Map Visualizer also change. However, the variables displayed in the “Selection:” message box show only the data values of the nearest discrete data position, not intermediate (interpolated) data values.

The animation is produced in the following manner: Assume you have data for 10 years, on a per-year basis (that is, 10 data values) and that these correspond to the height of one state in the Map Visualizer. If the years are 1991 to 2000, the height for 1991 is 20, and the height for 1992 is 40. As you move the year slider from 1991 to 1992, the height changes by being uniformly interpolated between 20 and 40. For example, midway between 1991 and 1992, the height appears to be 30. As you approach 1992, the height approaches 40. However, you cannot Stop an animation between discrete data points, and you cannot drag the *Path* slider to a stationary position between discrete data points.

The data points in the summary window represent the slider positions corresponding to the actual data from the data file. For example, the heights 20 and 40 are representations of actual data, but the height 30 is not. In this example, there would be data points in the summary window at the slider positions corresponding to each year.

Note that not all variables are required to vary with a slider. For example, in the Map Visualizer, the area and name of the state do not vary with the slider (for example, year). If there are two sliders, some variables can vary with only one of the sliders, while other variables vary with both.

Pulldown Menus

Four pulldown menus let you access additional Map Visualizer functions. These are labeled File, View, InterTool, and Help. If you start the Map Visualizer without specifying a configuration file, only the File and the Help menus are available. The View menu is available after a valid dataset is loaded.

The File Menu

The File menu (Figure 5-14) contains six options. This section describes those options.

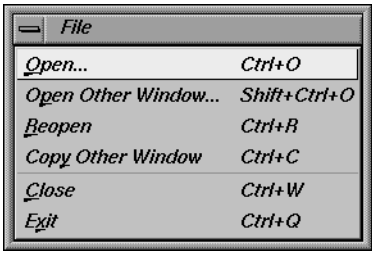


Figure 5-14 Map Visualizer’s File Pulldown Menu

- *Open* loads and opens a configuration file. This causes it to be displayed in the main window. Previously displayed data is discarded. Use *Open* to view a new dataset, or to view the same dataset after changing its configuration.
- *Open Other Window* opens a configuration file and displays its results in a different window. The current dataset in the first window remains open.
- *Reopen* opens the currently open configuration file again.
- *Copy Other Window* opens a new window displaying the same dataset. You can interact with these windows independently, or you can synchronize these windows using the InterTool pulldown menu.
- *Close* closes the current window and all its associated panels. If no other windows are open, *Close* exits the application.
- *Exit* closes all windows and exits the application.

The View Menu

The View menu (Figure 5-15) contains five options. This section describes those options below.

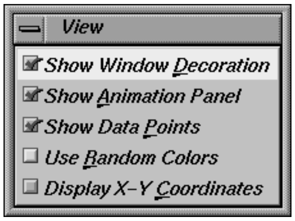


Figure 5-15 Map Visualizer’s View Pulldown Menu

- *Show Window Decoration* causes the buttons around the main window to be displayed. Default for this option is on. Toggle this option to make the window decoration disappear.
- *Show Animation Panel* causes the animation control panel to be displayed to the right of the main view. Click this option again to deselect it. When this option is deselected, the animation panel is not displayed. Not displaying the animation panel can be useful when you have applied the InterTool menu’s *Synchronize All Mapviz Sliders* option (described in the “The InterTool Menu” on page 149) and need only a single animation control panel on the screen.
- *Show Data Points* causes a grid of black dots to appear (or disappear) in the 2D summary window. Each dot denotes the precise position of a discrete data value in the input dataset. For example, if the input dataset has 10 data values across one independent dimension, then you see heights and colors of the graphical objects in the main window vary continuously, based on data values that are interpolations between these discrete data points. These data point dots in the summary window help you better understand when the heights and colors are derived directly from the input data values, and when they are derived indirectly from interpolated values.
- *Use Random Colors* causes the configuration file’s color mapping specifications (for example, white-to-red shadings representing population density) to be ignored. Random, constant colors are assigned to the graphical objects. Click this option again to deselect it.

- *Display X-Y Coordinates* puts the Map Visualizer into a special mode that lets you identify X-Y vertex pairs at specific points of the scene in the main window. In this mode, the Map Visualizer resets the cursor to select mode and displays 3D objects as flat background lines. Clicking the left mouse button on various parts of the displayed scene causes the corresponding X-Y vertex pair values to appear in the Selection Details window. You can also enter the vertex pair points into the *.gfx* file to identify point objects or the endpoints of line objects for subsequent display. Note that displaying X-Y coordinates is used for developing and refining *.gfx* files, not for data analysis.

  When *Display X-Y Coordinates* mode is initially enabled, or when a point in the background is selected, the selection window shows the minimum and maximum X-Y pairs of the currently displayed image in the main window. Add these two value pairs to the new *.gfx* file you are generating. The first record in the file *gfx_files/usa.cities.gfx* shows an example of how the min-max pairs of the *usa.states.gfx* file were entered into the associated *usa.cities.gfx* file. This ensures that the X-Y coordinate pairs in *usa.cities.gfx* share the same coordinate system as the X-Y coordinate pairs in *usa.states.gfx*.

The InterTool Menu

The InterTool menu has one option, as shown Figure 5-16.

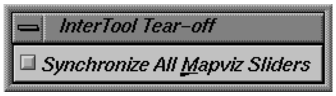


Figure 5-16 Map Visualizer’s InterTool Pulldown Menu

Selecting *Synchronize All Mapviz Sliders* identifies this Map Visualizer window as one in a “synchronized sliders” cooperative: changing the current slider positions in one Map Visualizer window causes / produces the same change in all others currently open. Click this option again to deselect it. This menu option must be selected in every Mapviz main window that is to be part of the synchronization.

Note that currently only the sliders' physical positions are synchronized, not the underlying meanings of those positions. For example, synchronizing *population.usa.mapviz* (with dates ranging from 1770 to 1990) and *population.canada.mapviz* (with dates ranging from 1871 to 1991) probably is not useful, since the slider physical midpoint position represents 1880 in the United States and 1931 in Canada. Generally, synchronization is useful only when the sliders of each dataset represent the same range of independent variables.

The Help Menu

The Help menu (see Figure 5-17) provides access to five help functions. This section describes those functions.

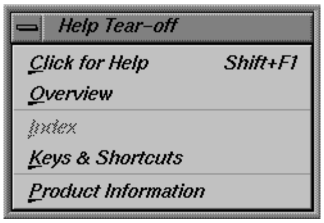


Figure 5-17 Map Visualizer's Help Pulldown Menu

- *Click for Help* turns the cursor into a question mark. Placing this cursor over an object in the Map Visualizer's main window and clicking the mouse causes a help screen for that object to appear. Closing the help window restores the cursor to its arrow form and deselects the help function. The keyboard shortcut for this function is Shift+F1. (Note that it also is possible to place the arrow cursor over an object and press the F1 function key to access a help screen about that object.)
- *Overview* provides a brief summary of the major functions of this tool, including how to open a file and how to interact with the resulting view.
- *Index* provides an index of the complete help system. This option is currently disabled.
- *Keys & Shortcuts* provides the keyboard shortcuts for all of Map Visualizer's functions that have accelerator keys.

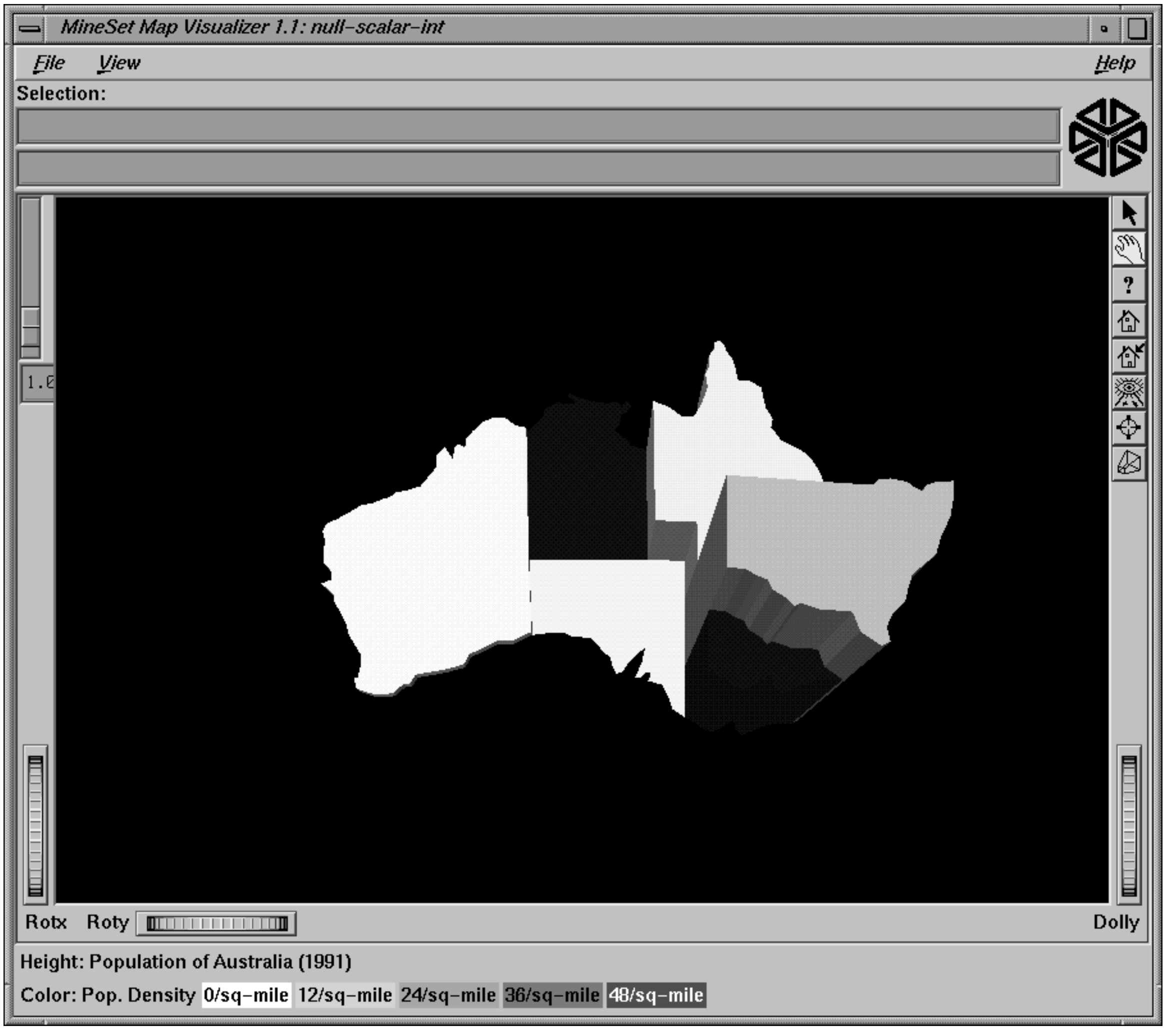
- *Product Information* brings up a screen with the version number and copyright notice for the Map Visualizer.
- *MineSet User's Guide* invokes the IRIS Insight viewer with the online version of this manual.

Null Handling in the Map Visualizer

Nulls represent unknown data (see Appendix G, "Nulls in MineSet").

In the Map Visualizer, nulls can occur when any of the following are true:

- The database or data file contains a null.
- The Tool Manager is used to make an array based on bins and no data falls into a specific bin. For example, if there is no data for the 30-40-year-old population, that bin is null.
- The Tool Manager is used to make an array and the null enum option is specified. In this case, an extra array element is created to represent the aggregation of all the values for which the bin value is null. The Tool Manager assigns the question mark (?) character to this extra bin. To view the values of this bin, move the corresponding slider to its left-most position. If there are no data for that null bin, the values associated with it are null as well, and the Map Visualizer represents the corresponding graphical object(s) as a "null object."
- Expressions and aggregations of nulls can generate nulls (see Appendix G, "Nulls in MineSet").
- Mapping a null value to a visual attribute, special representations are used in the Map Visualizer. A null height results in a dark grey object with zero height; a null color results in an object with appropriate height (as defined by the value mapped to height), but with a dark gray color (see Figure 5-18).



Height: Population of Australia (1991)

Color: Pop. Density 0/sq-mile 12/sq-mile 24/sq-mile 36/sq-mile 48/sq-mile

Figure 5-18 Representation of a Null Value Mapped to Height (Top Middle Object) and to Color (Bottom Right Object)

When selecting an object with a null value, it is shown as a question mark (?) in the selection field.

Sample Configuration and Data Files

The provided sample configuration and data files demonstrate the Map Visualizer's features and capabilities. The *.data* and *.mapviz* files are in the directory */usr/lib/MineSet/mapviz/examples*; the *.gfx* and *.hierarchy* files are in the directory */usr/lib/MineSet/mapviz/gfx_files*.

- *blocks.mapviz*, *blocks.data*, *blocks.gfx*, and *blocks.hierarchy*
  This simple example shows four adjacent blocks. The height and color of each block varies based on the underlying data in *blocks.data*. You can drill up using the middle mouse button (see the "Select Mode" section) to see the upper pair and the lower pair of blocks aggregate; then drill up again to see these upper and lower blocks aggregate into a single block. You can drill down using the right mouse button to see the objects of finer granularity reappear.
- *population.australia.mapviz*, *population.australia.data*, *australia.states.gfx*, and *australia.states.hierarchy*
  The data file contains one row for each Australian state and territory. Each row contains three tab-separated items: a keyword name for the state or territory, the population value, and the size of the territory.

  This sample graphically displays the 1991 population and population density of the Australian states and territories. Heights of the graphical objects represent the relative population; color represents the relative population density. A legend at the bottom of the display describes the color range and the associated values.
- *population.canada.mapviz*, *population.canada.data*, *canada.provinces.gfx*, and *canada.provinces.hierarchy*
  The data file contains one row for each Canadian province and territory. In this example, each row contains 13 blank-separated values (one for each decade between 1871 and 1991).

  This sample graphically displays the population and population density of the Canadian provinces and territories from 1871 to 1991, in 10-year increments. The animation control panel lets you dynamically view the datasets across a range of time. Animation operation is explained in "Sliders Controlling Independent Dimensions" on page 140.

- *population.europe.mapviz, population.europe.data, europe.countries.hierarchy,* and *europe.countries.gfx*
  When graphically displayed, this shows the 1992 population and population density of countries in Western and Central Europe.
- *population.usa.mapviz***,** *population.usa.data, usa.states.gfx,* and *usa.states.hierarchy*
  When graphically displayed, this shows the population and population density of the United States from 1770 to 1990. The animation controls let you dynamically view population and density changes across time.
- *population.usa.cities.mapviz, population.usa.cities.data, usa.states.gfx, usa.states.hierarchy,* and *usa.cities.gfx* and *usa.cities.hierarchy*
  The *usa.states.gfx* file specifies the United States, which is displayed as a background. The *usa.cities.gfx* file specifies the location of the cities on this background. The *.data* file specifies the population of each city.

  This sample graphically displays the population of the 48 largest U.S. cities from 1950 to 1990. No data has been mapped to the colors. The animation controls let you dynamically view changes across time.
- *perhouse.perage.mapviz***,** *perhouse.perage.data, usa.states.gfx,* and *usa.states.hierarchy*
  This sample graphically displays consumer household spending data from July-August 1988 to May-June 1991. Color is mapped to the gender of the spending household member; height represents the average dollar spent per household for a given time period and age group. This data has two independent dimensions: time and age. The highest spending is indicated in the summary window (see “The Summary Window” on page 143) by the areas with the greatest color density, namely “May-June 1989 (Age: 30-39)” and “May-June 1990 (Age: 30-39).”
- *telecom.mapviz***,** *telecom.data, usa.cities.lines.gfx, usa.cities.lines.hierarchy***,** *usa.states.gfx,* and *usa.states.hierarchy*
  This sample graphically displays a flat map with arched lines on it. These lines connect two endpoints. The lines can have variable width and color. In this example, the widths and colors are random; however, they could relate to the volume and duration of the connections between the endpoints.

- *fasta.m.data*, *fasta.m.mapviz*, *fasta.m.gfx*, and *fasta.m.hierarchy*

  The data file for this example contains the partial results of a full biological sequence comparison between two complete genomes (courtesy of Dr. Tom Flores, European Bioinformatics Institute). When graphically displayed, scientists can quickly identify and locate the regions of similarity between the two genomes. The ability to display such large amounts of information in a visual data exploration method such as this could be extended to include much more information about the individual genomes. Scientists could explore this data more easily and thereby perhaps better understand the function and purpose of the similar genetic sequences.

  In this example, the "map" is the circular-shaped genome of a biological organism called *Mycoplasma genitalium* (MG). The MG genome is divided into 500 equal segments, each representing a 1000-nucleotide sequence in the genome. The slider selects one of the segments of the second genome, called *Haemophilus influenzae* (HI), for cross-comparison between the two genomes. The Summary Window in the Animation Control Panel indicates which segments show the greatest similarities, and you can move the slider to examine those particular segments of interest. The bar heights and colors on the "map" therefore indicate the relative similarity of each MG segment to each HI segment, where higher bars correspond to greater measures of similarity. This similarity is measured by the "Reciprocal Evalues," which ranges from 0.0 to 1.0.

Chapter 6

Using the Scatter Visualizer

This chapter discusses the features and capabilities of the Scatter Visualizer. It provides an overview of this database visualization tool, then explains the Scatter Visualizer's functionality when working with the

- main window
- external controls
- pulldown menus

Finally, it lists and describes the sample files provided for this tool.

Overview of Scatter Visualizer

The Scatter Visualizer lets you visually analyze relationships among several variables (see Figure 6-1), either statically or by animation. This analysis is done using

- a three-dimensional landscape
- an animation control panel that includes a two-dimensional slider
- graphical objects, called *entities*, that can be animated in the three-dimensional landscape

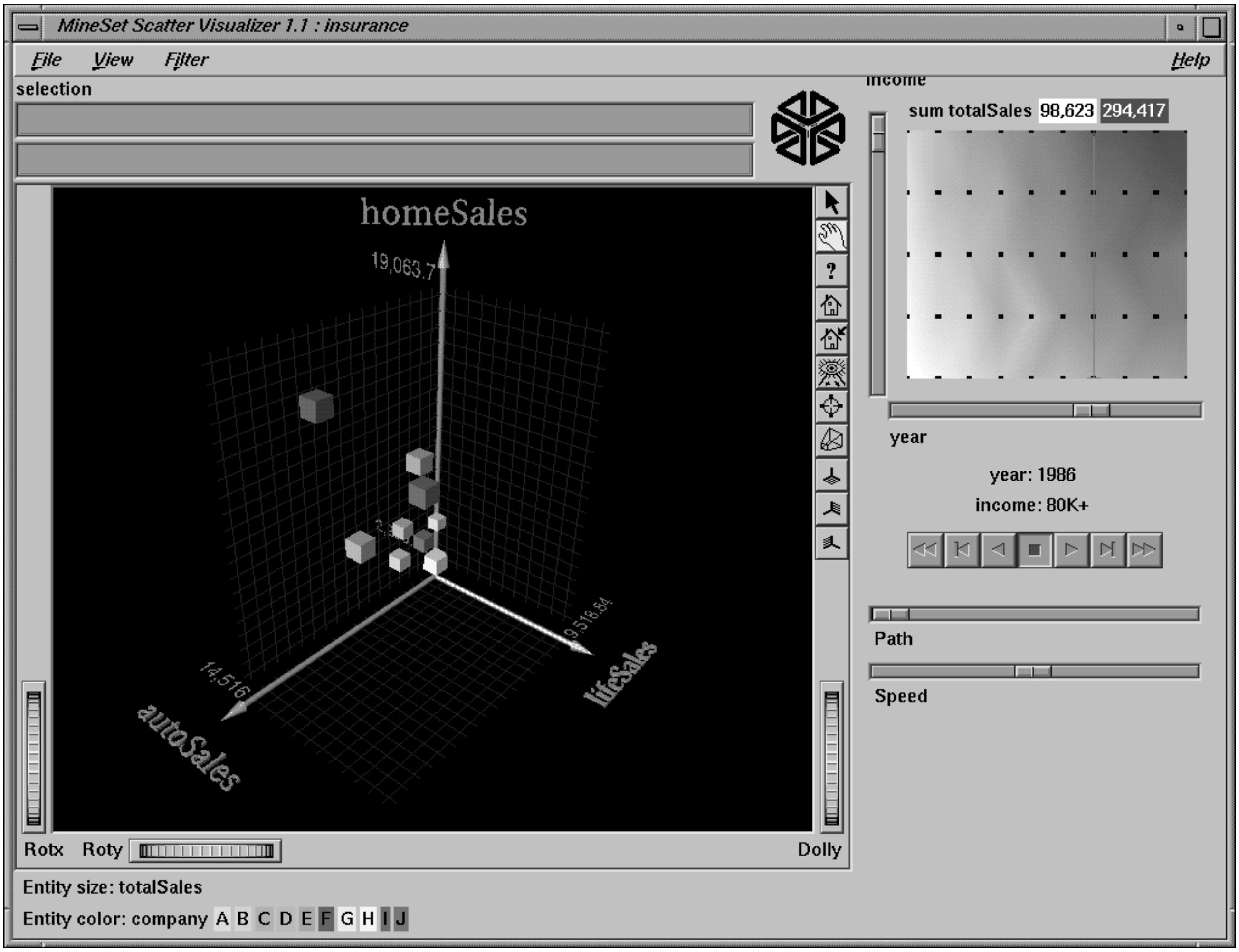


Figure 6-1 Sample Scatter Visualizer Screen

The Scatter Visualizer lets you visualize your data by mapping each record, or row, in the dataset to an entity in the three-dimensional landscape. Variables in the data can be mapped to the sizes, colors, and positions of the entities. Also, you can map one or two numeric variables to the sliders in the animation control panel. If the variables mapped to sizes, colors, or positions of the entities depend on the variables mapped to sliders, the sliders can be used to drive an animation. For example, if the data represents the sales of several companies over time. If the time variable is mapped to a slider and the sales variable is mapped to size, then the entities grow or shrink as the time slider is animated.

After you create a visualization of your data, the Scatter Visualizer lets you analyze the data in various ways. The animation control panel lets you trace animation paths in one or two dimensions. By playing back the path you created, you can watch the size, color, and motion of the entities for trends or anomalies. In the three-dimensional landscape, you can orient the display to emphasize particular dimensions or a point of view. The Scatter Visualizer lets you scale the values of variables to give them greater emphasis. Also, you can filter the display to show only those entities meeting certain criteria.

File Requirements

The Scatter Visualizer requires the following files:

- A data file, consisting of rows of tab-separated fields. This file is easily created using the Tool Manager (see Chapter 3). If you are generating this file yourself, see Appendix C, “Creating Data and Configuration Files for the Scatter Visualizer” for the required file format.

  You can generate data files by extracting data from a source (such as a database) and formatting it specifically for use by the Scatter Visualizer. Data files have user-defined extensions (the sample files provided with the Scatter Visualizer have a *.data* extension).

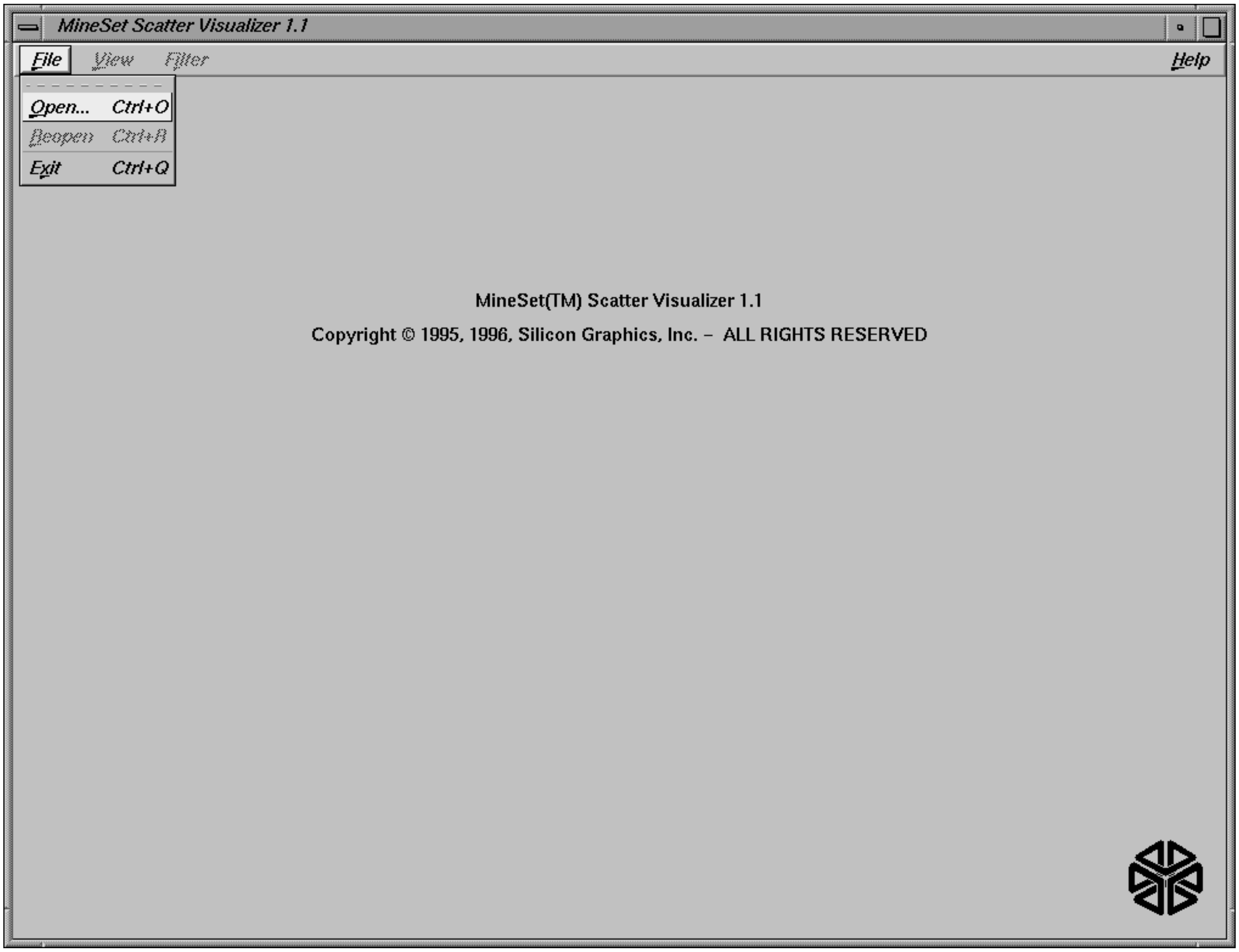
- A configuration file, describing the format of the input data and how it is to be displayed. The Tool Manager can create this file (see Chapter 3), or you can use an editor (such as jot, vi, or Emacs) to produce this file yourself (see Appendix C, “Creating Data and Configuration Files for the Scatter Visualizer”).

  Configuration files must have a *.scatterviz* extension. When starting the Scatter Visualizer, or when opening a file, you must specify the configuration file, not the data file.

Starting the Scatter Visualizer

There are five ways to start the Scatter Visualizer:

- Use the Tool Manager to configure and start the Scatter Visualizer. (See Chapter 3 for details on most of the Tool Manager's functionality, which is common to all MineSet tools; see "Configuring the Scatter Visualizer Using the Tool Manager" on page 162 for details about using the Tool Manager in conjunction with the Scatter Visualizer.)
- Double-click the Scatter Visualizer icon, which is in the MineSet page of the icon catalog. The icon is labeled *scatterviz*. Since no configuration file is specified, the start-up screen requires you to select one by using File > Open.



MineSet(TM) Scatter Visualizer 1.1

Copyright © 1995, 1996, Silicon Graphics, Inc. – ALL RIGHTS RESERVED

Figure 6-2 Scatter Visualizer Start-Up Screen With File Pulldown Menu Selected

Starting the Scatter Visualizer without specifying a configuration file causes the main window to show the copyright notice and license agreement for this tool. Only the File and Help pulldown menus can be used. For the main window to be fully functional, open a configuration file by selecting File > Open (Figure 6-2).

- If you know what configuration file you want to use, double-click the icon for that configuration file. This starts the Scatter Visualizer and automatically loads the configuration file you specified. This works only if the configuration filename ends in *.scatterviz* (which is always the case for configuration files created for the Scatter Visualizer via the Tool Manager).
- Drag the configuration file icon onto the Scatter Visualizer icon. This starts the Scatter Visualizer and automatically loads the configuration file you specified. This works even if the configuration filename does not end in *.scatterviz*.
- Start the Scatter Visualizer from the UNIX shell command line by entering this command at the prompt:

  ```
  scatterviz [ configFile ]
  ```

  configFile is optional and specifies the name of the configuration file to use. If you don't specify a configuration file, you must use File > Open to specify one (see Figure 6-2).

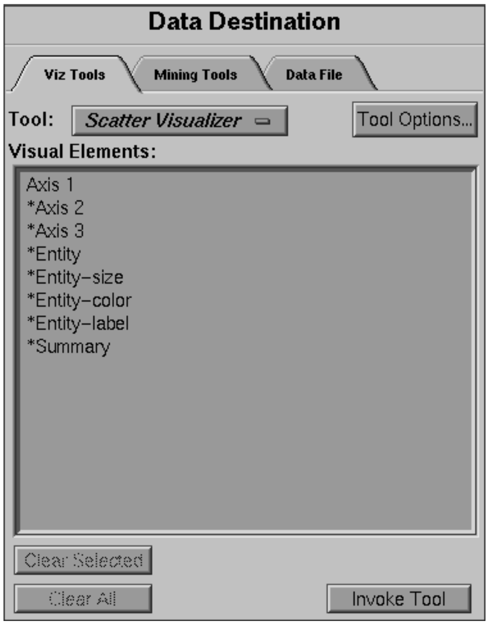
Configuring the Scatter Visualizer Using the Tool Manager

This section describes how the Scatter Visualizer can be configured using the Tool Manager. Although the Tool Manager greatly simplifies the task of configuring the Scatter Visualizer, you can construct a configuration file manually for this tool using a text editor (see Appendix C, “Creating Data and Configuration Files for the Scatter Visualizer”).

Note that the steps required to connect to a data source are described in Chapter 3.

Selecting the Scatter Visualizer Tool

Select the *Viz Tools* tab in the Data Destination panel of the Tool Manager's main screen (Figure 6-3). From the popup list of tools, select *Scatter Visualizer*. The mapping requirements for the Scatter Visualizer are displayed in the window on the right side of this panel. Items in the Visual Elements list that are preceded by an asterisk are optional.



Tool: Scatter Visualizer

Visual Elements:

Axis 1
*Axis 2
*Axis 3
*Entity
*Entity–size
*Entity–color
*Entity–label
*Summary

Figure 6-3 Data Destination Panel With Scatter Visualizer Selected

- Axis 1—This lets you assign to the first axis in the Scatter Visualizer's main window the data you want represented. Assigning data to this axis is required. However, this alone does not produce a useful display. By assigning data to Axis 2, you can create an XY chart. Assigning data to all three axes produces a 3D chart.
- Entity—This mapping is a placeholder that is currently not in use. It does not affect the display. You can also map columns to the size, color, and labels of the entities.
- Summary—This is the value mapped to the summary column, if you have a slider. It determines the color of the slider's background.

Mapping Requirements to Columns

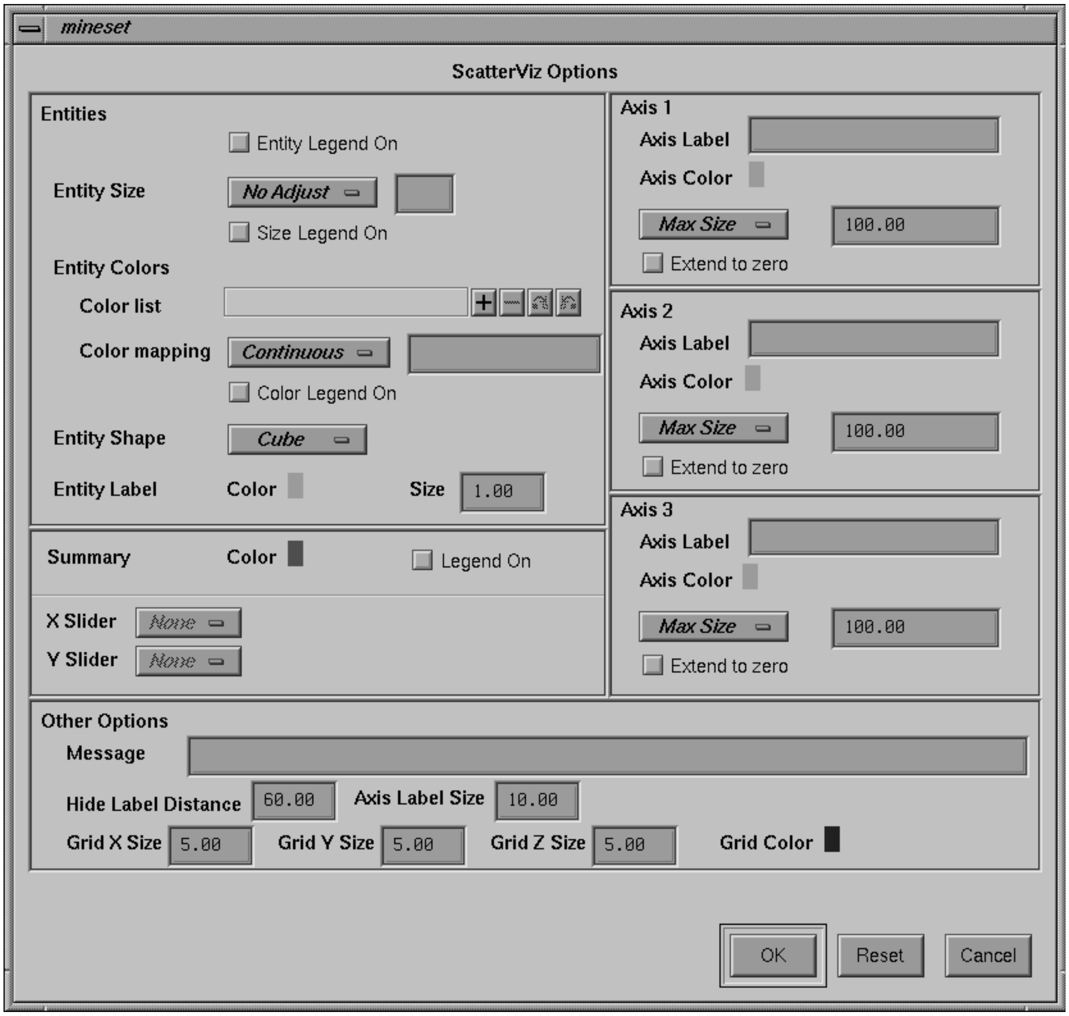
You can map requirements to columns by selecting a column name in the Current Columns window of the Table Processing panel, then selecting a category in the Visual Elements window.

Undoing Mappings

To undo a specific mapping, select that mapping in the Requirements window, then click the *Clear Selected* button. To undo all mappings, click the *Clear All* button.

Specifying Tool Options

Clicking the *Tool Options* button causes a new dialog box to be displayed (Figure 6-4). This lets you change some of the Scatter Visualizer options from their default values.



mineset

ScatterViz Options

Entities

□ Entity Legend On

Entity Size: No Adjust

□ Size Legend On

Entity Colors

Color list

Color mapping: Continuous

□ Color Legend On

Entity Shape: Cube

Entity Label Color Size 1.00

Summary Color □ Legend On

X Slider: None

Y Slider: None

Axis 1
Axis Label
Axis Color
Max Size 100.00
□ Extend to zero

Axis 2
Axis Label
Axis Color
Max Size 100.00
□ Extend to zero

Axis 3
Axis Label
Axis Color
Max Size 100.00
□ Extend to zero

Other Options

Message

Hide Label Distance 60.00 Axis Label Size 10.00

Grid X Size 5.00 Grid Y Size 5.00 Grid Z Size 5.00 Grid Color

Figure 6-4 Scatter Visualizer's Options Dialog Box

The Scatter Visualizer's Options dialog box has four basic options blocks:

- Entities
- Axes
- Summary
- Other

Entity Options

This option lets you specify a number of characteristics for the entities that the Scatter Visualizer then graphically displays.

- Entity Legend On—Lets you determine whether the entity legend is displayed or hidden.
- Entity Size—Lets you scale the entity to a max size, a scale size, or a default (no adjustment). You also can specify whether the legend for entity size is displayed or hidden.
- Entity Shape—Lets you choose a visual representation for the entities: cubes, bars, or diamonds.
- Entity Colors—Lets you control the colors in which entities are displayed. You can
  - specify the list of colors to use
  - specify the kind of mapping
  - map the list of colors to a list of values
  - specify whether the legend for color is displayed or hidden
  - map colors to entities

To use these Colors options, you must have mapped a column to the *Entity-color requirement of the Data Destination panel. See “Color Options for the MineSet Visualizers” in Chapter 3 for a more detailed explanation of how to choose and change colors.

Color list to use—You can specify the color list using the + button next to the color list label. This brings up a color editor that lets you specify a color to be added to the list.

Color mapping—You can specify whether the color change that is shown in the graphic display is *Continuous* or *Discrete*. If you choose Continuous (see), the color values shift gradually between the colors entered in the *Color list to use* field as a function of the values that are mapped to those colors in the *Color mapping* field. describes the discrete function.

The field to the right of the popup button lets you enter specific values for mapping the colors. If you do not specify any mapping values, the range of values in the color variable is used.

Example 1:

If you

- used the Color Browser to apply red and green to bars
- selected *Continuous* for the *Kind of mapping*
- entered the values `0 100`

then the display shows all entities with values less than or equal to 0 as completely red, those as greater than or equal to 100 as completely green, and those between 0 and 100 as shadings from red to green.

Example 2:

If you

- used the Color Browser to apply red and green to entities
- selected *Discrete* for the *Kind of mapping*
- entered the values `0 50`

then the display shows all entities with values of less than 50 in red, and all those with values greater than or equal to 50 in green.

Entity Label Color—You can modify a label color by clicking on it. This causes the Color Choose dialog box to appear, which lets you implement your color changes.

Summary Options

Summary options let you specify what color to use for the Summary window. You can also specify whether the summary legend, which indicates what the values are, is displayed or hidden.

If you have an array of values, you can specify an X or Y slider. The popup buttons next to these options provide a list of available keys, and let you specify which to use as sliders.

Axis Options

The Axis options let you specify the following, for each axis:

- A label. (If you leave this box blank, the Scatter Visualizer defaults to using the column names for each axis.)
- A size type for each axis. (This can be *Max Size*, *Scale Size*, or *No Adjustment*.)
- A size value.
- Whether the axis should be extended to include the value 0.

Other Options

The Other Options, at the bottom of the dialog box, include the following fields:

- *Message*—Lets you specify the message displayed when an entity is selected. For a listing and description of format types that can be entered in this field, see the "Message Statement" section in Appendix C, "Creating Data and Configuration Files for the Scatter Visualizer."
- *Entity Label Size*—Controls the size of the entity labels. A smaller number decreases the size, a larger one increases it.
- *Hide Label Distance*—Controls the distance at which entity labels become invisible. Smaller distances might improve performance, but the labels disappear more quickly. The higher the number, the greater the distance at which labels are hidden.

- *Axis Label Size*—This controls the size of the axis labels. A smaller number decreases the size, a larger one increases it.
- *Grid Color*— Lets you modify a grid color by clicking on it. This causes the Color Choose dialog box to appear, which lets you implement your color changes.
- *Grid (X, Y, Z) Size*—Lets you specify the spacing between grid lines for the respective axis. A smaller number decreases the size, a larger one increases it.

Resetting the Tool Options

If you want to reset the values of all options to their default values, click the *Reset Options* button.

Saving the New Tool Options

Once you have finished making changes to the Tool Options dialog box, click *OK* to return the Tool Manager's main screen.

To have these changes take effect, you must save the configuration file again by clicking the *Save Config File* button, described below. Note that if you give the name of a previously saved configuration file, the new file overwrites the saved file. After you have saved your new configuration file, click the *Invoke Tool* button again to see the results of your changes.

Saving Scatter Visualizer Settings

The Tool Manager stores information for the Scatter Visualizer in several files, all sharing the same prefix:

- *<prefix>.scatterviz.data* contains data.
- *<prefix>.scatterviz.schema* describes the data file.
- *<prefix>.scatterviz* contains information needed by the Scatter Visualizer.
- *<prefix>.mineset* contains all the information needed to create the other files.

To specify a prefix, use the *Save...* button in the lower right of the Viz Tools panel, or the Save... menu option in the File menu. If you do not specify a prefix, the default *untitled* is used.

When you use the *Invoke Tool* button, the *.data*, *.schema*, and *.scatterviz* files are updated, if necessary.

Invoking Scatter Visualizer

To see Scatter Visualizer graphically represent your data, click the *Invoke Tool* button at the bottom of the Data Destination panel.

Null Handling in the Scatter Visualizer

The Scatter Visualizer uses special representations when fields with unknown data values, or nulls, are mapped to visual attributes. (For a discussion of null values, see Appendix G, “Nulls in MineSet.”) When a null value is mapped to an entity’s size, the entity is drawn as the outline of a cube. When a null value is mapped to an entity’s color, it is drawn in dark grey. When a null value is displayed in the Selection Window or “Pointer is Over” area, it is shown as a question mark (?). (The Selection Window and “Pointer is Over” areas are discussed in the “Select Mode” section.)

If a null value is mapped to the *x*, *y*, or *z* position of an entity, the result depends on the Show Entities with Null Positions option under the View Menu (see “The View Menu” on page 183). If the option is set, the entity is shown below the range of the corresponding axis. If the option is not set, the entity is not shown.

Working in the Scatter Visualizer’s Main Window

If you started the Scatter Visualizer without specifying a configuration file, the main window shows the copyright notice and license agreement for the Scatter Visualizer. Only the File and Help pulldown menus can be used. For the main window to show all menus and controls, open a configuration file. Use File > Open (Figure 6-2) to see a list of configuration files.

When a valid configuration file has been selected, the 3D landscape it specifies is visible. For example, selecting *company.scatterviz* gives results as shown in Figure 6-5.

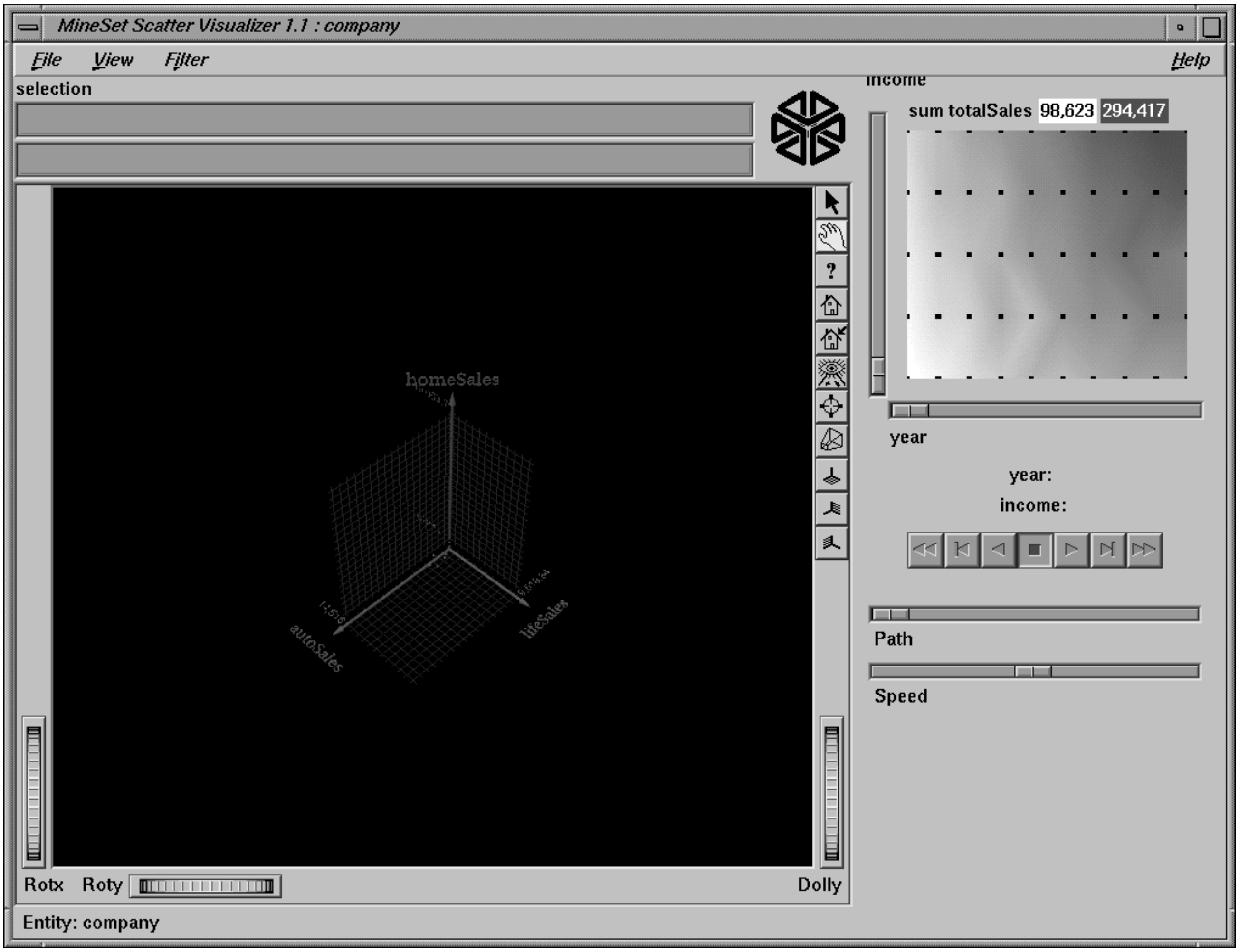


Figure 6-5 Initial View When Specifying company.scatterviz

This shows the sales of life insurance, auto insurance, and home insurance with respect to income brackets over time.

Viewing Modes

The two modes of viewing are *grasp* and *select*. To toggle between these modes, press the Esc key or click the appropriate cursor button adjacent to the top-right of the viewing area. (These and the rest of the buttons are described later in this chapter.)

Grasp Mode

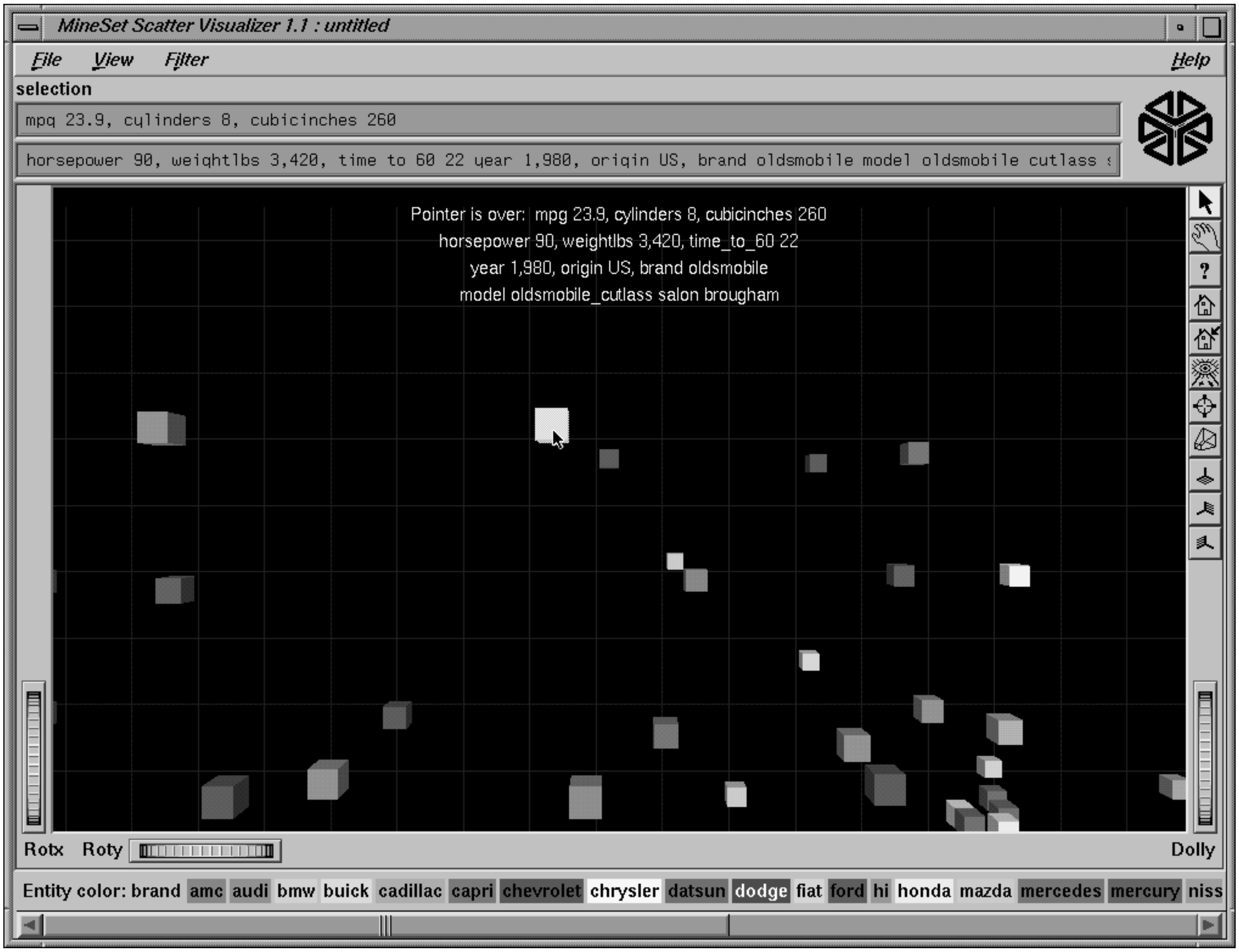
In *grasp* mode, the cursor appears as a hand. This mode supports panning, rotating, and scaling the scene's size in the main window.

- To pan the display, press the middle mouse button and drag it in the direction you want the display panned.
- To rotate the display, press the left mouse button and move the mouse in the direction you want to rotate. (Also see the thumbwheel controls *Rotx* and *Roty*, described in "Thumbwheels" on page 176.)
- To move the viewpoint forward, press the left and middle mouse buttons simultaneously and move the mouse downwards. To move the viewpoint backward, press the left and middle mouse buttons simultaneously and move the mouse upwards. This is equivalent to the functions provided by the Dolly thumbwheel.

Select Mode

In *select* mode, you can highlight an object by positioning the cursor over that object. Information about that object then appears at the top of the view area, under the "Pointer is over:" label (Figure 6-6). This information remains visible in the window only as long as the pointer cursor remains over the object. If you position the pointer cursor over an object and click the left mouse button, that same information appears in the Selection Window, which is above the main window, under the "Selection" label.

This Selection information remains visible until another object is selected, or you click the black background. Using the mouse, you can cut and paste this selection information into other applications, such as reports or databases.


MineSet Scatter Visualizer 1.1 : untitled
File
View
Filter
Help
selection
Rotx
Roty
Dolly
Entity color: brand amc audi bmw buick cadillac capri chevrolet chrysler datsun dodge fiat ford hi honda mazda mercedes mercury niss

mpg 23.9, cylinders 8, cubicinches 260

horsepower 90, weightlbs 3,420, time to 60 22 year 1,980, origin US, brand oldsmobile model oldsmobile cutlass s

Pointer is over: mpg 23.9, cylinders 8, cubicinches 260
horsepower 90, weightlbs 3,420, time_to_60 22
year 1,980, origin US, brand oldsmobile
model oldsmobile_cutlass salon brougham

Figure 6-6 Cursor Over an Object

The information is displayed when the cursor is over the object.

External Controls

Several external controls surround the main window, including buttons and thumbwheels. This section describes each type of control.

Buttons

At the top right of the image area are 11 buttons (see Figure 6-7).

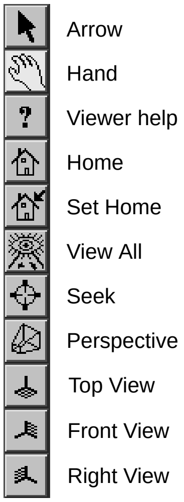


Figure 6-7 Detail View of Top Right Buttons

- *Arrow* puts you in select mode, which lets you highlight entities in the main window. When in this mode, the cursor shape is an arrow.
- *Hand* puts you in grasp mode, which lets you rotate, zoom, and pan the display in the main window. When in this mode, the cursor shape is a hand.
- *Viewer help* brings up a help window describing the viewer itself.

- *Home* takes you to a designated location. Initially, this is the first viewpoint shown after invoking the Scatter Visualizer and specifying a configuration file. If you have been working with the Scatter Visualizer and have clicked the *Set Home* button, then clicking *Home* returns you to the viewpoint that was current when you last clicked *Set Home.*
- *Set Home* makes your current location the Home location. Clicking the *Home* button returns you to the last location where you clicked *Set Home*.
- *View All* lets you view the entire graphic display, without changing the angle of view you had before clicking on this option. To get an overhead view of the scene, rotate the camera so that you are looking directly down on the entities, then click the *View All* button.
- *Seek* takes you to the point or object you click after selecting this button.
- *Perspective* is a toggle button that lets you view the scene in 3D perspective (closer objects appear larger, farther object appear smaller). Clicking this button again turns 3D perspective off.
- *Top View* lets you view the scene from the top.
- *Front View* lets you view the scene from the front.
- *Right View* lets you view the scene from the right side.

Thumbwheels

Three thumbwheels appear around the lower part of the main window border (see Figure 6-8). They let you dynamically move the viewpoint.

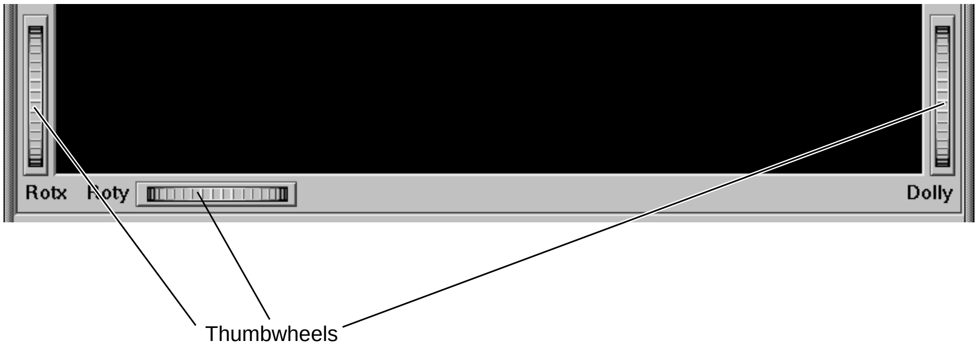


Figure 6-8 View of Lower Half of Window With Thumbwheels

- The vertical thumbwheel *Rotx* (rotate about the x axis), on the left, rotates the display up and down.
- The horizontal thumbwheel *Roty* (rotate about the y axis), at the bottom left, rotates the scene in the main window around its centerpoint left and right.
- The vertical thumbwheel *Dolly*, on the right, moves the viewpoint forward and backward. Note that as you use the Dolly thumbwheel to magnify the scene in the main window, additional detail can appear. If *Perspective* is off, the Dolly thumbwheel becomes the Zoom thumbwheel.

The Animation Control Panel

The animation control panel, which appears to the right of the main window, consists of a summary window, with up to two adjacent sliders, an information field, animation buttons, and animation sliders.

Sliders Controlling Independent Dimensions

The number of sliders appearing adjacent to the summary window is dependent on the dataset displayed in the Scatter Visualizer's main window. Datasets can have two, one, or no independent dimensions.

Datasets With Two Independent Dimensions

If the dataset has two dimensions of independently varying data (such as *company.scatterviz*), the controls to the right of the main graphics window become visible (see Figure 6-9).

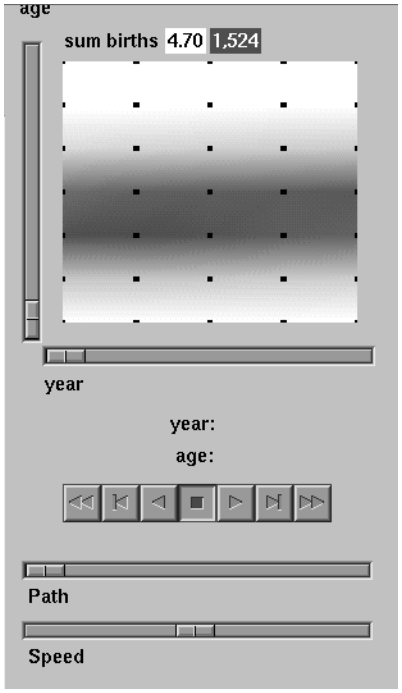


Figure 6-9 Animation Control Panel With Summary Window and Both Slider Controls

To the right of the main window are the 2D summary window and slider controls. The summary window has a horizontal slider below it for selecting data points of the first independent dimension, and a vertical slider to the left for selecting data points of the second independent dimension. The horizontal slider's dimension is identified by a label below it. The vertical slider's dimension is identified by a label above it.

Datasets with One Independent Dimension

For datasets with one independent dimension (such as *store-type.scatterviz*), only the slider below the summary window appears, and the summary window is compressed (see Figure 6-10). This slider's dimension is identified by a label below it.

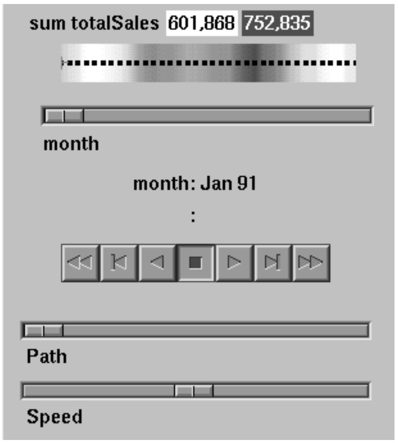


Figure 6-10 Animation Control Panel With Summary Window and One Slider Control

Datasets With No Independent Dimension

For datasets with no independent dimensions (such as *brand.scatterviz*), no slider control appears (see Figure 6-11).

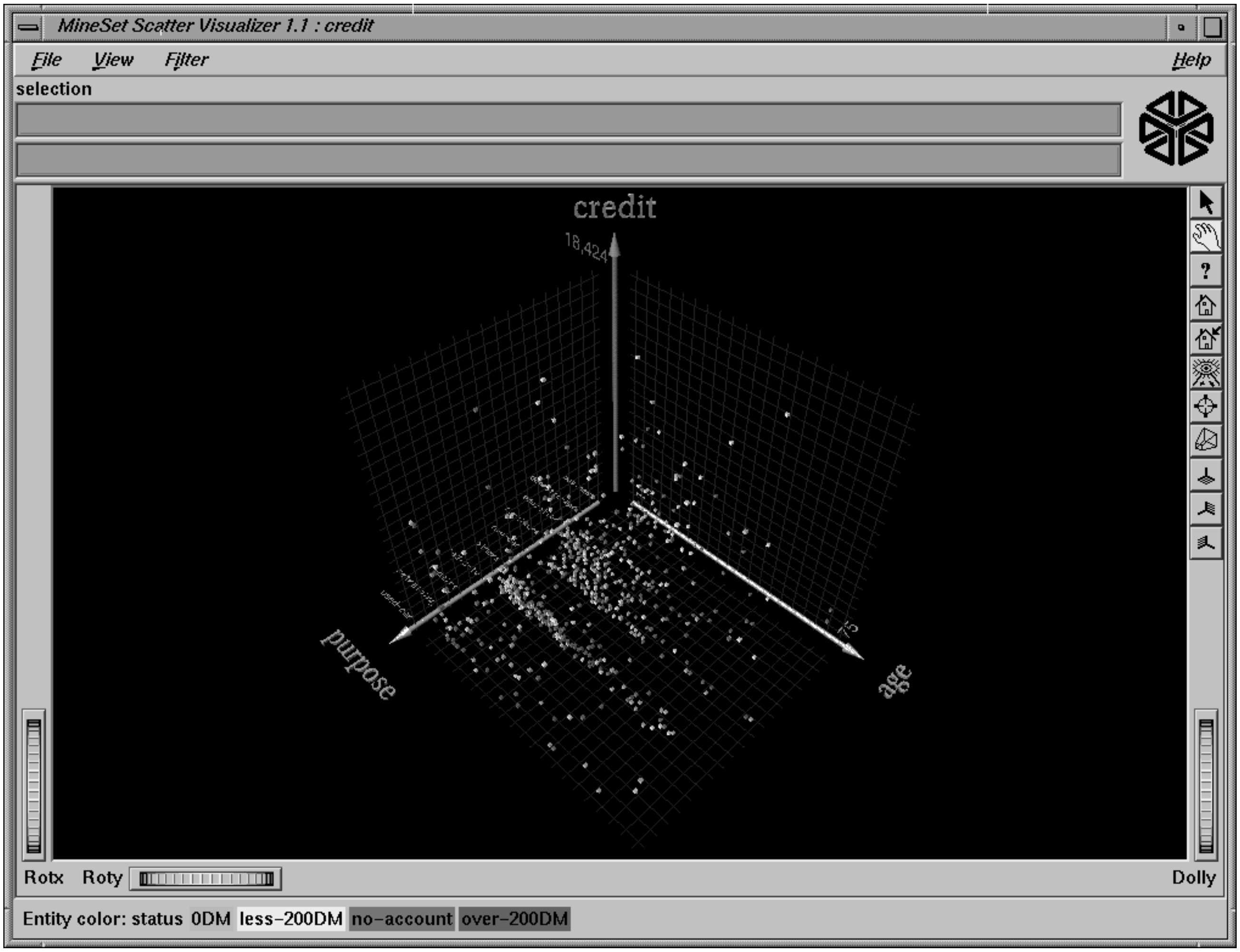


Figure 6-11 Scatter Visualizer Without Independent Dimension or An Animation Control Panel

The Summary Window

The summary window provides a 2D representation of the aggregation of values that the main window displays in 3D. The whiter the areas of the summary window, the lower the total values represented by the entities in the main window. The greater the color density in areas of the summary window, the higher the total of those values. The density of these colors in the summary window provides a summary of the data across the one or two independent dimensions in the dataset.

By default, the summary window also contains a set of black dots, evenly spaced across the one or two dimensions of data. These dots indicate the precise positions of the discrete datapoints. You can turn off these black dots using the View > Show Data Points menu option.

Color Density Examples in the Summary Window

After opening the *company.scatterviz* file, for example, the 2D summary window shows a color range from white (on the left) to red (on the right). White corresponds to a low sales volume; red represents a higher aggregate sales volume. In this example, the greater the density of red, the higher the total sales of life, auto, and home insurance.

Creating a Path in the Summary Window

If the dataset loaded into the Scatter Visualizer has at least one independent dimension, it is possible to view all or any part of that dataset via animation. This is done by first creating a path in the summary window (this path connects a sequence of data points), then activating the animation controls described in the next section.

The three ways to draw a path in the summary window are as follows:

- Define a starting point by clicking and holding down the left mouse button, then draw an arbitrary path by dragging the cursor over the window. End the path by releasing the left mouse button.
- Define a starting point by clicking the left mouse button, then define an endpoint by moving the cursor to another part of the window and clicking the middle mouse button. A line appears between those two points. To add more line segments, continue with repeated middle mouse clicks.
- Define a starting point by clicking the left mouse button, then drag one of the independent dimension sliders, thus drawing a straight line along this dimension. If there are two sliders, use of the second slider causes a straight line to be drawn along the axis controlled by this second slider.

Animation Buttons and Sliders

The seven VCR-like buttons and two sliders (Path and Speed) below the 2D summary window let you control the animation.

Animation Buttons

Once a path is drawn in the summary window (see “Creating a Path in the Summary Window,” above), you can use the VCR-like buttons to control animation along this path. The middle *Stop* button is highlighted in blue, indicating an initial state. Use the adjacent *Play Forward* button (to the right of *Stop*) or *Play Reverse* (to the left) to begin simple movement along the drawn path in a forward or reverse direction. (*Forward* and *Reverse* are defined by the sequence that the path was drawn, not by the left-to-right or right-to-left movement.)

To stop and restart the animation, click the *Stop* button, then use the *Play Forward* or *Reverse* button again. Note that when you stop, the animation continues in the current direction until the position falls upon a discrete data point.

Adjacent to the *Play* buttons are the *Single-Step* buttons, as well as *Forward* and *Reverse*. Clicking on one of these buttons changes the current path position to the next discrete data point.

On the outside are the *Fast Forward* and *Fast Reverse* buttons. Clicking one of these buttons while in *Stop* state changes the path position to the end (for *Forward*) or to the beginning (for *Reverse*) of the path. Clicking a *Fast* button when in *Play* state increases the animation speed.

Animation Sliders

While animation is stopped, you can move the Path slider to reset the position along the path. Note that when you use the Path slider, the cursor in the summary window moves across the drawn path, and the 1D sliders (below and to the left of the drawing area) move consistently with the cursor position. Then use the *Play* or *Reverse* button to restart the animation from the newly specified point. You can drag the Path slider to an arbitrary position between discrete data points; however, when you release the slider, the path position changes to the nearest discrete data point.

Use the Speed slider to adjust the speed of the animation along the path.

Data Points and Interpolation

As animation proceeds, the variables mapped to size, color, and axes (positions) in the Scatter Visualizer changes smoothly. However, the information displayed in the “Selection:” message box and the “Pointer is over:” field show only the data values of the nearest discrete data position; they do not show interpolated data values.

The animation is producedin the following manner: Assume you have data for 10 years, on a per-year basis (that is, 10 data values) and that these correspond to the size of one entity in the Scatter Visualizer. Assume further that the years are 1991 to 2000, the size for 1991 is 20, and the size for 1992 is 40. As you move the year slider from 1991 to 1992, the size changes by being uniformly interpolated between 20 and 40. For example, midway between 1991 and 1992, the size is 30. As you approach 1992, the size approaches 40. However, you cannot stop an animation between discrete data points, and you cannot drag the Path slider to a stationary position between discrete data points.

The data points in the summary window represent the slider positions corresponding to the actual data from the data file. For example, sizes 20 and 40 are representations of actual data, but size 30 is not. In this example, there would be data points in the summary window at the slider positions corresponding to each year.

Note that not all variables are required to vary with a slider. If there are two sliders, some variables can vary with only one of the sliders, while other variables vary with both.

Pulldown Menus

Four pulldown menus let you access additional Scatter Visualizer functions. These are labeled File, View, Filter, and Help. If you start the Scatter Visualizer without specifying a configuration file, only the File and the Help menus are available.

The File Menu

The File menu lets you open a new configuration file, reload the current configuration and data files, and exit the Scatter Visualizer.

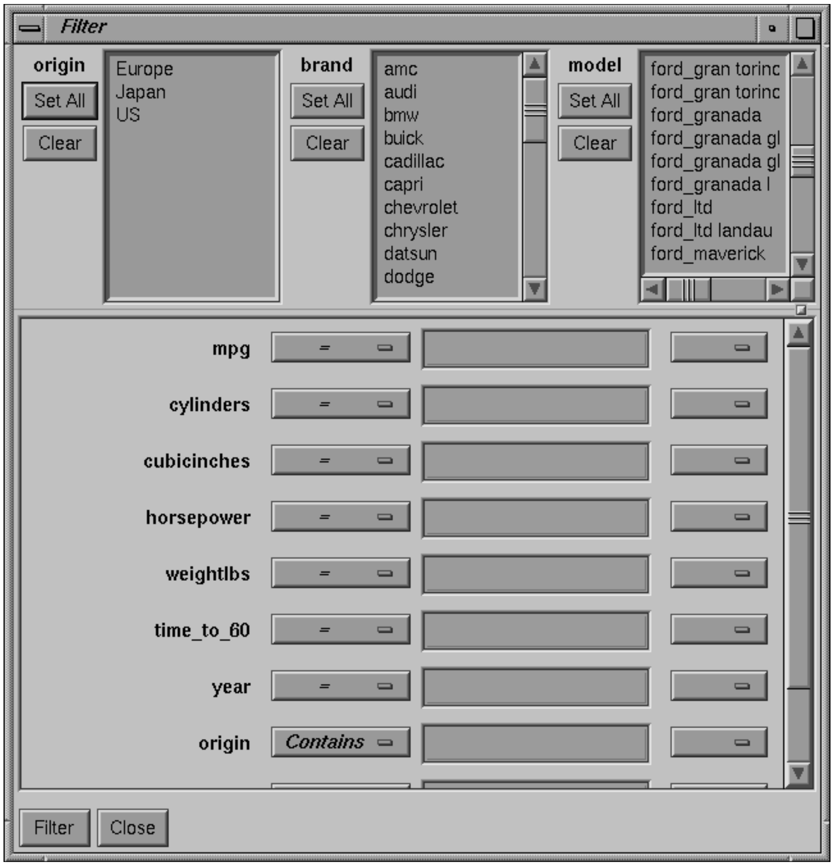
The View Menu

The View menu lets you control certain aspects of what is shown in the Scatter Visualizer window. This menu contains three options:

- Show Window Decoration lets you hide or show the external controls around the main window.
- Show Entities with Null Positions lets you hide or show entities that have null or unknown position values along one or more axes.
- Show Animation Panel lets you show or hide the animation control panel. This menu item is disabled for datasets with no independent dimension.
- Show Data Points lets you show or hide the data points in the summary window. This option is disabled for datasets with no independent dimensions.

The Filter Menu

The Filter menu brings up a filter panel (Figure 6-12) that lets you reduce the number of entities displayed in the main viewing area, based on one or more criteria. You can use the filter panel to fine-tune the display, emphasize specific information, or simply shrink the amount of information displayed. *Set Landscape to Filter* lets you specify whether the landscape in the main window covers the entire dataset or just the filtered data.



Europe
Japan
US

amc
audi
bmw
buick
cadillac
capri
chevrolet
chrysler
datsun
dodge

ford_gran torinc
ford_gran torinc
ford_granada
ford_granada gl
ford_granada gl
ford_granada l
ford_ltd
ford_ltd landau
ford_maverick

Figure 6-12 Scatter Visualizer Filter Panel

The filter panel has two panes. The top pane lets you filter based on string variables. To select all values of a variable, click *Set All*. To clear the current selections, click *Clear*. To select a value, click it. To deselect a value, simply click it again.

The bottom pane lets you filter based on the values of both string and numeric variables. Only variables whose values do not change as you navigate the slider can be used in filtering.

To filter numeric values, enter the value, and select a relational operation (=, !=, >, <, >=, <=). To filter alphanumeric values, enter the string. You can use any of three types of string comparisons:

- Contains indicates that it contains the appropriate string. For example, California contains the strings Cal and forn.
- Equals requires the strings to match exactly.
- Matches allows wildcards:
  - An asterisk (*) represents any number of characters.
  - A question mark (?) represents one character.
  - Square braces ([ ]) enclose a list of characters to match.

  For example, California matches Cal*, Cal?fornia, and Cal[a-z]fornia.

In some cases (usually associated with binning in the Tool Manager), an option menu of values appears, instead of a text field. To ignore that variable, select *Ignored* in the *Option* menu. You can use relational operators (such as >=) with these options. This means that the specified value as well as subsequent ones are selected.

In addition to numeric and string comparison operations, you can specify `Is Null`, which is true if the value is null.

To the right of each field is an additional option menu that lets you specify “And” or “Or” options. For example, you could specify “sales > 20 And < 40.” You can have any number of And or Or clauses for a given variable, but cannot mix And and Or in a single variable.

Click the *Filter* button to start filtering. If you press *Enter* while the panel is active, filtering starts automatically.

Click the *Close* button to close the panel.

The Help Menu

The Help menu provides access to five help functions (see Figure 6-13).

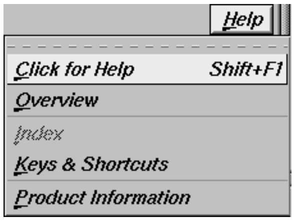


Figure 6-13 Scatter Visualizer Help Menu

- Click for Help turns the cursor into a question mark. Placing this cursor over an object in the Scatter Visualizer's main window and clicking the mouse causes a help screen to appear; this screen contains information about that object. Closing the help window restores the cursor to its arrow form and deselects the help function. The keyboard shortcut for this function is Shift+F1. (Note that it also is possible to place the arrow cursor over an object and press the F1 function key to access a help screen about that object.)
- Overview provides a brief summary of the major functions of this tool, including how to open a file and how to interact with the resulting view.
- Index provides an index of the complete help system. This option is currently disabled.
- Keys & Shortcuts provides the keyboard shortcuts for all of the Scatter Visualizer's functions that have accelerator keys.
- Product Information brings up a screen with the version number and copyright notice for the Scatter Visualizer.
- *MineSet User's Guide* invokes the Insight viewer with the online version of this manual.

Sample Configuration and Data Files

The provided sample data and configuration files demonstrate the Scatter Visualizer's features and capabilities. The following files are in the */usr/lib/MineSet/scatterviz/examples* directory:

- *company.data*
  This file contains fictitious sales data of several insurance companies in three product categories: life insurance, auto insurance, and home insurance. The data spans ten years (in increments of one year) and includes five income brackets (the customer's annual income).
- *company.scatterviz*
  This file specifies that the years form one slider dimension and the income brackets form the other slider. Sales of life insurance, auto insurance, and home insurance become the three dimensions in the Scatter Visualizer landscape. The color density in the slider summary window represents the total sales of all companies across all categories of insurance.
- *company-total.scatterviz*
  This file contains the same specifications as *company.scatterviz*, except that the size of each company is determined by the total sales of that company across all the categories of insurance.
- *company-life.scatterviz*
  This file contains the same specifications as *company.scatterviz*, except that the color of each object indicates the life insurance sales as a fraction of total sales.

- *store-type.data* and *store-type.scatterviz*
  These files show sales of various product groups by store type during a three-year period. The single independent variable for which a slider appears is time. Each entity represents a store type (such as Food Store, Drug Store, Service Station, and so forth). For each store type, the data file contains the total sales of several product groups, such as alcoholic beverages, cereal, and so forth. The data spans 36 months, in increments of one month.

  The configuration file uses the month as the single slider dimension. One axis is sales of alcoholic beverages, the other is sales of tobacco products. A third axis is not used.

  Note: The data file includes other categories. You can edit the configuration file to use other product categories for the axes (see Appendix C, "Creating Data and Configuration Files for the Scatter Visualizer").
- *brand.data* and *brand.scatterviz*
  These files show sales of several soft-drink brands in a variety of store types. In this dataset the brands form the entities, and the store types are associated with the axes. The total sales are mapped to the size of each brand. The color mapping is random. Since there are no independent variables, no slider is present.
- *cars.data* and *cars.scatterviz*
  These files show the weight, horsepower, model year, and acceleration of several car models.
- *people.data* and *people.scatterviz*
  These files show the height, weight, density, and cholesterol level of several people.
- *nl.births.data* and *nl.births.scatterviz*
  These files show birth patterns in the Netherlands. For each region, the population density, birth rate, and population are shown. The animation sliders are mapped to the age of the mother and the year.

See */usr/lib/MineSet/scatterviz/examples/README* for additional information on the files in that directory.

Chapter 7

Using the Rules Visualizer

This chapter discusses the components and capabilities of the Rules Visualizer. It first provides an overview of this data mining and visualization tool, then it explains this tool's functionality when working with the

- main window
- external controls
- pulldown menus

Finally, it lists and describes the provided sample files for these tools.

Overview of Rules Visualizer

The Rules Visualizer gives you the power to mine data by constructing, verifying, and graphically representing models of patterns in large databases. These patterns are expressed via association rules, which indicate the frequency of items occurring together in a database.

Discovering and graphically displaying association rules can be relevant to many enterprises, including supermarket inventory planning, shelf planning, and attached mailing in direct marketing.

The tool execution scenario described in Chapter 1 of this document (see Figure 1-1) is slightly modified for the Rules Visualizer. First, the "raw" data in your database must be converted into a specially formatted file that can be processed by the association rules generator part of the Rules Visualizer. When the association rules generator has processed this file, the results can be displayed by the rules visualizer part of this tool.

Thus, the Rules Visualizer consists of three operations:

1. Data conversion. The association data converter processes a “raw” data file and creates a file usable by the association rules generator.
2. Association rules generation. The data file created by the association data converter is processed by the association rules generator, which creates a file usable by the rules visualizer.
3. Rules visualization. This operation displays the generated association rules.

In addition to the input data and rules file requirements, each operation requires a configuration file that specifies operational parameters.

The sequence of actions by the user, at the user’s workstation, and at the host server is shown schematically in Figure 7-1. The phases indicated at the right of the illustration correlate to the operations listed above.

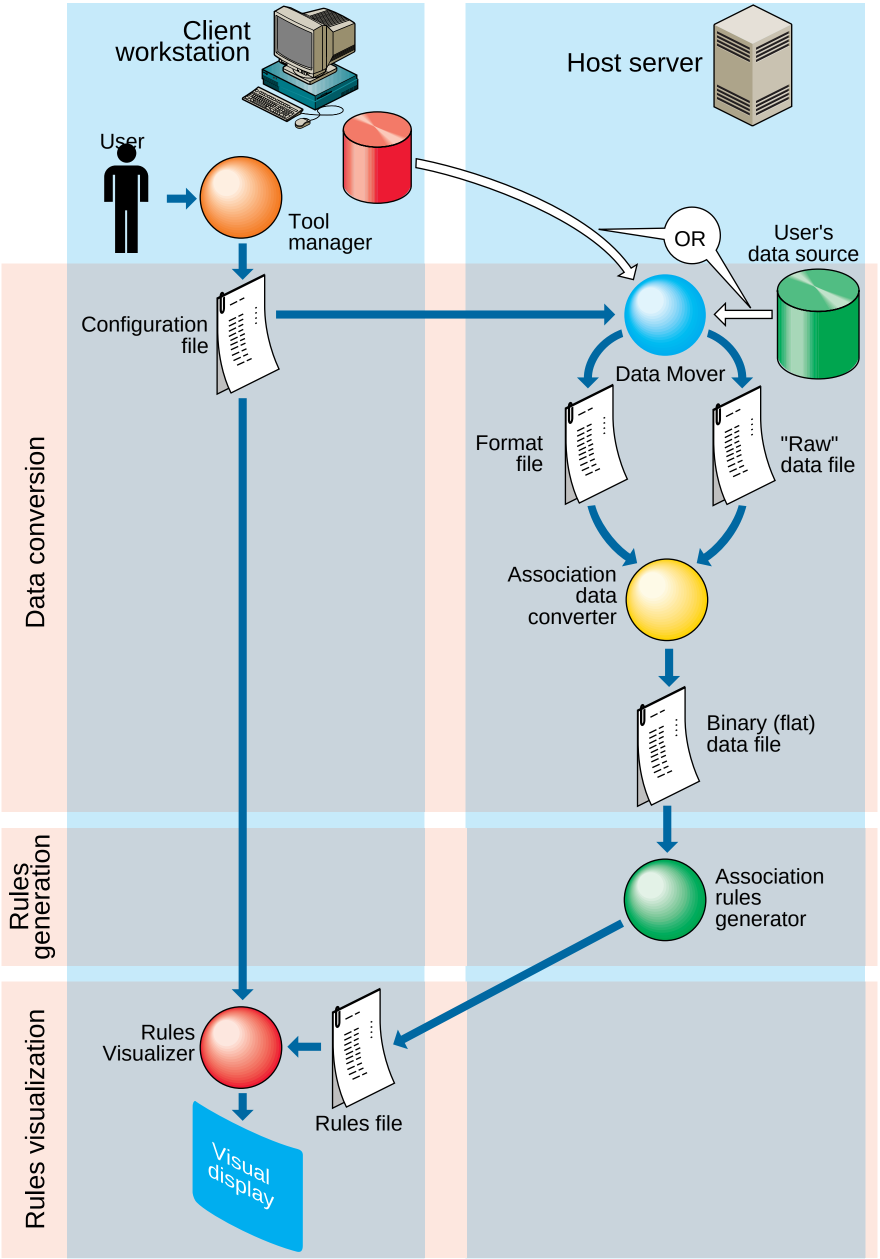


Figure 7-1 Execution Sequence of the Rules Visualizer

Data Conversion

The association data converter takes a “raw” data file, such as one resulting from a database query, and creates a binary data file in the format used by the association rules generator. The internal format of this generated file allows optimum processing by the rules generator.

Association Rules Generator

One example of applying the association rules generator is to obtain “market basket” data for customer buying patterns. Here, “market basket” is the set of items bought by each customer on a single visit to a store. An example rule in this context might be: “80% of the people that buy diapers buy baby powder.” This percentage is known as the *predictability* of the rule.

In the example, “diapers” is the item on the left-hand side (LHS) of the rule, and “baby powder” is the item on the right-hand side (RHS) of the rule.

Some applications of these rules are as follows:

- If “Fizzy Pop” appears on the RHS, the LHS can help us determine what the store should do to boost sales of this beverage.
- If “Bagels” appears on the LHS, the RHS can help us determine what products might be affected if the store no longer sells bagels.

The association rules generator part of this tool processes an input file, then generates an output file consisting of the rules. If X and Y are items in a record, then a rule such as

$X \Rightarrow Y$

indicates that whenever X occurs in a record, expect Y to occur with some frequency.

Components of a Generated Association Rule

The strength of the association is quantified by three numbers. The first number, the *predictability* of the rule, quantifies how often *X* and *Y* occur together as a fraction of the number of records in which *X* occurs. For example, if the predictability is 50%, *X* and *Y* occur together in 50% of the records in which *X* occurs. Thus, knowing that *X* occurs in a record, expect that 50% of the time *Y* occurs in that record.

The second number, the *prevalence* of the rule, quantifies how often *X* and *Y* occur together in the file as a fraction of the total number of records. For example, if the prevalence is 1%, *X* and *Y* occur together in 1% of the total number of records. The lower the prevalence, the more rules are generated, and the slower the performance of the tool might be.

Rules that meet a *minimum prevalence threshold* are important for two reasons:

1. A rule might have business value only if a reasonably significant fraction of records support the rule. For example, if everyone who buys caviar also buys vodka, the rule Caviar ⇒Vodka has 100% predictability. However, if only a handful of people buy caviar, the rule might be of limited value to the retailer.
2. A rule might not be statistically significant if a very small number of records support the rule. The rule might be due to chance, and it would not be prudent to make decisions based on such a rule.

You can specify a minimum prevalence threshold for the generated rules. The default minimum prevalence threshold is 1%. You can also specify a minimum predictability threshold for the generated rules. The minimum predictability threshold default is 50%.

The third number is *expected predictability*. The expected predictability is the frequency of occurrence of the RHS items. So the difference between expected predictability and predictability is a measure of the change in predictive power due to the presence of the LHS rule. Expected predictability gives an indication of what the predictability would be if there were no relationship between the items.

The Association Rules generator does not report rules in which the predictability is less than the expected predictability. In other words, a rule such as A->B is not reported if the frequency of A and B occurring together is less than the frequency of B alone.

Note: Given just Y and a rule of the form X ⇒ Y, nothing is known about X. Rules specify implications only from the LHS to the RHS.

Table 7-1 summarizes the three numbers that quantify the strength of each association rule.

Table 7-1 Association Rules Components

Measure	Description
Prevalence	Frequency of LHS and RHS occurring together.
Predictability	Fraction of RHS out of all items with LHS, or the prevalence divided by the frequency of occurrence of LHS items.
Expected Predictability	Frequency of occurrence of RHS items.

Hierarchical Data

The rules generator also works on hierarchical data, which includes a component that relates (or maps) data to new data at varying degrees of generality. The ability to handle hierarchical data allows rules to be generated at the desired level of generality.

For example, consider the hierarchy shown in Table 7-2. This hierarchical information, in addition to the "market basket" data that lists the products purchased in each record, allows rules to be generated at four levels. In contrast to rules learned at the lowest level, which relate specific products to each other, a rule at the highest level might be "Milk implies Bread."

Table 7-2 Example of Hierarchical Levels

Level	Example
Product Group	Milk
Category	Non-Refrigerated Milk
Brand	Lucerne
Product ID (UPC / SKU Code)	1 pint can of Premium Condensed Milk

Rules Visualization

The rules visualization part lets you graphically display and explore the generated association rules. The rules are presented on a grid landscape, with left-hand side (LHS) items on one axis, and right-hand side (RHS) items on the other. As shown in Figure 7-2, attributes of a rule are displayed at the junction of its LHS and RHS item. The display can include bars, disks, and labels.

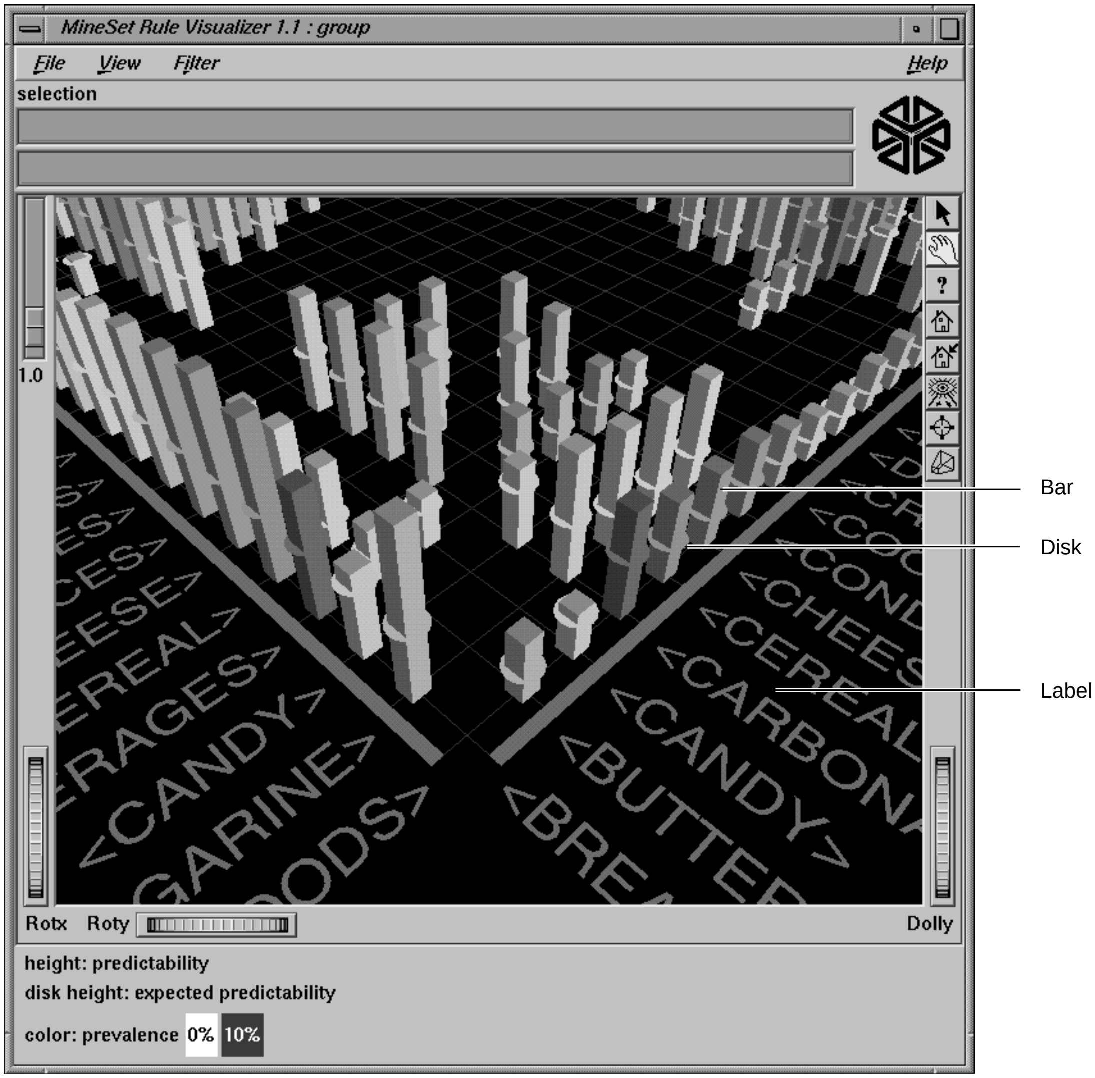


Figure 7-2 Detail View of the Rules Visualizer's Main Window

If the displayed view is too small, item labels do not appear on the side of the axes. You can zoom in on the view until the item labels appear (see the Dolly description in the “Thumbwheels” section).

A legend indicating the mapping between displayed attributes (such as bar heights and colors) and the values associated with the underlying rules (such as predictability and prevalence) can be displayed at the bottom of the main window.

File Requirements

Each of the Rules Visualizer’s three components has its own file requirements. These are detailed in the following subsections.

Files Required by the Association Data Converter Part

- A “raw” data file that results from extracting raw data from a source (such as a relational database). This file is processed by the association data converter to produce the internal binary data file used by the association rules generator.
- A format file that specifies the format of the data file. If the internal binary data file (see next subsection) is created via the Tool Manager, this format file is created automatically. If the internal binary data file is created via the command line, this format file must be created manually (see Appendix D, “Creating Data and Configuration Files for the Rules Visualizer”).

Files Required by the Association Rules Generator Part

- An internal binary data file, which results from running the association data converter on your original data.

  If you have hierarchical data, the association rules generator also requires the following two files:
- A mapping file, which specifies the mapping between hierarchical levels.
- A description file, which specifies a string description for each item at a specific hierarchical level.

Files Required by the Rules Visualization Part

- A rules file that results from running the association rules generator.
- A **.**ruleviz configuration file that specifies parameters used by the rules visualizer program (such as mapping colors to prevalence values) when displaying the generated rules. This file is easily created using the Tool Manager (see Chapter 3). You also can use an editor (such as jot, vi, or Emacs) to produce this file (see Appendix D, “Creating Data and Configuration Files for the Rules Visualizer”).

  These configuration files must have a *.ruleviz* extension.

Starting the Rules Visualizer

The Rules Visualizer has three components. The following subsections describe the procedure for starting each one.

Starting the Association Data Converter Part

There are two ways to start the association data converter part of the Rules Visualizer:

- Use the Tool Manager to configure and start the data converter. (See Chapter 3 first for details on most of the Tool Manager’s functionality, which is common to all MineSet tools; see below for details about using the Tool Manager in conjunction with the data converter.)
- Enter the following command at the UNIX shell command-line prompt:

  ```
  assoccvt parameters
  ```

  The *parameters* are described in Appendix D, “Creating Data and Configuration Files for the Rules Visualizer.”

Starting the Association Rules Generator Part

There are two ways to start the association rules generator part of the Rules Visualizer:

- Use the Tool Manager to configure and start the association rules generator. (See Chapter 3 first for details on most of the Tool Manager's functionality, which is common to all MineSet tools; see below for details about using the Tool Manager in conjunction with the association rules generator.)
- If the data with which you are working is non-hierarchical, enter this command at the UNIX shell command line prompt:

  ```
  assocgen parameters
  ```

  If your data is hierarchical, enter this command at the UNIX shell command-line prompt:

  ```
  mapassocgen parameters
  ```

  The *parameters* for both instances are described in Appendix D, "Creating Data and Configuration Files for the Rules Visualizer."

Starting the Rules Visualization Part

There are five ways to start the rules visualization part of this tool:

- Use the Tool Manager to configure and start the Rules Visualizer. (See Chapter 3 first for details on most of the Tool Manager's functionality, which is common to all MineSet tools; see below for details about using the Tool Manager in conjunction with the Rules Visualizer.)

- Double-click the Rules Visualizer icon, which is in the MineSet page of the icon catalog. The icon is labeled *ruleviz*. Since no configuration file is specified, the start-up screen requires you to select one by using File > Open.

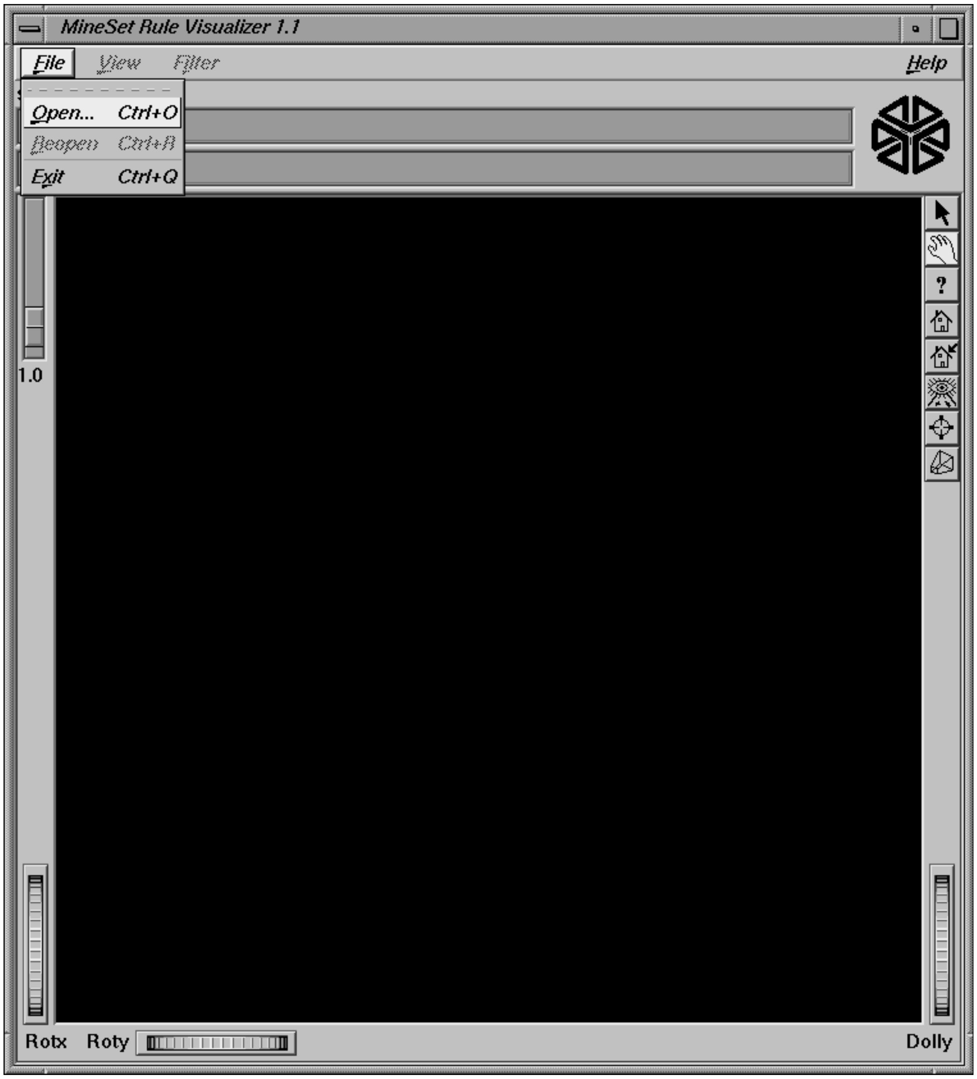


Figure 7-3 Rules Visualizer Start-Up Screen With File Menu Pulled Down, and Open Selected

- If you know what configuration file you want to use, double-click the icon for that configuration file. This starts the Rules Visualizer and automatically loads the configuration file you specified. This only works if the configuration filename ends in *.ruleviz* (which is always the case for configuration files created for the Rules Visualizer via the Tool Manager).
- Drag the configuration file icon onto the Rules Visualizer icon. This starts the Rules Visualizer and automatically loads the configuration file you specified. This works even if the configuration filename does not end in *.ruleviz*.
- Enter this command at the UNIX shell command-line prompt:

  `ruleviz [` *configFilename* `]`

When starting the rules visualization part of this tool, you must specify the configuration file, not the data or rules file.

Configuring the Rules Visualizer Using the Tool Manager

This section describes how the components of the Rules Visualizer can be configured using the Tool Manager. Although the Tool Manager greatly simplifies the task of configuring the Rules Visualizer, you can construct a configuration file for this tool using an editor (see Appendix D, “Creating Data and Configuration Files for the Rules Visualizer”).

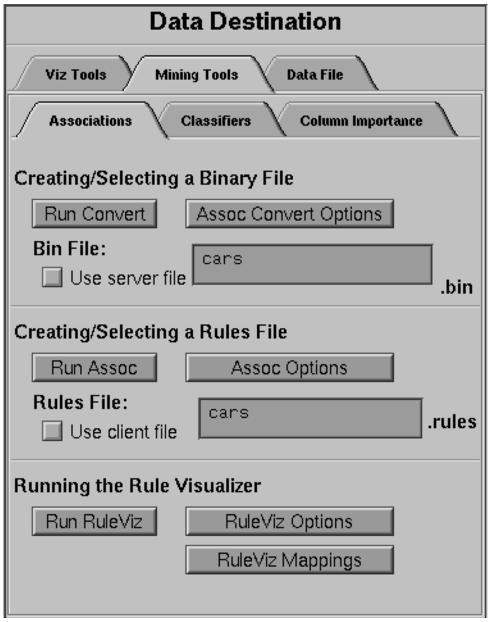
Note that the steps required to connect to a data source are described in Chapter 3.

The sections below follow the configuration and invocation of the Rules Visualizer components in the conventional order:

- creating a file for the association rules generator
- generating rules
- displaying rules

Setting Up Associations

To set up associations, use the example database table *cars*. Assume that you want to find out if there is an association between miles per gallon, horsepower, and the year the car was built. For example, did mileage improve over time? Did engines become less powerful? The following steps (and Figure 7-4) show you how to set up the associations and map table columns to the data you want to study.



Bin File: cars .bin
□ Use server file

Rules File: cars .rules
□ Use client file

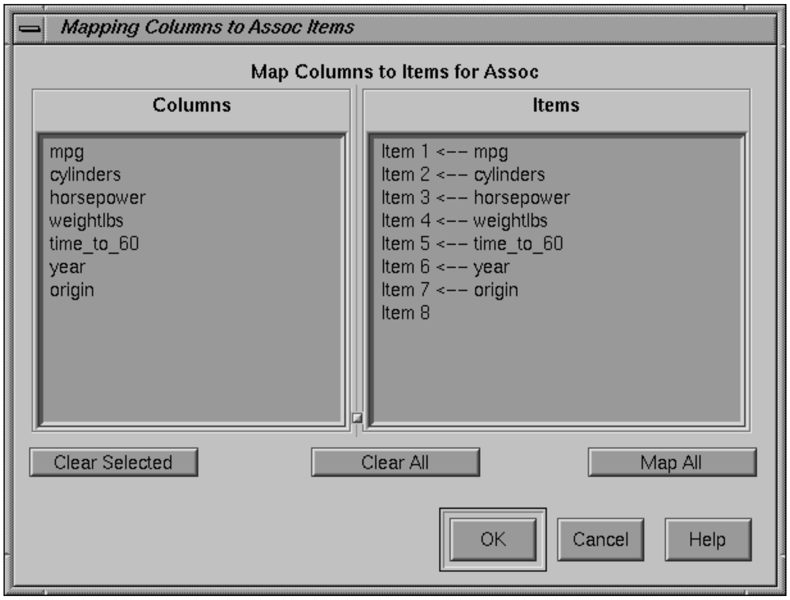
Figure 7-4 Initial Tool Manager Window for Association Generation

1. Specify a server name in the startup window; for this example, **`aztec`**.
2. Select Database from the Data Source tab. Connect to the chosen DBMS by logging in with your username and password.
3. Use either the Single Table or the SQL Query option to extract the information you want. For this example, choose the *cars* table from the correct database.
4. (Optional step) In the Data Transformations tab you can choose the transformations you want do on the data before you give it to the associations engine. One recommended transformation is to create bins for numeric data. (The binning operation and the options available for it are described in detail in Chapter 3.) This leads to more "meaningful" rules from the association engine. For example, instead of using discrete values for the weightlbs attribute in the "cars" table such as 3504, 3693, 3436, 3433, and so on, it may be more meaningful to give weightlbs_bin value ranges such as 1600-2500, 2501-3500, and so on.

   For this example, click on the *Bin Columns* ... button, and select all the columns in the Bin Column window for binning.

   Note: If you run associations without binning any of the numerical columns (**int**s, **float**s, **double**s) you get the warning message `Running associations on unbinned non-categorical data. Binning is recommended for producing more useful results.`
5. Choose the Mining Tools tab from the Data Destination tab.
6. Choose the Associations tab from the Mining Tools tab.

7. At this point you can either have your data file converted to the associations internal binary format by clicking on the *Assoc Convert Options* button, or use a previously-converted binary file by selecting the Use server file checkbox. This example assumes you are creating a new binary file from your data file.


Mapping Columns to Assoc Items
Map Columns to Items for Assoc
Clear Selected
Clear All
Map All
OK
Cancel
Help

Columns

mpg
cylinders
horsepower
weightlbs
time_to_60
year
origin

Items

Item 1 <-- mpg
Item 2 <-- cylinders
Item 3 <-- horsepower
Item 4 <-- weightlbs
Item 5 <-- time_to_60
Item 6 <-- year
Item 7 <-- origin
Item 8

Figure 7-5 Association Convert Options Dialog Box

The database in the Current Columns text panel can contain multiple table columns. By mapping specific columns to association rules, the association rules generator can find the association between any possible pair of those items.

8. The Map Columns to Items for Assoc window shows two panels:
   - Columns shows the columns in the data
   - Items shows a single item: Item 1

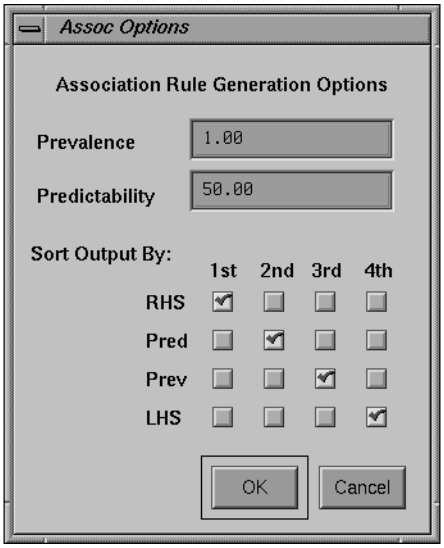
   The *Map All* button on this window can be used to map all the attributes in the data source to items for the associations engine. The *Clear All* and *Clear Selected* buttons can be used to clear / change the mapping between a column and an item.

   The default behavior is to map all columns to items. Therefore, if you omit this step or if you open this window, you find all columns mapped. For this example, click *OK*.
9. Click the *Run Convert* button in the *Associations* tab to convert the data into an internal binary file. The name of the binary file created appears in the window (you can type in another name if you do not like the default provided by the Tool Manager).

For the cars data, you are setting up a binary file that lets you explore corollaries between different attributes of cars. The Tool Manager causes the information to be extracted from the database and converted to a binary format. As this procedure is executed, the message `Waiting for server to create binary files` appears. When this procedure is finished, the message `Binary file created` appears.

Applying Association Rule Options

After creating the binary file (or choosing a previously created one), you can run the Association Rules generator. You can choose options for this by clicking on the *Assoc Options* button. This causes the dialog box in Figure 7-6 to be displayed.



Assoc Options

Association Rule Generation Options

Prevalence: 1.00

Predictability: 50.00

Sort Output By:

	1st	2nd	3rd	4th
RHS	☑	□	□	□
Pred	□	☑	□	□
Prev	□	□	☑	□
LHS	□	□	□	☑

Figure 7-6 Association Rule Options Dialog Box

Prevalence—Lets you specify the minimum prevalence threshold as a percentage of the total number of records. The default is 1%. The possible values are 0–100.

Predictability—Lets you specify the minimum predictability threshold for rules. Rules with a predictability below this value are not generated. The default is 50%. The possible values are 0–100.

Sort Output By—Lets you specify how you want the output sorted, by one of the following:

- the right hand side (RHS) of the rules
- the left hand side (LHS) of the rules
- predictability
- prevalence

Click the buttons to specify which comes first, second, third, and so forth.

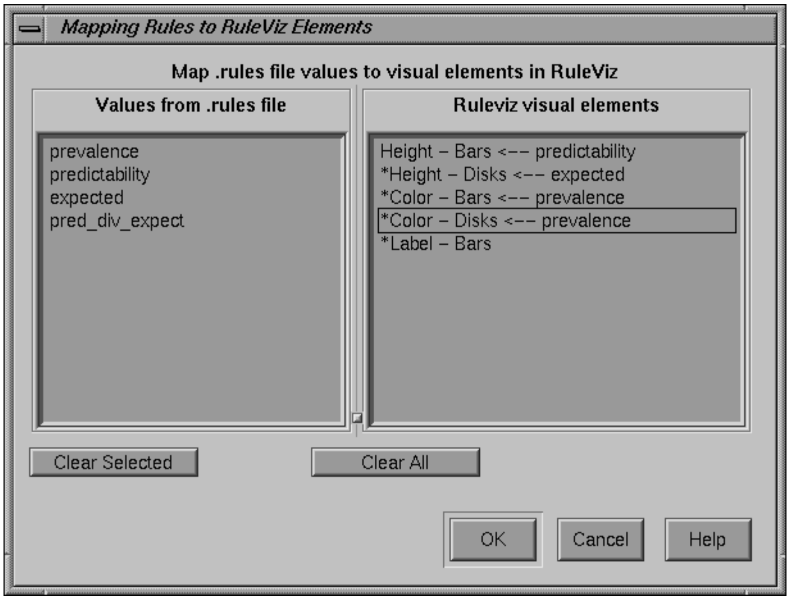
Once you have made your association rule options selections, click the *OK* button. This returns you to the Tool Manager startup screen.

To start generating rules based on the data you have chosen and the options you have configured, click the *Run Assoc* button on the Associations tab.

While the rules are being generated, the message `Waiting for server to create rules file` is displayed. When the process is finished, the rules file is downloaded to your local disk, and the message `Rules file received from server` is displayed. You are now ready to visualize these rules.

Mapping Columns to Visual Elements

The Rules Visualizer lets you map attributes of the rules to visual elements of the display. Clicking on the *RuleViz Mappings* button brings up the *Ruleviz Mappings* panel shown in Figure 7-7.


Mapping Rules to RuleViz Elements
Map .rules file values to visual elements in RuleViz
Values from .rules file
Ruleviz visual elements
Clear Selected
Clear All
OK
Cancel
Help

prevalence
predictability
expected
pred_div_expect

Height – Bars <–– predictability
*Height – Disks <–– expected
*Color – Bars <–– prevalence
*Color – Disks <–– prevalence
*Label – Bars

Figure 7-7 The Rules Visualizer's Mappings Panel

The visual elements that can be mapped are listed below:

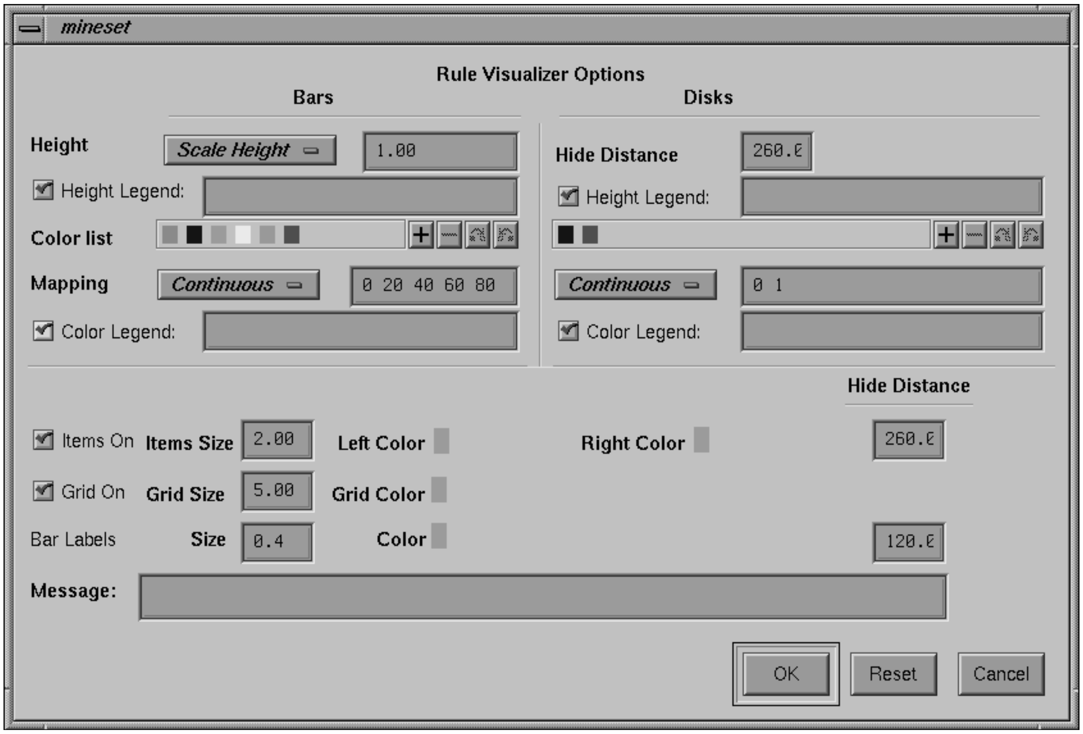
- *Height* - *Bars*—Lets you specify what the bar heights represent.
- *Height* - *Disks*—Lets you specify what the disk heights represent.
- *Color* - *Bars*—Lets you specify what the bar colors represent.
- *Color* - *Disks*—Lets you specify what the disk colors represent.
- *Label* - *Bars*—Lets you specify what the bar labels represent.

The default mappings are as follows:

- *predictability* to bar heights
- *expected predictability* to disk heights
- *prevalence* to bar and disk colors

Specifying Ruleviz Options

Clicking on the *Ruleviz Options* button causes a new dialog box to be displayed (Figure 7-8). This lets you change some of the Rules Visualizer options from their default values.



Rule Visualizer Options

Bars

Height Scale Height 1.00

☑ Height Legend:

Color list ▨ ■ ▨ □ ▨ ■

Mapping Continuous 0 20 40 60 80

☑ Color Legend:

Disks

Hide Distance 260.0

☑ Height Legend:

■ ■

Continuous 0 1

☑ Color Legend:

Hide Distance

☑ Items On Items Size 2.00 Left Color ▨ Right Color ▨ 260.0

☑ Grid On Grid Size 5.00 Grid Color ▨

Bar Labels Size 0.4 Color ▨ 120.0

Message:

Figure 7-8 Rule Visualizer Options Dialog Box

This dialog box has two panels: the top one lets you set options for bars and disks and the bottom one lets you specify options for items, the grid, and labels.

Items in the top panel are listed below:

- Height button—Lets you specify whether the bars and disk heights are to be normalized so that the tallest bar equals the height field value (*Max Height*), or whether they are to be scaled by the height field value (*Scale Height*).
- Height field—Lets you enter the maximum or scale value for bar and disk heights.
- Hide Distance—Lets you specify the distance at which disks are not graphically represented. Smaller numbers in this field specify a shorter distance; this means fewer disks are shown and performance is greater. Larger numbers indicate a greater distance; this means disks are always visible.
- Legends—Lets you enter a text string that appears as mapping information displayed at the bottom of the main Rules Visualizer window. This is information about mapping between display entities and data values (for example, bar height corresponds to predictability values).
- Color list—Lets you add or edit a color. To add a color to the list, click the + button. To edit a color, click the color. See “Color Options for the MineSet Visualizers” in Chapter 3 for a more detailed explanation of how to choose and change colors.
- Mapping—Lets you specify whether the color change that is shown in the graphic display is *Continuous* or *Discrete*. If you choose Continuous, the color values (of the bars or disks) shift gradually between the colors entered in the Color list field as a function of the values that are mapped to those colors in the Color list field.

Example 1:

If you

- used the Color Browser to apply red and green (for bars and / or disks)
- selected *Discreet* for the *Mapping*
- entered the values `0 100`

then the display shows all bars and / or disks with values of less than 50 in red, and all those with values greater than or equal to 50 in green.

Example 2:

If you

- used the Color Browser to apply red and green (for bars and / or disks)
- selected *Continuous* for the *Mapping*
- entered the values `0 100`

then the display shows all bars and / or disks with values less than or equal to 0 as completely red, those as greater than or equal to 100 as completely green, and those between 0 and 100 as shadings from red to green.

If no mapping and values are specified, a continuous mapping is used, and values are generated automatically from the minimum value to the maximum value in the data.

Items in the bottom panel are as follows:

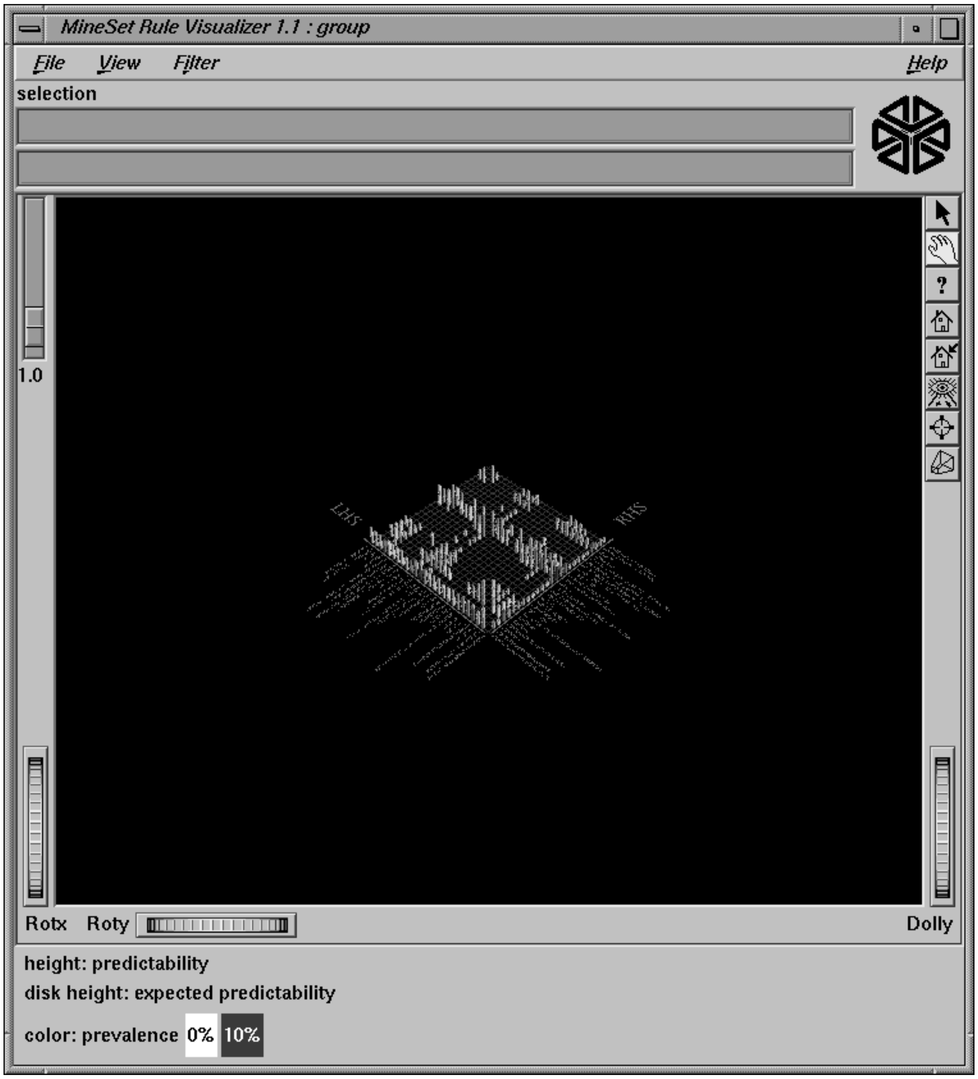
- Items On and Grids On checkbox buttons—Let you determine whether items (the names on the side of the grid) are displayed or hidden.
- Size (for Items, Grid, and Bar Labels)—Lets you specify the size for items, the grid, and bar labels. If you mapped a column value to bar labels in the Requirements panel of the Tool Manager startup screen, you can specify a size for those labels.
- Color (for Left-Hand Items, Right-Hand Items, Grid, and Bar Labels)—Lets you specify the color for LHS and RHS items, the grid, and bar labels. If you mapped a column value to bar labels in the Requirements panel of the Tool Manager startup screen, you can specify a size for those labels.
- Hide Distance—Lets you specify the distance at which the LHS items, RHS items, grid, or labels become invisible. Smaller distances might improve performance, but the objects disappear more quickly. The higher the number, the greater the distance at which labels are hidden.
- Message—Lets you specify the message displayed when the pointer is moved over an object or when an object is selected.

Invoking the Rules Visualizer

To see the Rules Visualizer graphically represent your data, click the *Run RuleViz* button at the bottom of the Associations tab in the Data Destination panel of the main Tool Manager window.

Working in the Rules Visualizer's Main Window

The Rules Visualizer part of this tool graphically displays the data in a rules file using the specifications of a valid configuration file. For example, specifying *group.ruleviz* results in the image shown in Figure 7-9.



height: predictability

disk height: expected predictability

color: prevalence 0% 10%

Figure 7-9 Initial Rules Visualizer View When Specifying group.ruleviz

The rules are presented on a grid, initially displayed with left-hand side (LHS) items displayed on the left side of the window and right-hand side (RHS) items on the right. A rule is displayed at the junction of its LHS and RHS items. The display can include bars, disks, and labels.

When the scene is close enough, the LHS and RHS axes are labeled with the item names, unless this has been turned off in the configuration file. (To view the grid and rules at closer range, use the Dolly thumbwheel, described in the "Thumbwheels" section.)

You can change the labels as well as what the heights and colors of the bars and disks represent by modifying the configuration file via the Tool Manager (see Chapter 3) or using an editor to change the configuration file.

For example, in Figure 7-9, bar heights correspond to predictability values, bar colors correspond to prevalence values, and disk heights correspond to expected predictability.

Viewing Modes

The two modes of viewing are *grasp* and *select*. To toggle between these modes, press the Esc key. You also can change from one mode to the other by clicking the appropriate button: to enter select mode, left-click the arrow button (to the top right of the main window); to enter *grasp* mode, left-click the hand button (immediately below the arrow button, near the top right of the main window).

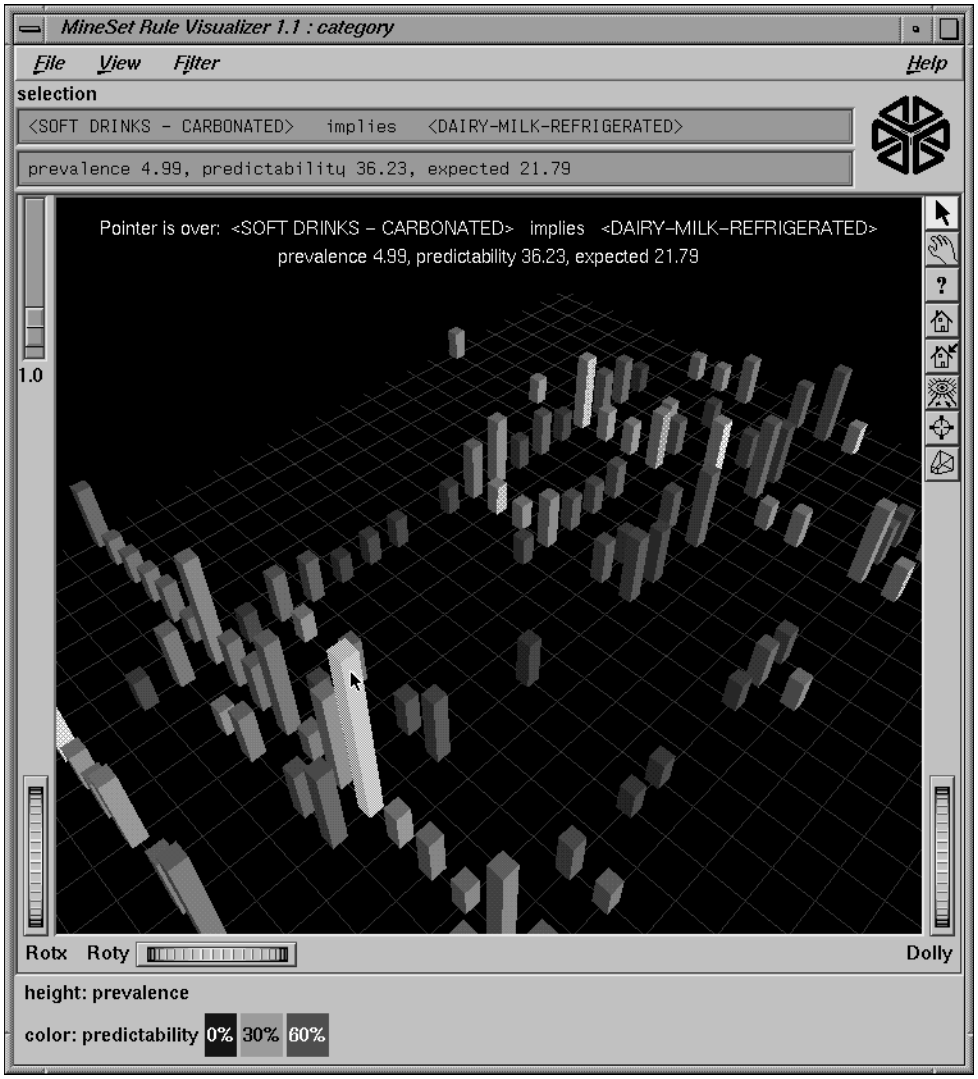
Grasp Mode

In grasp mode the cursor appears as a hand. This mode supports panning, rotating, and scaling the scene’s size in the main window.

- To rotate the display, press the left mouse button and move the mouse in the direction you want to rotate. (Also see the rotating controls *Rotx* and *Roty* described in “Thumbwheels” on page 219.)
- To pan the display, press the middle mouse button and drag it in the direction you want the display panned.
- To move the viewpoint forward, press the left and middle mouse buttons simultaneously and move the mouse downwards. To move the viewpoint backward, press the left and middle mouse buttons simultaneously and move the mouse upwards. This is equivalent to the functions provided by the Dolly thumbwheel.

Select Mode

In select mode, you can obtain additional information about a rule by placing the cursor over a bar. This highlights the selected bar and causes information about the rule represented by that bar to appear at the top of the main window.



```
<SOFT DRINKS - CARBONATED>   implies   <DAIRY-MILK-REFRIGERATED>
prevalence 4.99, predictability 36.23, expected 21.79
```

Pointer is over: <SOFT DRINKS – CARBONATED> implies <DAIRY–MILK–REFRIGERATED>
prevalence 4.99, predictability 36.23, expected 21.79

Figure 7-10 Cursor Over a Rules Visualizer Object

The information is displayed as long as the cursor remains over the object. If you position the pointer cursor over an object and click the left mouse button, that same information appears in the Selection Window, which is above the main window, under the “Selection” label.

This Selection information remains visible until another object is selected, or until no object is selected (if you click the black background). Using the mouse, you can cut and paste this text into other applications, such as reports or databases.)

External Controls

Several external controls surround the graphics window. These consist of buttons, thumbwheels, and sliders.

Buttons

At the top right of the image area are eight buttons, shown in Figure 7-11.

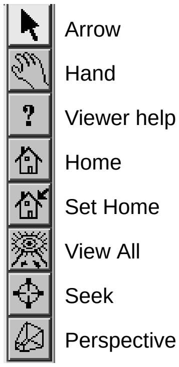


Figure 7-11 Rules Visualizer External Buttons

- *Arrow* puts you in select mode. When in this mode, the cursor shape is an arrow. Select mode lets you highlight graphical objects in the main window.
- *Hand* puts you in grasp mode. When in this mode, the cursor shape is a hand. Grasp mode lets you rotate, zoom, and pan the display in the main window.
- *Viewer help* brings up a help window describing the viewer itself.

- *Home* takes you to a designated location. Initially, this location is the first viewpoint shown after invoking the Rules Visualizer and specifying a configuration file. If you have been working with the Rules Visualizer and have clicked the *Set Home* button, then clicking *Home* returns you to the viewpoint that was current when you last clicked *Set Home*.
- *Set Home* makes your current location the Home location. Clicking the *Home* button returns you to the last location where you clicked *Set Home*.
- *View All* lets you view the entire grid and all the rules, keeping the angle of view. To get an overhead view of the scene, rotate the camera so that you are looking directly down on the rules grid, then click the *View All* button. (To rotate the camera, see the description of the Rotx thumbwheel on page 220.)
- *Seek* takes you to the point or object you click after selecting this button.
- *Perspective* is a toggle button that lets you view the scene in 3D perspective (closer objects appear larger, farther objects appear smaller). Clicking this button toggles 3D perspective on (default setting) or off.

  Note: If perspective is off, the Dolly thumbwheel becomes the Zoom thumbwheel.

Thumbwheels

Three thumbwheels appear around the lower part of the graphics window border. They let you dynamically move the viewpoint.

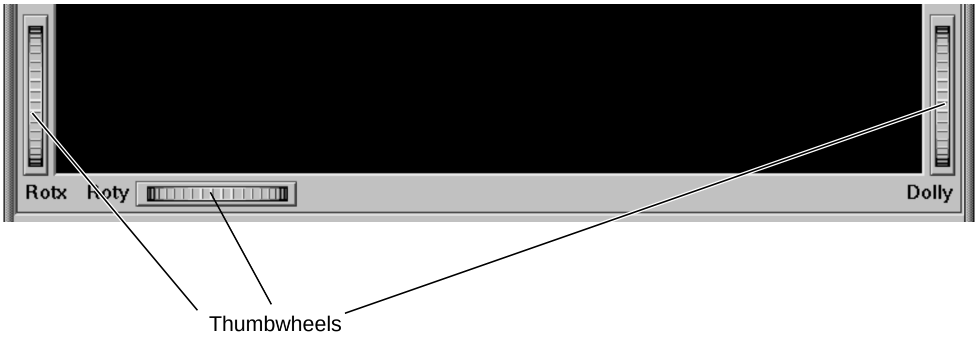


Figure 7-12 Rules Visualizer Thumbwheels

- The vertical thumbwheel Rotx (rotate about the x axis), on the left, rotates the display up and down.
- The horizontal thumbwheel Roty (rotate about the y axis), at the bottom left, rotates the scene in the main window around its centerpoint left and right.
- The vertical thumbwheel Dolly, on the right, moves the viewpoint forward and backward. Note that as you use the Dolly thumbwheel to magnify the scene in the main window, additional detail can appear. This is not the case with the Zoom slider, which merely enlarges the scene without adding detail.

  Note: If perspective is off, the Dolly thumbwheel becomes the Zoom thumbwheel.

The Height Slider

The *Height* slider, at the upper left corner of the main window, lets you scale the heights of objects (bars and disks) in the main window.

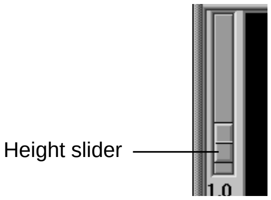


Figure 7-13 Rules Visualizer's Height Slider

Pulldown Menus

The Rules Visualizer has three pulldown menus, labeled File, Filter, and Help.

The File Menu

The File menu (Figure 7-14) contains three options.

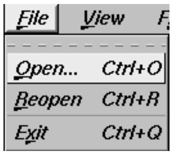


Figure 7-14 Rules Visualizer File Menu

- Open loads and opens a configuration file, displaying it in the main window. Previously displayed data is discarded.
- Reopen reloads the current configuration file. This is useful if either the configuration file or data file has changed.
- Exit closes the current window and exits the application.

The View Menu

The View menu (Figure 7-15)contains one option.

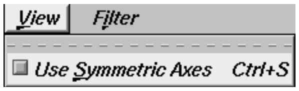


Figure 7-15 Rules Visualizer View Menu

Use Symmetric Axes controls how items are displayed along the left- and right-hand side axes. If enabled, every item appears on both axes, making the axes identical. Otherwise, only the required items appear on each axis.

The Filter Menu

The Filter menu brings up a Filter panel (Figure 7-16) that lets you reduce the number of rules displayed in the main viewing area, based on one or more criteria. You can use the filter panel to fine-tune the display, emphasize specific information, or simply shrink the amount of information displayed.

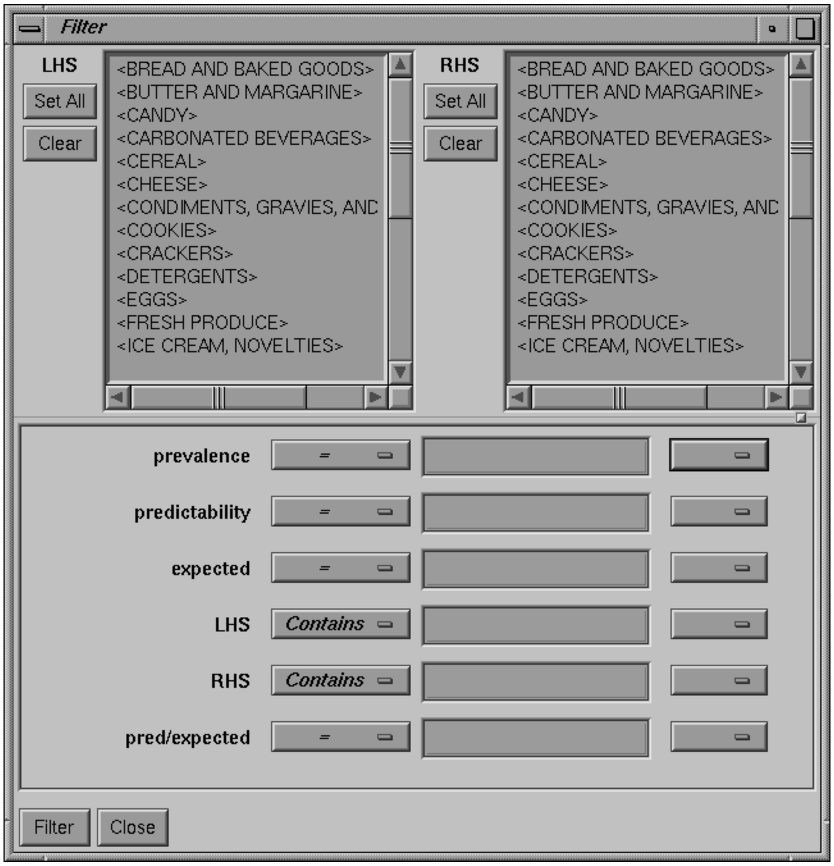


Figure 7-16 Rules Visualizer Filter Panel

The top pane lets you filter based on string variables, such as LHS and RHS. To select all values of a variable, click *Set All*. To clear the current selections, click *Clear*. To select a value, click it. To deselect a value, click it again.

The bottom pane lets you filter based on the values of both string and numeric variables.

To filter numeric values, enter the value, and select a relational operation (=, !=, >, <, >=, <=). To filter alphanumeric values, enter the string. You can use any of three types of string comparisons:

- Contains indicates that it contains the appropriate string. For example, California contains the strings Cal and forn.
- Equals requires the strings to match exactly.
- Matches allows wildcards:
  - An asterisk (*) represents any number of characters.
  - A question mark (?) represents one character.
  - Square braces ([ ]) enclose a list of characters to match.

  For example, California matches Cal*, Cal?fornia, and Cal[a-z]fornia.

In addition to numeric and string comparison operations, you can specify `Is Null.` Currently, this option does not match any rules, resulting in an empty display.

To the right of each field is an additional option menu that lets you specify “And” or “Or” options. For example, you could specify “sales > 20 And < 40.” You can have any number of And or Or clauses for a given variable, but cannot mix And and Or in a single variable.

Click the *Filter* button to start filtering. If you press *Enter* while the panel is active, filtering starts automatically.

Click the *Close* button to close the panel.

The Help Menu

The Help menu (see Figure 7-17) provides access to five help functions.

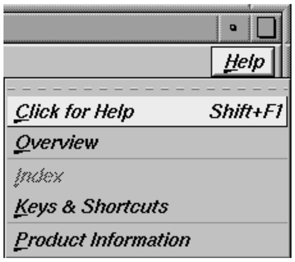


Figure 7-17 Rules Visualizer Help Menu

- Click for Help turns the cursor into a question mark. Placing this cursor over an object in the Rules Visualizer's main window and clicking the mouse causes a help screen to appear; this screen contains information about that object. Closing the help window restores the cursor to its arrow form and deselects the help function. The keyboard shortcut for this function is Shift+F1. (Note that it also is possible to place the arrow cursor over an object and press the F1 function key to access a help screen about that object.)
- Overview provides a brief summary of the major functions of this tool, including how to open a file and how to interact with the resulting view.
- Index provides an index of the complete help system. This option is currently disabled.
- Keys & Shortcuts provides the keyboard shortcuts for all of the Rules Visualizer's functions that have accelerator keys.
- Product Information brings up a screen with the version number and copyright notice for the Rules Visualizer.
- *MineSet User's Guide* invokes the IRIS Insight viewer with the online version of this manual.

Sample Files

The provided sample data, rules, and configuration files demonstrate the features and capabilities of the Rules Visualizers.

Sample Files for the Association Data Converter

There are two sample files provided for each of the two formats of the association data converter. These files are located in the */usr/lib/MineSet/assoccvt/examples* directory.

- *sing.dat* and *sing.fmt*
  The *sing.dat* file is a “raw” data file type, as described in the “Files Required by the Association Data Converter Part” on page 197. The *sing.fmt* file is the format file described in the same section. Both files are of the single-item-per-record format.
- *mult.dat* and *mult.fmt*
  The *mult.dat* file is a “raw” data file type, as described in the “Files Required by the Association Data Converter Part” on page 197. The *mult.fmt* file is the format file described in the same section. Both files are of the multiple-item-per-record format.

Sample Files for the Association Rules Generator

These files are located in the */usr/lib/MineSet/assocgen/examples* directory. Except for the *synthn.dsc* file, the sample files for the association rules generator are provided in 2-byte and 4-byte integer versions. The difference between the respective files is that the 4-byte integer version requires twice the amount of storage space of the 2-byte integer version.

- **synthn.dsc**
  This is a description file for items at the *n*th level of the hierarchy. For example, if n is 0, this file describes the lowest level; if n = 1, the file describes the next higher level of the hierarchy, and so forth. Description files are common to both 2-byte and 4-byte integer files.

Two-byte Integer Version

- *synths.dat*
  This is a data file with 2-byte integers. It corresponds to the data shown in Table D-9 on page 448.
- *synths.map*
  This is a 2-byte integer mapping file for hierarchical data.

Four-byte Integer Version

- *synthb.dat*
  This is a data file with 4-byte integers. It corresponds to the data shown in Table D-9 on page 448.
- *synthb.map*
  This is a 4-byte integer mapping file for hierarchical data.

Sample Files for the Rules Visualization Part

The following sample rules and configuration files are provided for use with the rules visualization part of this tool. These files correspond to the hierarchical datasets. Rules files contain the generated rules obtained by running the association rules generator part of the Rules Visualizer. Rules files must have a *.rules* extension. Each configuration file specifies how the corresponding rules file is displayed. Configuration files must have a *.ruleviz* extension. The files mentioned in this subsection are in the */usr/lib/MineSet/ruleviz/examples* directory.

- *group.rules* and *group.ruleviz*

  These files provide the generated rules and configuration specifications for product groups, such as bread and baked goods, dairy milk, and carbonated beverages.

- *category.rules* and *category.ruleviz*

  These files provide the generated rules and configuration specifications for product categories within product groups, such as refrigerated or non-refrigerated milk.

- *people94.rules* and *people94.ruleviz*

  These files provide the generated rules and configuration specifications for a census database, showing associations among marital status, education level, age, income, and other variables.

- *germanCredit.rules* and *germanCredit.ruleviz*

  These files provide the generated rules and configuration specifications for a credit database from Germany, showing associations among credit history, employment, savings, and other variables.

See */usr/lib/MineSet/ruleviz/examples/README* for additional information on the files in that directory.

Chapter 8

MineSet Inducers and Classifiers

This chapter provides a cursory introduction to classifiers and the algorithms that build them, called inducers. MineSet provides two inducer-classifier pairs:

- Decision Tree
- Evidence

The information in this chapter is equally applicable to either of these classifiers. Detailed descriptions of the MineSet inducers and classifiers are provided in Chapter 9, “Inducing and Visualizing the Decision Tree Classifier,” and Chapter 10, “Inducing and Visualizing the Evidence Classifier.”

Classifiers

A *classifier* predicts one attribute of a set of data given several other attributes. For example, if you have a database of iris flowers, a classifier can be built to predict the type of iris (*iris-setosa*, *iris-versicolor*, or *iris-virginica*) given the petal length, petal width, sepal length, and sepal width. The attribute being predicted (in this case, the type of iris) is called the *label*, and the attributes used for prediction are called the *descriptive attributes*.

MineSet can build a classifier automatically from a *training set*. This training set consists of records in the database for which the label has been determined, based on the descriptive attributes. For example, you supply a database table with one column for each descriptive attribute (such as petal length, petal width, sepal length, and sepal width) and one column for the label (*iris-setosa*, *iris-versicolor*, or *iris-virginica*). An algorithm that automatically builds a classifier from a training set is called an *inducer*.

When a classifier is generated, MineSet also generates a visualization that can help you understand how the classifier operates. This visualization can also provide valuable insight into the data itself.

Once a classifier is generated, it can be used to classify additional records not in the training set. These records need not contain the label attribute, since this value is predicted by the classifier.

Note: See Appendix I for a list of further readings about classifiers as well as acknowledgements for the datasets used in MineSet sample files.

Decision Tree Classifiers

Figure 8-1 shows the decision tree generated by the Decision Tree inducer for the example mentioned above.

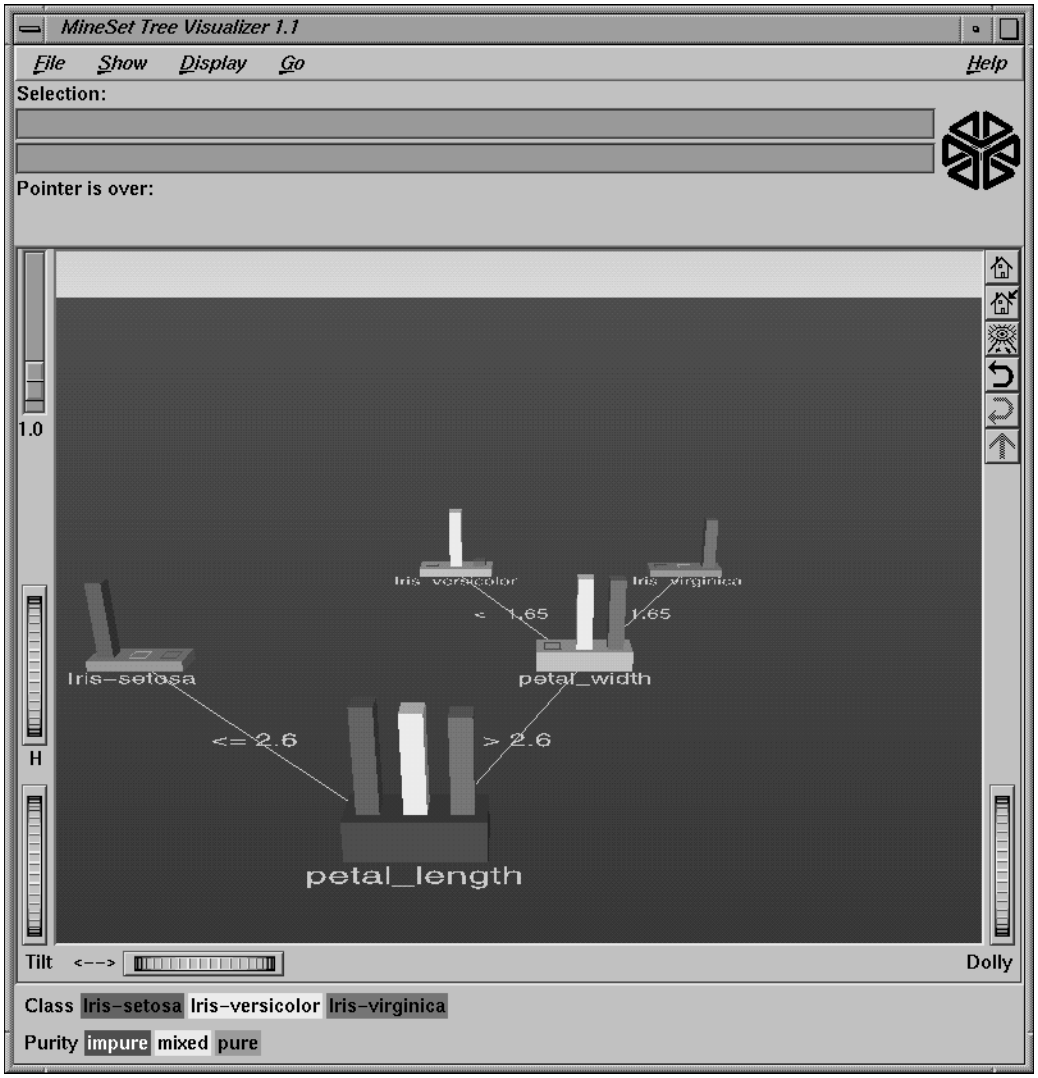


Figure 8-1 The Decision Tree Generated by the Decision Tree Inducer for Iris Database

To understand how the Decision Tree classifier assigns a label to each record, look at the attributes tested at the nodes and the values on the connecting lines. In the decision tree shown in Figure 8-1, the first test (at the root of the tree) is for petal length. There are two branches from this root. If the petal length is ≤ 2.6, the left branch is taken; otherwise, the right branch is taken. The process is repeated until a leaf (final node) is reached. The leaf is labeled with the predicted class. The leaf represents a rule that is the conjunction of all tests from the root to the leaf. For example, the leaf labeled `Iris-Virginica` matches the rule

petal_length >2.6 and petal_width >1.65 implies iris_type = iris-virginica

Evidence Classifiers

Figure 8-2 shows the evidence information generated by the evidence inducer.

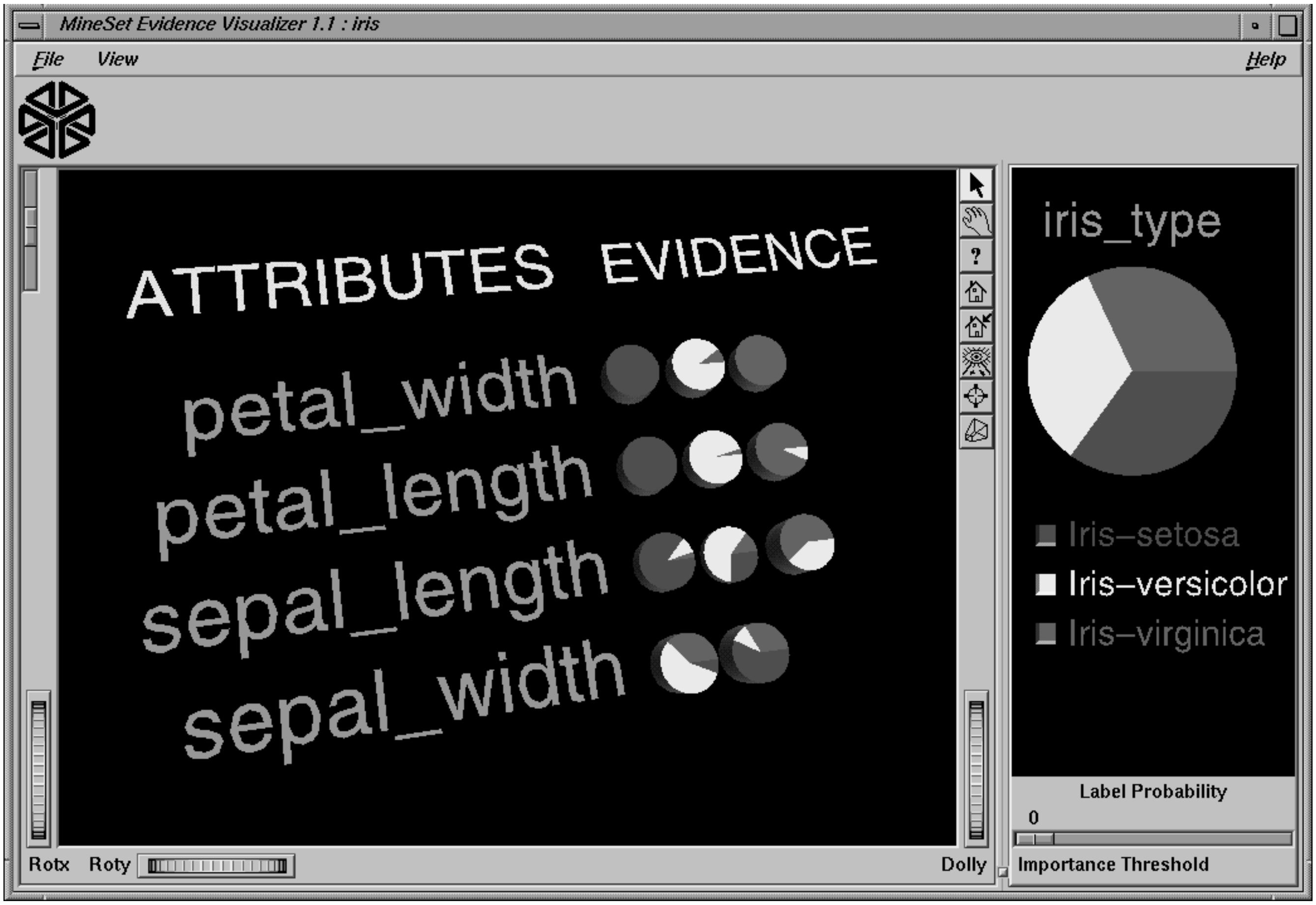


Figure 8-2 Results of Evidence Classifier for Iris Database

The right window of the screen shows the distribution of the classes in the training set. The left side shows rows of pie charts, one for each attribute. For every value of an attribute in the data, there is one pie chart matching it in the row for the attribute. Given a record with an attribute value corresponding to a pie chart, the pie chart represents how much evidence the classifier "adds" to each possible label value. For example, in Figure 8-2, a record with petal_width 1.2 (matching the second pie chart in the first row) adds much evidence for the *iris-versicolor* label value, little evidence for the *iris-virginica* label value, and no evidence for the *iris-setosa* label value. After evidence is accumulated from all the attributes (corresponding to one pie from every row), the label value with the most evidence is predicted.

Inducers

An inducer is an algorithm that builds a classifier from a *training set*, which consists of records with labels. The training set is used by the inducer to "learn" how to construct the classifier, as shown in Figure 8-3.

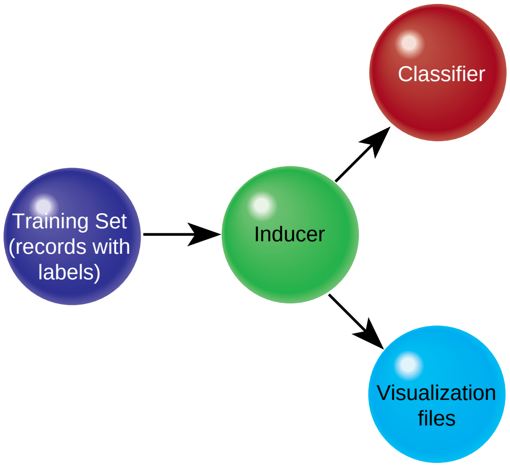


Figure 8-3 Method for Building a Classifier

Once the classifier is built, its structure can be visualized or used to classify unlabeled records, as shown in Figure 8-3 and Figure 8-4.

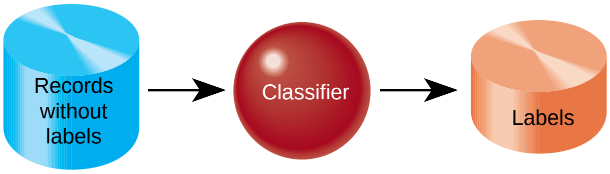


Figure 8-4 Using a Classifier to Label New Records

Running inducers can be a CPU- and I/O-intensive process. For this reason, the MineSet inducers run on the server, rather than on your workstation (see Figure 8-5).

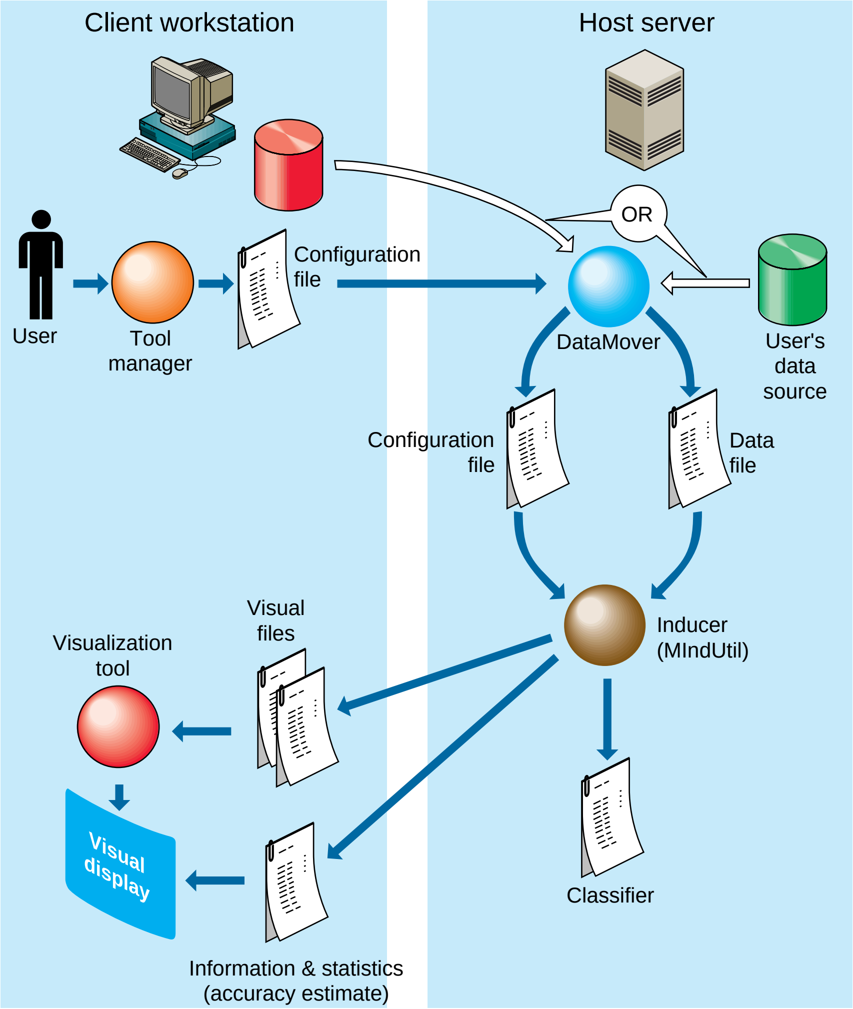
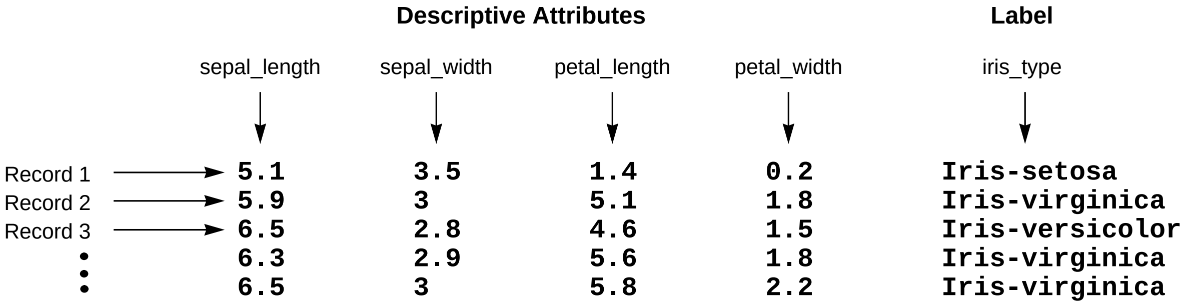


Figure 8-5 Tool Execution Sequence for Classifiers

Training Set

Inducers require a training set, which is a database table containing attributes, one of which is designated as the class label. The label attribute type must be discrete (binned values, character string values, or a few integers). The number of possible values for the label attribute should be small, preferably two or three values. The accuracy of the classifier is usually higher the fewer the number of label values. An example of this is the above-mentioned iris type attribute, which takes on one of three values (*iris-setosa*, *iris-versicolor,* or *iris-virginica*).

Figure 8-6 shows several records from a sample training set.



	Descriptive Attributes				**Label**
	sepal_length	sepal_width	petal_length	petal_width	iris_type
Record 1	**5.1**	**3.5**	**1.4**	**0.2**	**Iris-setosa**
Record 2	**5.9**	**3**	**5.1**	**1.8**	**Iris-virginica**
Record 3	**6.5**	**2.8**	**4.6**	**1.5**	**Iris-versicolor**
•	**6.3**	**2.9**	**5.6**	**1.8**	**Iris-virginica**
•	**6.5**	**3**	**5.8**	**2.2**	**Iris-virginica**

Figure 8-6 Sample Records From a Training Set

Once a classifier is built, it can classify new records as belonging to one of the classes (see Figure 8-4). These new records must be in a table that has the same attributes as the training set; however, the table need not contain the label attribute.

Applying a Classifier

After building a classifier, you can apply it to records to predict the label (see “The Apply Classifier Button” in Chapter 3). For example, if you built a classifier for predicting iris_type, you can apply the classifier to records containing only the descriptive attributes, and a new column is added with the predicted iris type.

In a marketing campaign, for example, a training set can be generated by running the campaign at one city and generating label values according to the responses in the city. A classifier can then be induced and campaign mail can then be sent only to people who are labelled by the classifier as likely to respond, thus saving mailing costs.

As an example of using mining tools for data quality, after building a classifier you can apply it to the training set in order to identify records that are mislabeled by the classifier (use *Classifier Only* mode to train on the whole dataset). Such records could warrant closer investigation. Perhaps they are “noise,” or they might give special insights. If, for example, you have a decision tree for the iris dataset, by applying the classifier, you get a new column (`iris_type_1`) containing the predicted labels. You can then add a column that is defined as type **int** with the expression (`iris_type != iris_type_1`). The new column has a 1 whenever the classifier misclassifies, and a zero when it correctly classifies. Another alternative is to define the new column as a float with the expression `(iris_type != iris_type_1) + 0.01`. The Scatter Visualizer can then be used with the original label mapped to color, and this new column mapped to size. Incorrect predictions are shown as big cubes; correct predictions are shown as small cubes.

Accuracy Estimation

When a classifier is built, it is useful to know how well you can expect it to perform in the future (how accurate its classification will be). Factors affecting classification accuracy include the following:

- The number of records available in the training set.

  Since the inducer must learn from the training set, the larger the training set, the more reliable the classifier should be; however, the larger the training set, the longer it takes the inducer to build a classifier. The improvement to the accuracy decreases as the size of the training set increases (this is a case of diminishing returns).

- The number of attributes.

  More attributes mean more combinations for the inducer to compute, making the problem more difficult for the inducer and requiring more time. Note that sometimes random correlations can lead the inducer astray; consequently, it might build less accurate classifiers (technically, this is known as "overfitting"). If an attribute is irrelevant to the task, remove it from the training set (this can be done using the Tool Manager).

- The information in the attributes.

  Sometimes there is not enough information in the attributes to correctly predict the label with high accuracy (for example, trying to determine someone's salary based on their eye color). Adding other attributes (such as profession, hours per week, and age) might increase the accuracy.

- The distribution of future unlabeled records.

  If future records come from a distribution different from that of the training set, the accuracy will probably be low. For example, if you build a classifier to predict something from a training set containing family cars, it may not be useful to classify records containing many sport cars because the distribution of attribute values can be very different.

The two common methods for estimating the accuracy of a classifier are described below. Both of these assume that future records will be sampled from the same distribution as the training set.

- Holdout: A portion of the records (commonly two-thirds) is used as the training set, while the rest is kept as a *test set*. The inducer is shown only two-thirds of the data and builds a classifier. The test set is then classified using the induced classifier, and the accuracy on this test set is the estimated accuracy. Figure 8-7 shows this accuracy estimation method.

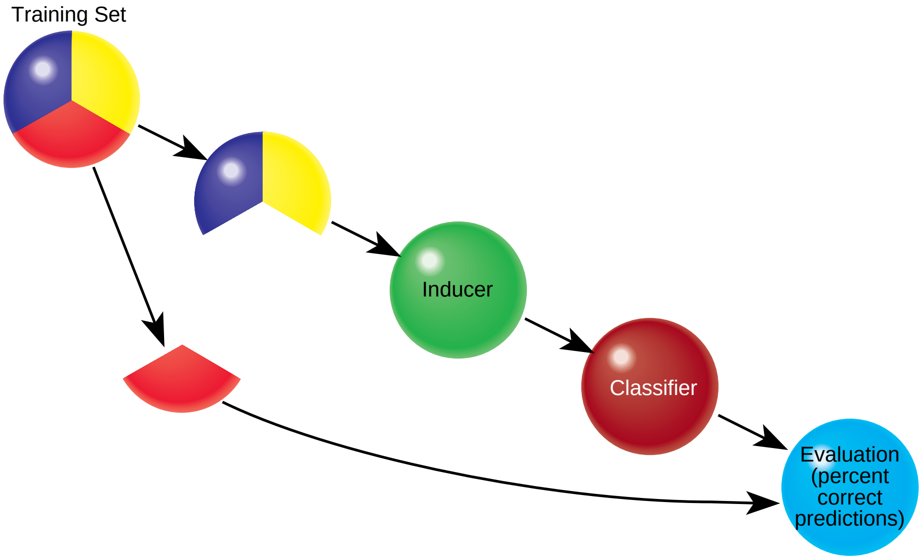


Figure 8-7 Estimating the Classifier’s Accuracy

This method is fast, but since it uses only two-thirds of the data for building the classifier, it does not make efficient use of the data for learning. If all the data were used, it is possible that a more accurate classifier could be built.

- Cross-validation: The data is split into *k* mutually exclusive subsets (*folds*) of approximately equal size. The inducer is trained and tested *k* times; each time, it is trained on all the data minus a different fold, then tested on that heldout fold. The estimated accuracy is then the average of the accuracies obtained. Figure 8-8 shows cross-validation with *k*=3 (note that the default value is *k*=10).

  Cross-validation can be repeated multiple times (*t*). For a *t* times *k*-fold cross-validation, *k***t* classifiers are built and evaluated. This means the time for cross-validation is *k***t* times longer. By default, *k*=10 and *t*=1, so cross-validation takes approximately 10 times longer than building a single classifier.

  Increasing the number of repetitions (*t)* increases the running time and improves the accuracy estimate and the corresponding confidence interval.

  You can increase or decrease *k*. Reducing it to 3 or 5 shortens the running time; however, estimates are likely to be biased pessimistically because of the smaller training set sizes. You can increase *k*, but this is recommended only for very small datasets.

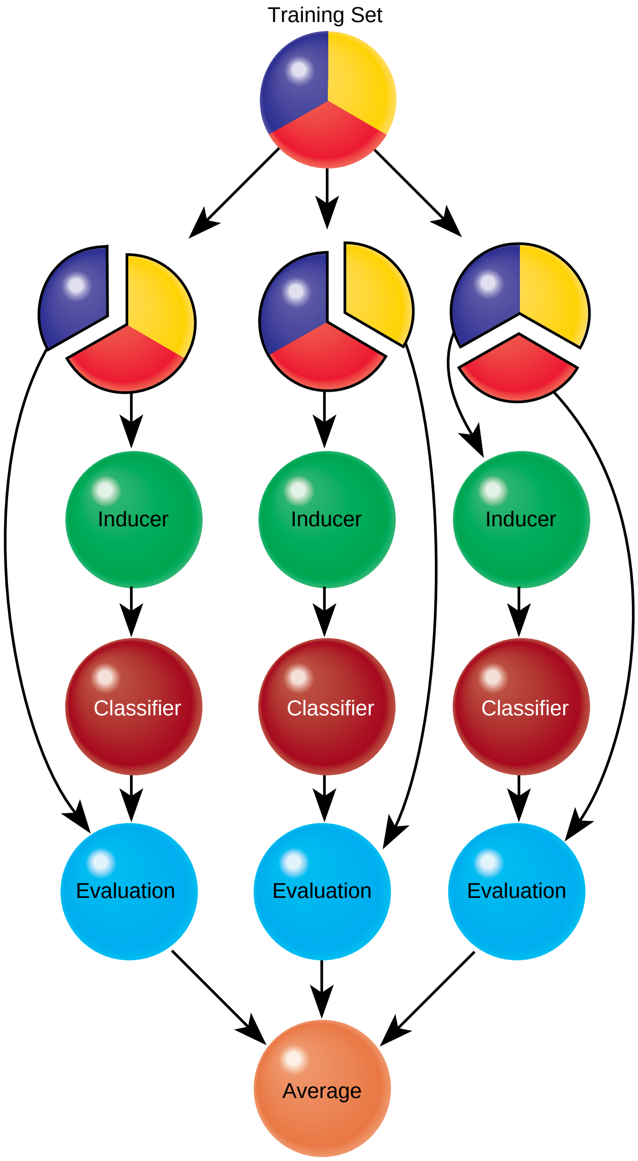


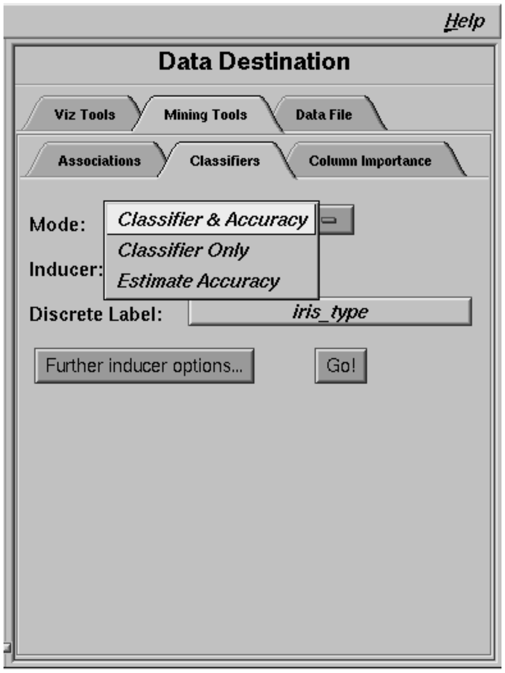
Figure 8-8 Classifier Cross-Validation (k=3)

Generally, a holdout estimate should be used at the exploratory stage, as well as on very large datasets. Cross-validation should be used for the final classifier building phase, as well as on small datasets.

Inducer Modes in Tool Manager

There are three modes for running an inducer (shown in Figure 8-9).

- Classifier and Accuracy
- Classifier Only
- Estimate Accuracy



Mode: Classifier & Accuracy

Discrete Label: iris_type

Figure 8-9 Options for Running the Inducer

The Classifier and Accuracy mode uses a holdout method to build a classifier: a random portion of the data is used for training (commonly two-thirds) and the rest for testing. This holdout proportion can be set in *Further Inducer Options* (see “Accuracy Estimation” on page 239). This method is the default mode and is recommended for initial explorations. It is fast and provides an accuracy estimate.

The Classifier Only mode uses all the data to build the classifier. There is no accuracy estimation. Use this mode when there is little data or when you build the final classifier.

The Estimate Accuracy mode does not build a classifier, but assesses the accuracy of a classifier that would be built if all the data were used (as with Classifier Only mode). Estimate Accuracy uses cross-validation, resulting in long running times. Cross-validation splits the data into *k* folds (commonly 10) and builds *k* classifiers. The process can be repeated multiple times to increase the reliability of the estimate. You can set the number *k* and the number of times in Further Inducer Options, as explained in “Accuracy Options for Inducers” below.

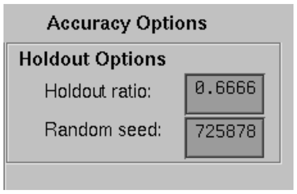
Use this method when there is little data (less than 1,000 records), or when a classifier is built from the full dataset using the Classifier Only option and you need a final accuracy estimate.

Accuracy Options for Inducers

The following options are available to fine tune the accuracy estimation for the inducers. The Accuracy Options available to you depend on the mode you have chosen.

In both Classifier & Accuracy and Estimate Accuracy, you can see a random seed that determines how the data is split into training and testing sets. Changing the random seed causes a different split of the data into training and test sets. If the accuracy estimate varies appreciably, the induction process is not stable.

In Classifier & Accuracy (see Figure 8-10), you can set the Holdout Ratio of records to keep as the training set. This defaults to 0.6666 (two-thirds). The rest of the records are used for assessing the accuracy.



Accuracy Options

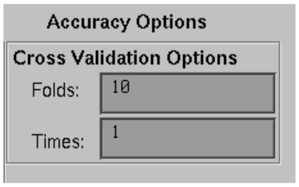
Holdout Options

Holdout ratio: 0.6666

Random seed: 725878

Figure 8-10 Accuracy Options With Holdout

In Estimate Accuracy (see Figure 8-11), you can set the number of folds in cross validation and the number of times to repeat the process.



Accuracy Options

Cross Validation Options

Folds: 10

Times: 1

Figure 8-11 Accuracy Options With Cross Validation

OK and Cancel Buttons

Once you have specified the Classification Options, click *OK* to have these options take effect and to return to the Data Destination panel. To return to the Data Destination panel without having changes to the options take effect, click *Cancel*.

Go! Button

After you have set the options, click the *Go!* button in the Data Destination panel to run the inducer.

The Information and Statistics Popup Window

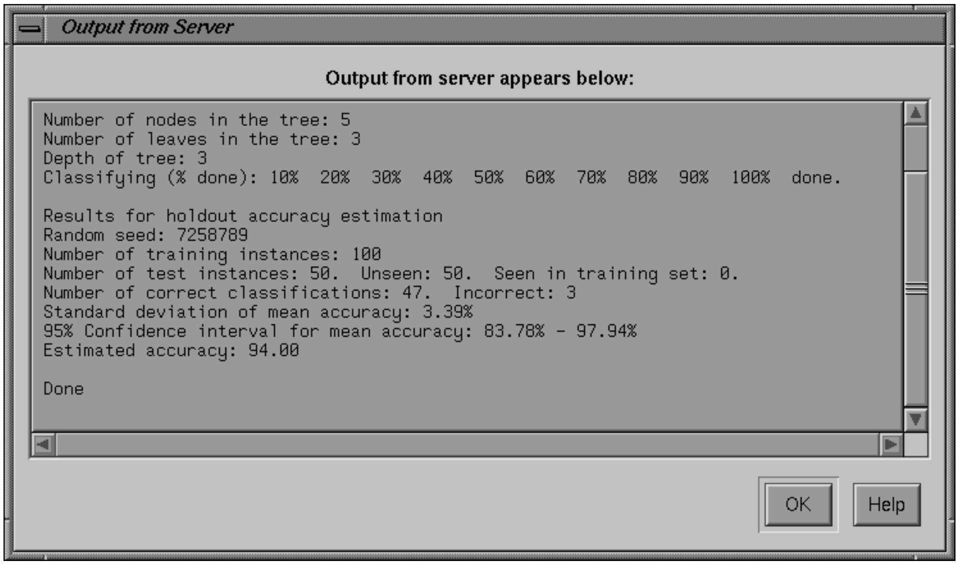
After you press *Go!* in the Data Destination panel, an information and statistics popup appears (unless you specified Classifier Only mode, in which case only the visualization appears). It displays specific information for the induced classifier. For example, for decision trees it shows the number of nodes, the number of leaves, and the depth of the decision tree (Figure 8-12). This information is saved automatically on your workstation under the session file name with a *-dt.out* or *-eviviz.out* extension, depending on whether a decision tree inducer or an evidence inducer was executed. If you specified the Classifier & Accuracy or the Classifier Only mode, the Tree Visualizer is invoked automatically.

The information above the results line are “dribble” information to show progress.

For Classifier & Accuracy, the first series of dots represent reading the file, then information about the classifier build progress is shown, then the test set classification progress is shown.

For Classifier Only mode, there is no test set classification phase.

For Estimate Accuracy, the times and folds are shown.



```
Number of nodes in the tree: 5
Number of leaves in the tree: 3
Depth of tree: 3
Classifying (% done): 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%  done.

Results for holdout accuracy estimation
Random seed: 7258789
Number of training instances: 100
Number of test instances: 50.  Unseen: 50.  Seen in training set: 0.
Number of correct classifications: 47.  Incorrect: 3
Standard deviation of mean accuracy: 3.39%
95% Confidence interval for mean accuracy: 83.78% - 97.94%
Estimated accuracy: 94.00

Done
```

Figure 8-12 The Information and Statistics Popup Window

When you have selected the Classifier & Accuracy mode, the Information and Statistics popup window contains the following information:

- The random seed used to split the data into training and test sets.
- The number of records used for training the inducer.
- The number of records used for evaluating the resulting classifier; of the test records, how many were seen during training, excluding the label attribute. It is possible to have duplicate records (“seen”) in a dataset; some records can be in both the training and test set. A large value of seen records indicates that there are many duplicate records. If their labels are contradictory, it may be impossible to achieve high accuracy without adding more attributes to the dataset.
- The number of correct and incorrect predictions made.
- The standard deviation of the estimated accuracy.

- A 95% confidence interval for the mean accuracy. This is the appropriate range of accuracy you can expect from the classifier if the data comes from the same distribution. Technically, this uses a more accurate formula than the two-standard deviation rule usually applied in statistics.
- The estimated accuracy.

When you have selected the Estimate Accuracy mode, the Information and Statistics popup window contains the following information:

- The number of cross-validation folds and times.
- The random seed.
- The estimated accuracy with standard deviation.
- The 95% confidence interval.

Special Options and Limitations

The following subsections describe how to set special options and the limitations of the inducers.

Setting Special Options

When the Tool Manager runs an inducer on the server (the MIndUtil program), it passes certain options to the inducers. Not all options are controlled through the Tool Manager GUI. Those options not controlled by Tool Manager take on their default values and can be overridden by setting them in a special file, called *.mineset-classopt*. Tool Manager prepends this file to the options sent. The file is optional. Tool Manager looks for it first in the current directory, then in your home directory. See Appendix E, “Command-Line Interface to MIndUtil: Classifiers, Discretization, Column Importance, and File Conversions” for more details about the options.

The file should contain one line per option, in the following format:

```
<OPTION>=<value>
```

For example, the special option LOGLEVEL increases the amount of information shown during the induction process. The default of zero shows very little information. Level 1 shows other options and slightly more information. Level 2 and higher show large amounts of information about the induction process. These levels are appropriate only if you have a firm understanding of the induction process. (See Appendix I, “Further Reading and Acknowledgments.”)

Default Limits and How to Override Them

Two limits and their respective options are as follows:

- Discrete attributes are ignored if they have more than 50 values. Discrete attributes with many values are usually inappropriate for classification. For example, first names and street addresses are unlikely to form predictive patterns.

  To speed up the induction process, attributes with over 50 values are ignored.

  You can override this value by setting MAX_ATTR_VALS to a higher number. For example, your *.mineset-classopt* file could contain the line

  ```
  MAX_ATTR_VALS=100
  ```
- Discrete labels with over 25 values are not allowed by default. Automatically induced classifiers are rarely appropriate for predicting one of a large number of label values. You should limit the label to a few values (preferably two or three). You can override this default limit by setting the option MAX_LABEL_VALS to a higher value in your *.mineset-classopt* file.

Other Limitations

There are three further limitations:

- Floating point numbers are read into MIndUtil as floats (4 bytes) even if they are represented as doubles (8 bytes) in the database or ASCII file. This limits the precision and magnitude of the representations allowed.
- Attributes of type arrays are always ignored.
- Dates are considered strings. Unless there are few dates, such attributes are usually ignored because of the limit on discrete attributes. You should bin dates before running an inducer.

Chapter 9

Inducing and Visualizing the Decision Tree Classifier

This chapter discusses the features and capabilities of the Decision Tree Inducer (its associated visualizer, the Tree Visualizer, is described in Chapter 4). This chapter provides an overview of this tool and discusses the ways of using it to generate Decision Tree classifiers. It then explains the Tree Visualizer's functionality when working with the main window. Finally, it lists and describes the sample files provided for this tool.

Note: It is assumed that you have read Chapter 8, "MineSet Inducers and Classifiers," before proceeding with this chapter.

Overview

A decision tree classifier assigns each record to a class. The underlying structure used for classification is a decision tree, such as the one shown in Figure 9-1.

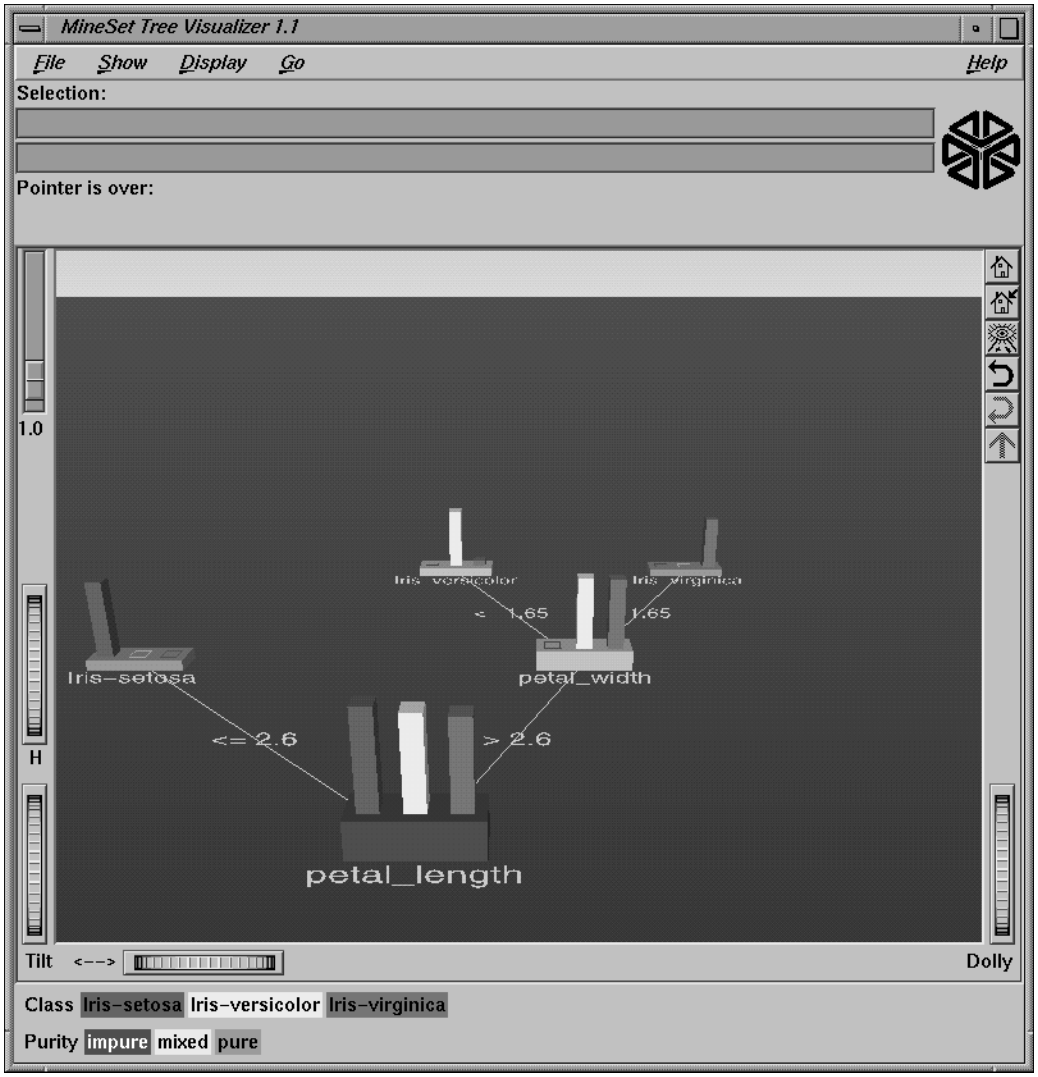


Figure 9-1 Decision Tree for the Iris Dataset

Inducing Decision Trees

A Decision Tree classifier is induced (generated) automatically from data. The data, which is made up of records and a label associated with each record, is called the *training set* (see Chapter 8, “MineSet Inducers and Classifiers”).

File Requirements

The Decision Tree Inducer requires a training set, as described in the “Training Set” section of Chapter 8. Files are generated by extracting data from a source (such as an ASCII file or a table in an Oracle, INFORMIX, or Sybase database). To apply the generated classifier, you should have a dataset of records with the same attributes as the training set, except that the label need not be present.

Running the Decision Tree Inducer

There are two ways to run the Decision Tree Inducer:

- From the Tool Manager.

  Connect to the server and select a data source (see “Connecting to a Server and Choosing a Data Source” in Chapter 3). In the Data Transformations panel, make the appropriate transformations; specifically, remove any column that you do not want considered.

  To see the induction process, choose Client File from Tool Manager and type `/usr/lib/MineSet/data/iris.schema`. You’ll see four continuous attributes and one discrete attribute in the Data Transformation panel. Since there is only one discrete attribute, the label option automatically shows it. Select the Decision Tree Inducer, and ensure you have selected the *Classifier & Accuracy* mode. To run the Inducer, click *Go!*.

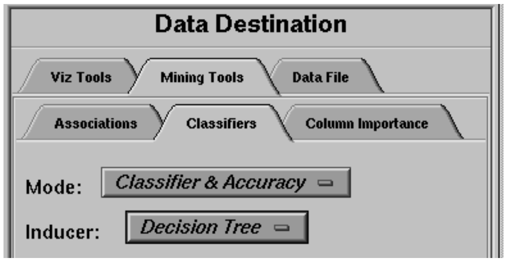
  You’ll see a small popup window with information and statistics, followed by another window showing the classifier structure in the Evidence Visualizer.

- From the command line.

  To induce a decision tree classifier from the command line, refer to Appendix E, “Command-Line Interface to MIndUtil: Classifiers, Discretization, Column Importance, and File Conversions.”

Configuring the Decision Tree Inducer Using the Tool Manager

To access the options for configuring the Decision Tree Inducer, select the Mining Tools tab on the Data Destination panel (Figure 9-2). From the subsequent tabs, select Classifiers. Ensure that the inducer you select is Decision Tree (the default). Your selections in the Mode and Algorithm menus determine the options available in the Further Inducer Options menu. After you have made your selections in these menus, click *Go!* to run the inducer, which, in turn, creates the classifier.



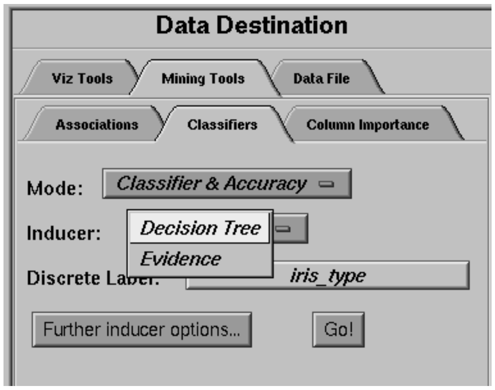
Mode: Classifier & Accuracy

Inducer: Decision Tree

Figure 9-2 Data Destination Panel in Tool Manager Showing Classifiers

Inducers

Figure 9-3 shows the two possible induction algorithms for creating the two MineSet classifiers: Decision Tree and Evidence.



Mode: Classifier & Accuracy

Inducer: Decision Tree

Discrete Label: iris_type

Figure 9-3 Data Destination Panel With Classifiers Selected and Inducer Algorithm Pulldown Menu

Discrete Labels

The Discrete Labels menu provides a list of possible discrete labels. Discrete attributes (binned values, character string values, or a few integers) have a limited number of values. You should select a label attribute with few values; for instance, two or three (see “Training Set” in Chapter 8). If there are no discrete attributes, the menu shows *No Discrete Label*, and the *Go!* button is disabled. You then must create a discrete attribute by binning or adding a new column using the Tool Manager’s Data Transformations panel.

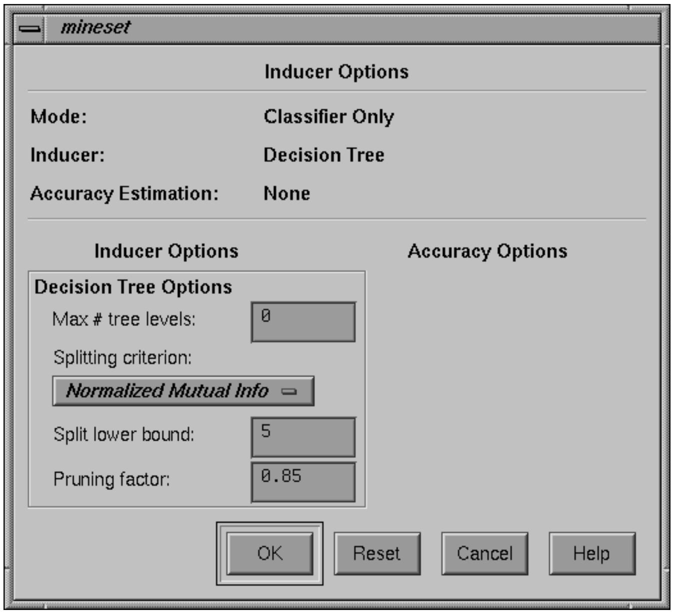
Classifier Name

The generated classifier is named with the prefix of the session filename (as determined in Tool Manager) with the suffix *-dt.class*. By default, all classifiers are stored on the server in the *file_cache* directory, which defaults to *mineset_files*. These classifiers can be used for future classification of unlabeled records; that is, they can be used to predict the labels for unlabeled datasets (see “The Apply Classifier Button” in Chapter 3).

Decision Tree Options

Selecting Further Classifier Options causes the Classifier Options dialog box to appear. This dialog box consists of three panels:

- The top panel indicates the choices you made in the Tool Manager's Data Destination panel.
- The bottom-left panel lets you specify further Inducer Options.
- The bottom-right panel lets you specify the Accuracy Options (unless the mode you chose in the Data Destination panel was Classifier Only, in which case this area is empty). The options shown in this panel depend on the type of Accuracy Estimation you chose (see "Accuracy Estimation" in Chapter 8).



mineset

Inducer Options

Mode: Classifier Only

Inducer: Decision Tree

Accuracy Estimation: None

Inducer Options

Accuracy Options

Decision Tree Options

Max # tree levels: 0

Splitting criterion: Normalized Mutual Info

Split lower bound: 5

Pruning factor: 0.85

Figure 9-4 Further Inducer Options

Decision Tree Inducer Options

To fine-tune the Decision Tree induction algorithm, you can change the following Decision Tree Inducer options (see Figure 9-4):

- Max # tree levels

  The default (0) indicates there is no limit to the number of levels in the decision tree. Limit the number of levels to speed up the induction or when you want to study the decision tree and not be distracted by too many nodes. Note that restricting the size decreases the run time but might degrade accuracy. Setting this option does not affect the attributes chosen at levels before the maximum level.

- Splitting criterion

  This option offers three splitting criteria selections. The definitions below are technical. It is difficult to know in advance which one of these criteria might be better. You should try them all and select the one that leads to the highest accuracy estimate or to a decision tree you find easiest to understand.

  Mutual Info—This is the change in purity (that is, the *entropy*) between the parent node and the weighted average of the purities of the child nodes. The weighted average is based on the number of records at each child node.

  Normalized Mutual Info (the default)—This is the Mutual Info divided by the log (base 2) of the number of child nodes.

  Gain Ratio—This is the Mutual Info divided by the *entropy* of the split while ignoring the label values.

  Normalized Mutual Info and Gain Ratio give preference to attributes with few values.

- Split lower bound

  This is a lower bound on the number of records that must be present in at least two of the node's children. The default for this option is 5. For example, if there is a three-way split in the node, at least two out of the three children must have at least five records. This provides another method of limiting the size of the decision tree.

  Raising this number creates smaller trees and speeds up the induction time. If you expect the data to contain noise (errors or anomalies), increase this number. If your dataset contains only a few hundred records, you can decrease this number to 2 or 3. If your dataset is very small (< 100 records), you might want to decrease this number to 1.

- Pruning factor

  A decision tree is built based on the limits imposed by Max # of Levels and Split Lower Bound. Statistical tests are then made to determine when some subtrees are not significantly better than a single leaf node in which case those subtrees are pruned.

  The default pruning factor of 0.85 indicates the recommended amount of pruning to be applied to the decision tree. Higher numbers indicate more pruning; lower numbers indicate less pruning. If your data might contain noise (errors or anomalies), increase this number to create smaller trees. The lowest possible value is 0 (no pruning); there is no upper value limit.

  Pruning is slower than limiting the maximum number of tree levels or increasing the split lower bound because a full tree is built and then pruned. Pruning, however, is done selectively, resulting in a more accurate classifier.

Working in the Tree Visualizer’s Main Window

The Tree Visualizer’s main window shows the decision tree. This decision tree consists of nodes connected by lines (see Figure 9-1).

Nodes

There are two types of nodes:

- decision
- leaf

Decision Nodes

Decision nodes specify the attribute that is tested at the node. Values (or ranges of values) against which the attributes are tested are shown at the lines. Each possible value for the attribute matches exactly one line. For example, the root of the decision tree in Figure 9-1 tests the attribute `petal_length`; the two lines emanating from the node specify the ranges of values for that attribute (≤`2.6` and `>2.6,`) so that every possible value matches either the right branch or the left branch. If the value is unknown and there is no line labeled with a question mark (?), the majority class of the current node is predicted.

Leaf Nodes

Leaf nodes in a decision tree specify a class. Follow the left branch in Figure 9-1 from the root to a leaf labeled `iris-setosa`. Note that the Decision Tree classifier classifies all records with petal_length ≤ 2.6 inches as belonging to the class *iris-setosa*.

Node Height and Color

The base of each node has a height and a color. The height corresponds to the number of records from the training set that have reached this node. In general, the more records, the more reliable the class distribution at every node.

Each node has a *measure of purity* (a number from 0 to 100) associated with it. This measure indicates how likely it is to accurately predict the correct class for the records at a given node. The color of the base indicates the purity of the node based on a street-light analogy: red indicates low purity (a prediction is difficult), yellow indicates mixed, green indicates high purity (making a prediction is easy and thus more likely to be correct).

Node bars

The vertical bars atop each node show the distribution of the classes at the node.

Lines

All possible outcomes are marked on the horizontal lines emanating from each decision node. Each line indicates the value (or range of values) against which the attribute of that node was tested.

Using the Main Window to Classify Records

To classify a record, start at the root, and test how to branch at every decision node. By following the appropriate lines based on the record's attribute values, you reach a leaf node. The label, or class, associated with the leaf node is the predicted classification of the record.

Some decisions are quickly made and take a shorter path (for example, `petal_length ≤ 2.6` implies `iris-setosa`). Other decisions can take a longer path (for example, the right branches, `petal_length > 2.6` and `petal_width > 1.65`). In general, every leaf corresponds to a rule that is the conjunction of all tests at the decision nodes and all the values (or ranges of values) on the lines leading to it from the root.

In the root of the tree shown in Figure 9-1, the purity is 0.06, indicating that it is extremely difficult to correctly predict the class. Indeed, the root has almost an equal number of records from each class.

The left child of the root has an purity of 100 because all records with petal_length ≤ 2.6 inches are of the *iris-setosa* class; thus, the prediction of *iris-setosa* is likely to be very accurate for all records with petal_length ≤ 2.6 inches. The right child of the root has a purity of 36.92. In this child, which matches records whose petal_length > 2.6 inches, there are no records belonging to the *iris-setosa* class; thus, the class is more likely to be *iris-versicolor* or *iris-virginica*. Because only two possibilities exist at this node, there is a higher purity than at the root.

The decision tree leaves segment the data into clusters sharing the same classification rule (path that leads to each leaf). By looking at the leaves, it is possible to see clusters that share the same set of properties.

External Controls

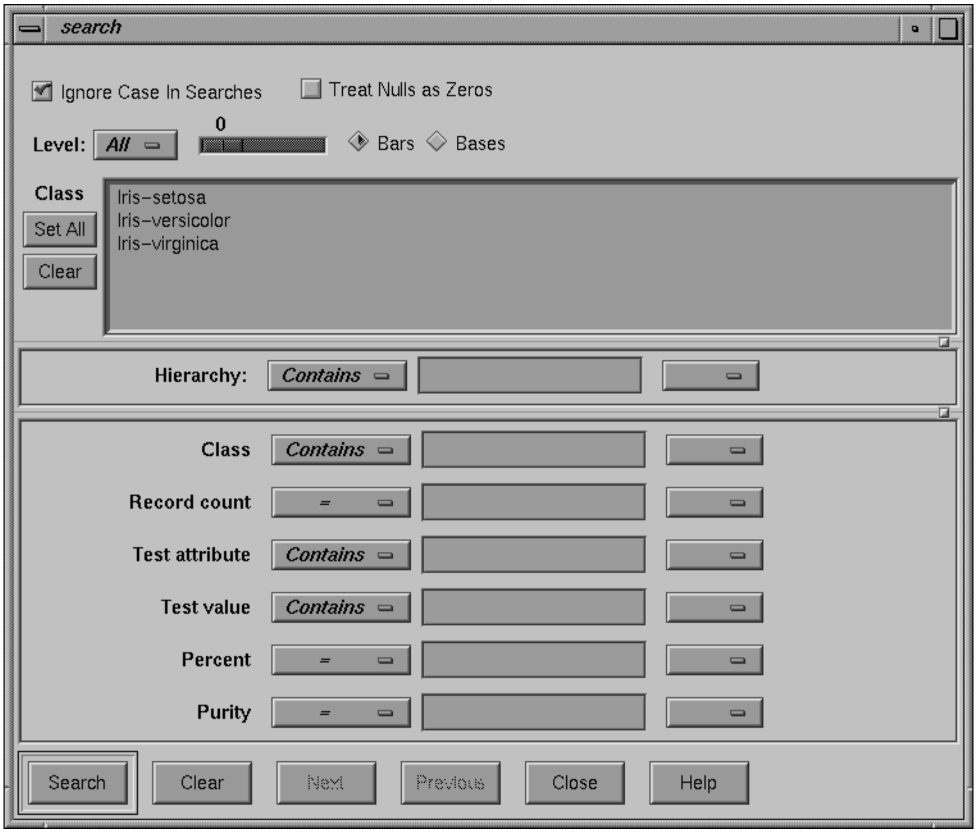
The external controls for the visualizer associated with the Decision Tree Classifier are the same as those for the Tree Visualizer. For a description of these controls, see “External Controls” in Chapter 4.

Pulldown Menus

The pulldown menus for the visualizer associated with the Decision Tree Classifier are the same as those for the Tree Visualizer. For a description of these menus, see “External Controls” in Chapter 4.

The Search Panel

Select Search Panel in the Show menu to bring up a dialog box that lets you specify criteria to search for objects (Figure 9-5). The search panel is the same search panel described in “The Search Panel” in Chapter 4; however, the item choices for decision trees are always the same. These are described below.



☑ Ignore Case In Searches □ Treat Nulls as Zeros

Level: All 0 ◆ Bars ◇ Bases

Class
Iris–setosa
Iris–versicolor
Iris–virginica

Hierarchy: Contains

Class Contains

Record count =

Test attribute Contains

Test value Contains

Percent =

Purity =

Figure 9-5 Tree Visualizer’s Search Dialog Box

The search can be restricted to specific class labels, either by selecting the values in the class list or by using the *Class* item, which allows more powerful comparison operators (such as Matches). Other items are described below:

- "Record count" lets you restrict the search to bars or bases (depending on the choice of the radio button bars / bases) with a given number of records. For example, you can restrict the search to bars containing over 50 records.
- "Test attribute" lets you restrict the search to nodes labeled by the given value that the node is testing. Note that decision node labels represent the test attribute, while leaf node labels show the predicted label. For example, if you select *Test attribute contains age*, only nodes that test the value of age are considered.
- "Test value" lets you restrict the search to nodes having an incoming line labeled with a value you specify.
- "Percent" lets you restrict the search to bars representing a percentage of the overall records at a node. For example, you might want to find all nodes such that a given class accounts for more than 80 percent of the records. To do this, click the class label, and select *Percent > 80*. Setting this item is meaningless if you select bases and not bars (the value for the bases is 0).
- "Purity" lets you restrict the search to nodes with a range of purity levels. For example, if you want to look at pure nodes (with one class predominant), you can select *Purity > 90*.
- "Level" lets you restrict the search to a specific level or range of levels. For example, you can search only the first five levels.

The following items and options are less useful for decision trees.

- "Hierarchy" finds all the nodes that match the given value at the tail of the path from the root. It then marks the children of these nodes.
- "Treat Nulls as Zeros" is not used by the Decision Tree Inducer because there are no null items generated for decision trees.

Once the search is complete, yellow spotlights highlight objects matching the search criteria. To display information about an object under a yellow spotlight, move the pointer over that spotlight; the information appears in the upper left corner, under the label "Pointer is over." To select and zoom to an object under a yellow spotlight, left-click the spotlight; if you press the Shift key while clicking, zooming does not occur.

Sample Files

The following examples show cases in which the Decision Tree Inducer can be useful. Each of these examples is associated with a sample data file provided with MineSet. By running the inducer, you can generate the *-dt.treeviz* files described below.

Note: The data files, which have a *.schema* extension, are located in */usr/lib/MineSet/data* on the client workstation. The classifier visualization files, which have a *-dt.treeviz* extension, reside on the client workstation in */usr/lib/MineSet/treeviz/examples*.

Origin of Cars

The *cars* dataset contains information about different models of cars from the 1970s and early 1980s. Attributes include weight, acceleration, and miles per gallon (mpg). The file */usr/lib/MineSet/treeviz/examples/cars-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/cars.schema* with the label set to origin (Japan, U.S., Europe) and the brand deleted to make the problem non-trivial.

If you have a dataset of car attributes, you might want to know what characterizes cars of different origins.

In the decision tree, you can see that cubic inches is an excellent discriminator for U.S.-made cars. Cars with large engines (>190.5 cubic inches) are all made in the U.S., but smaller cars are made everywhere. Note that in this tree, the root node (that is, the entire training dataset) has many more U.S. cars (67.8%), yet after a single split on the cubic inches, it is more difficult to predict the origin of cars with small engines. The purity of the root is 22.6 (orange color) showing that there is one class (U.S. in this case) that is dominant. The right node (cubic inches > 190.5) has purity 100 (green), indicating that we have identified a very pure subpopulation (all cars with large engines were made in the U.S.). The left node from the root has purity 0.33 (red). This subproblem is much harder than the original one: the number of records for each class is approximately the same. Branching left from this node, we can see that very small engines (≤85.5 cubic inches) are all Japanese.

Gender Attribution

The *adult* dataset contains information about working adults. This dataset was extracted from the U.S. Census Bureau. It contains data about people older than 16, with a gross income of more than $100 per year who work at least one hour a week. You might want to know how to characterize males and females. The file */usr/lib/MineSet/treeviz/examples/adult-sex-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/adult.schema*, with the label set to *sex*, after removing the relationship column (which would have made the classifier trivial). Note that this dataset contains almost 50,000 records; thus, running the Decision Tree Inducer can take several minutes.

The resulting visualization provides the following insights:

- The most important attribute is marital status.
- From the height of the base, it is obvious that most people are either divorced, married to a civilian spouse, or never married.
- The distribution at the root shows more males in this dataset. (Note that this database contains information about working adults and is not representative of the entire population.)
- The left-most node contains divorced working adults. We can see that the distribution is more balanced than at the root (60% female, 40% male). The second node contains married working adults. We can see that 89% are males. The third node contains working adults that have not married. As with the divorced group, they are approximately equal in number, with slightly more males. The right-most node contains working widowed adults. We can see that 81% are females, probably because of their higher life expectancy.

If you want to target working females for a new product, you can use the search panel to identify segments that have a large population of females. You can do this by choosing

- class matches female
- record count > 1000
- percent > 70

Two yellow spotlights show the matching nodes. Since they are both on one path, look at the node closest to the root. The path translates into the rule

```
marital_status = never_married and occupation =
     administrative_clerical implies that 71.6% are female
```

In this training set, out of 10,776 females at the root, 1129 satisfy this rule. This simple segment contains over 10% of working women.

Salary Factors

If you have a dataset of working adults, you might want to find out what factors affect salary. You might then divide the records into two classes: those making ≤ $50,000 a year, and those making more. Each record then has an attribute with two values: "– 50,000" and "50,000+". You can run a MineSet classifier to help determine what factors influence salary. The file */usr/lib/MineSet/treeviz/examples/adult-salary-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/adult.schema* with gross_income binned at the user-specified threshold of 50000 and the label set to gross_income_bin.

The resulting visualization provides the following insights:

- The root, which represents the entire training set, shows 70% of the working adults earning ≤ $50,000.
- Age is the most important factor. Only 3% of the people under 27 years old earn more than $50,000. Note that the base color is green, indicating a mostly pure segment.
- Education is an important factor for predicting salary for people over 27 years old. A high value leads to a node where 55% earn over $50,000.
- Of the segment that is older than 27 years and well educated, relationship is an important predictor of salary. For those persons that are married, chances of earning $50,000 or more increase to 73% for husbands and 76% for wives. (Note, however, that the node containing wives is small, representing few females.) If the person in this group is not married, chances of earning $50,000 or more decrease to 28% for males and 23% for females.

Iris Classification

In this dataset, each record describes four characteristics of iris flowers: petal width, petal length, sepal width, and sepal length. Each iris was further classified into the types *iris-setosa, iris-versicolor,* or *iris-virginica.* The goal is to understand what characterizes each iris type.

Before running a classifier, click the Column Importance tab in the Tool Manager's Classifiers tab; then click *Go!*. You obtain a ranking of the importance of the features: petal_width, petal_length, and sepal_length. You can map these to the axes in the Scatter Visualizer, with the iris_type mapped to the color, and see the clusters.

The file */usr/lib/MineSet/treeviz/examples/iris-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/iris.schema.*

Running the Tree Visualizer, you can see that the root is red (low purity), indicating that it's hard to make a prediction at the root. However, the left branch (petal-length ≤ 2.6 inches) goes to a green node (pure) containing only *iris-setosas.* The other branches are also quickly able to separate the classes using another test on the petal_width. The path petal-length > 2.6 and petal-width ≤ 1.65 ends with an impure leaf. There are 3 records of type *iris-virginica* and 33 of *iris-versicolor.* The decision tree did not split this node because it was deemed insignificant (by default, every split must contain five records).

To summarize: the flowers with petal length ≤ 2.6 inches are predicted as *iris-setosa*, those with petal length > 2.6 inches and petal width ≤1.65 inches are predicted as *iris-versicolor*, and those with a petal length >2.6 inches and a petal width > 1.65 inches are predicted as *iris-virginica.*

Note that because the decision tree makes binary splits on continuous attributes while Column Importance discretizes the data, the root split of the tree is different from the first attribute in column importance (see Chapter 11 for more details).

Mushroom Classification

The file */usr/lib/MineSet/treeviz/examples/mushroom-dt.treeviz* shows the Decision Tree classifier induced for the classification of mushrooms. This file was generated by running the inducer on */usr/lib/MineSet/data/mushroom.schema.*

The goal is to understand which mushrooms are edible and which are poisonous, given this dataset. There are over 8000 records in this set; thus, running this inducer might take several minutes. Note that under the default mode of the one-third holdout for accuracy estimation, a third of the records are kept for testing.

Each mushroom has many characteristics, including cap color, bruises, and odor. If you build a Decision Tree classifier, you can see that using only the odor attribute lets you determine in 50% of the cases whether the mushroom is poisonous or edible. If the mushroom has no odor, there is a 3% chance it is poisonous. The next attribute to look at is the shape of the stalk. If it tapers, the mushroom is edible; but if enlarges, there is a 10% chance the mushroom is poisonous. About 600 mushrooms reach this node. You can follow the tree down further nodes to see what other attributes to consider.

Party Affiliation

This dataset consists of voting records. The goal is to identify the party a congressperson belongs to given data about key votes. The dataset includes votes for each member of the U.S. House of Representatives on the 16 key votes identified by the *Congressional Quarterly Almanac* (*CQA*). The *CQA* lists nine types of votes: voted for, paired for, and announced for (these three are simplified to yes); voted against, paired against, and announced against (these three are simplified to no); voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three are simplified to an unknown disposition).

Before running a classifier, look at the 16 votes to see if you can perceive which features are important. Then run the Decision Tree classifier.

The file */usr/lib/MineSet/treeviz/examples/vote-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/vote.schema.*

Breast Cancer Diagnosis

The breast cancer dataset contains information about females undergoing breast cancer diagnosis. Each record is a patient with attributes such as cell size, clump thickness, and marginal adhesion. The final attribute is whether the diagnosis is malignant or benign. The file */usr/lib/MineSet/treeviz/examples/breast-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducers on */usr/lib/MineSet/data/breast.schema*.

The decision tree shows that *uniformity_of_cell_size* is a very strong discriminatory attribute. While the root is red (low purity), the two children of the root are shades of green (high purity), with the left node having over 97% of one class. While the accuracy is better with a fully built tree, a tree with a single test is already very accurate. As described in the "Decision Tree Inducer Options" section, you can limit the tree size to one level and estimate its accuracy.

Hypothyroid Diagnosis

The hypothyroid diseases dataset is similar to the one for breast cancer. The file */usr/lib/MineSet/treeviz/examples/hypothyroid-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/hypothyroid.schema*.

There are 3163 records in this dataset and most of them do not have hypothyroid (95.45%). While this means that one can predict "negative" and be correct with high probability, it's those people that have hypothyroid that we are most worried about. In technical terms, the false negatives are very important.

Looking at the decision tree, you can see that the root node is “green” (easy to predict). However, the single attribute on “fti” at the root shows that it is relatively easy to identify many of the negative diagnosis. People with high fti are 99.83% negative, and all those where the value is unknown are also negative (perhaps the doctor decided not to measure this attribute because something else was obvious), but the rest (145 people) are hard cases. We started with 2109 records (two-thirds of the whole dataset), but only 145 are really “interesting” to mine because it was very easy to throw away most cases. In this example most of the data is uninteresting and you want to concentrate on a small part quickly. Of the 145 people, you can see that about 65% are positive and 35% negative, which is why it’s hard to make a decision and the base of the node is orange-red.

As you move down the tree, increase the height scale (slider on the top left of the visualizer) to see the different heights. The node that catches most of the people with hypothyroid has the conditions “fti ≤ 64.5 and tsh > 5.96.” It contains 90 of the 96 records that have hypothyroid.

Pima Diabetes Diagnosis

This dataset is a diagnosis problem for diabetes using statistics gathered from a Native American tribe in Phoenix Arizona. The task is to determine whether a patient has diabetes, given some medical attributes, such as blood pressure, body mass, glucose level, and age.

The file */usr/lib/MineSet/treeviz/examples/pima-dt.treeviz* shows the Decision Tree classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/pima.schema.*

DNA Boundaries

There are 3,186 records in this DNA dataset. The domain is drawn from the field of molecular biology. Splice junctions are points on a DNA sequence at which "superfluous" DNA is removed during protein creation. The task is to recognize exon / intron boundaries, referred to as EI sites; intron / exon boundaries, referred to as IE sites; or neither. The IE borders are referred to as "acceptors" and the EI borders are "donors." The records were originally taken from GenBank 64.1 (*genbank.bio.net*). The attributes provide a window of 60 nucleotides. The classification is the middle point of the window, thus providing 30 nucleotides at each side of the junction.

In this example, the root of the decision tree shows the distribution of the three classes. By pointing to the bars, you can see that the composition is about 24% exon / intron, 24% intron / exon, and 52% none. The "left_01" in front of the root node indicates that this is an important attribute to look at first. The "left_01" notation refers to the first nucleotide found to the left of the splice junction in question. The choices of attribute values for this first nucleotide (and all nucleotides in general) are the "A", "G", "T", and "C" nucleotides. If the "left_01" nucleotide is a "G", then the "G" branch is taken and followed to the next node, where the distribution now shows that such a nucleotide is more likely to be an "exon / intron" or an "intron / exon" than at the root: the distribution is 35% for "exon / intron," 41% for "intron / exon", and 24% for "none." If the "left_01" nucleotide is an "A", "T", or "C", then the corresponding "A", "T", or "C" branch is taken instead and in all three cases, the probability of "none" increases dramatically (88%, 90%, and 95% respectively). This testing and branching process is repeated until the final node with the predicted class ("exon / intron", "intron / exon", or "none") is reached.

For this dataset, the Evidence Classifier (Chapter 10) is more appropriate than a Decision Tree due to the probabilistic nature of this domain. This can be verified by comparing the estimated accuracies.

Chapter 10

Inducing and Visualizing the Evidence Classifier

This chapter discusses the features and capabilities of the Evidence Classifier and Visualizer. It provides an overview of this classification tool as well as the inducer that generates it. It describes the ways of invoking this tool. It then explains the Evidence Visualizer's functionality when working with the

- Label Probability Pane
- Evidence Pane

Finally, it lists and describes the sample files provided for this tool.

Note: It is assumed that you have read Chapter 8, "MineSet Inducers and Classifiers," before proceeding with this chapter.

Overview

The Evidence Classifier assigns each record in a dataset to a class. The Evidence Visualizer displays the structure of an evidence classifier (Figure 10-1). The visualizer can help you understand the importance of specific attribute values for classification. Also, it can be used to gain insight into how classification is done, as well as to answer “what if” questions.

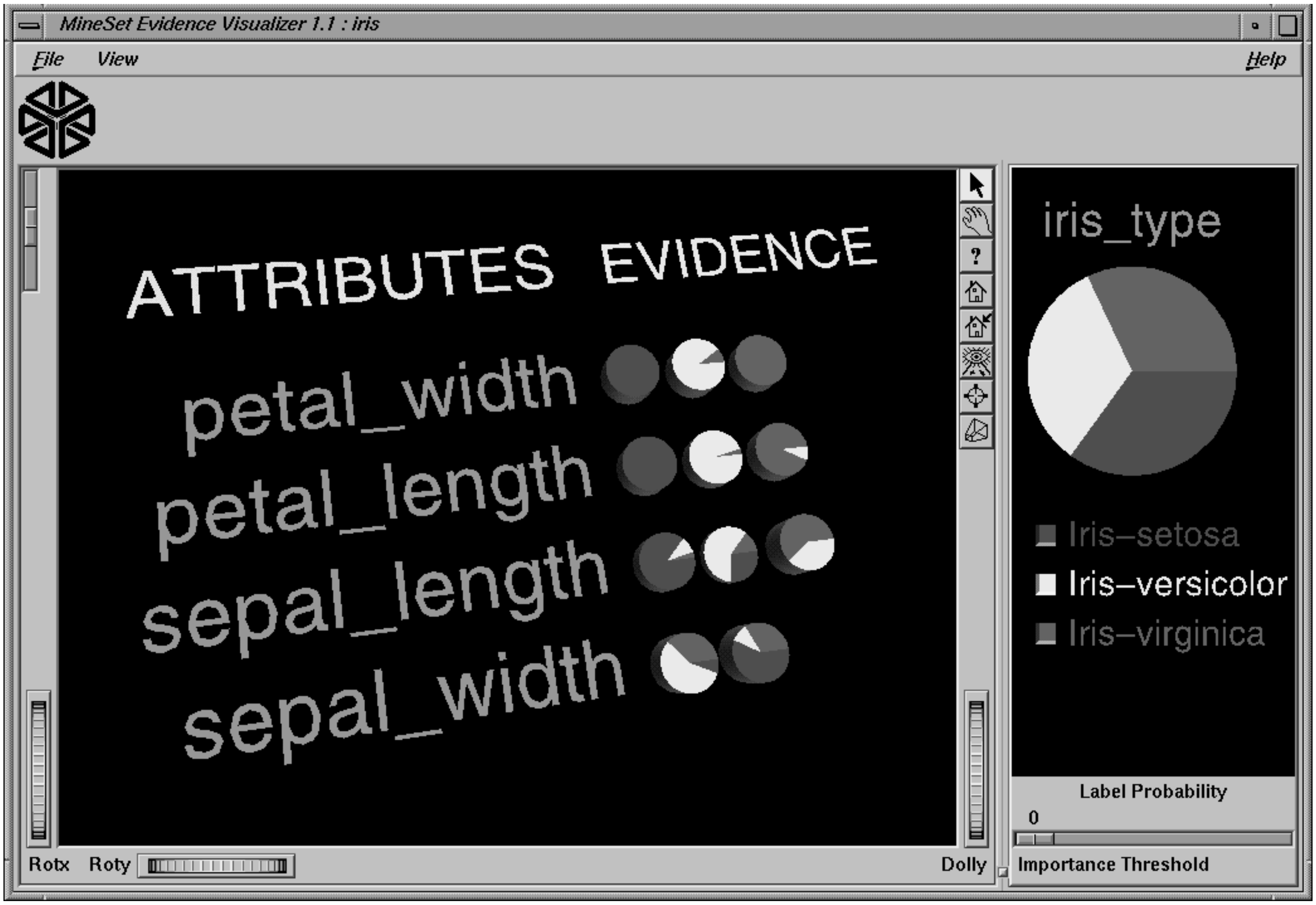


Figure 10-1 The Evidence Visualizer Applied to the Iris Dataset

The Evidence pane (on the left) consists of rows of pie charts or bars for the attributes used by the classifier. Characterization of a particular class label can be achieved by selecting one of the values in the Label Probability Pane (on the right). There is one pie chart or bar for each discrete value of the attribute. In the case where the attributes are not discrete, the continuous range has been discretized (binned) in a way that maximizes the differences between adjacent pie charts. Pie height is proportional to the number of records having that attribute value. The sum of the pie heights for every row is the same. The height of the graphical objects can be scaled to exaggerate the differences between the pie charts. By adjusting an importance threshold slider, attributes that are less useful for classification can be filtered out.

The kinds of questions you might answer by using the Evidence Visualizer are as follows:

- What is the likelihood that a new record, for which you know only the values for a few attributes, has a certain label?
- Which values of which attributes are the most useful for classifying the label?
- What is the distribution of records by attribute values?
- What are the characteristics of records that have a certain label?
- What is the probability that an attribute takes on a certain value given that it has a specific class label?

The *prior probability* for each class label is depicted in the pie chart in the Label Probability Pane, on the right of the screen. The prior probability for a class label is the probability of seeing this label in the data for a randomly chosen record, ignoring all attribute values. Mathematically, this is the number of records with the class label divided by the total number of records.

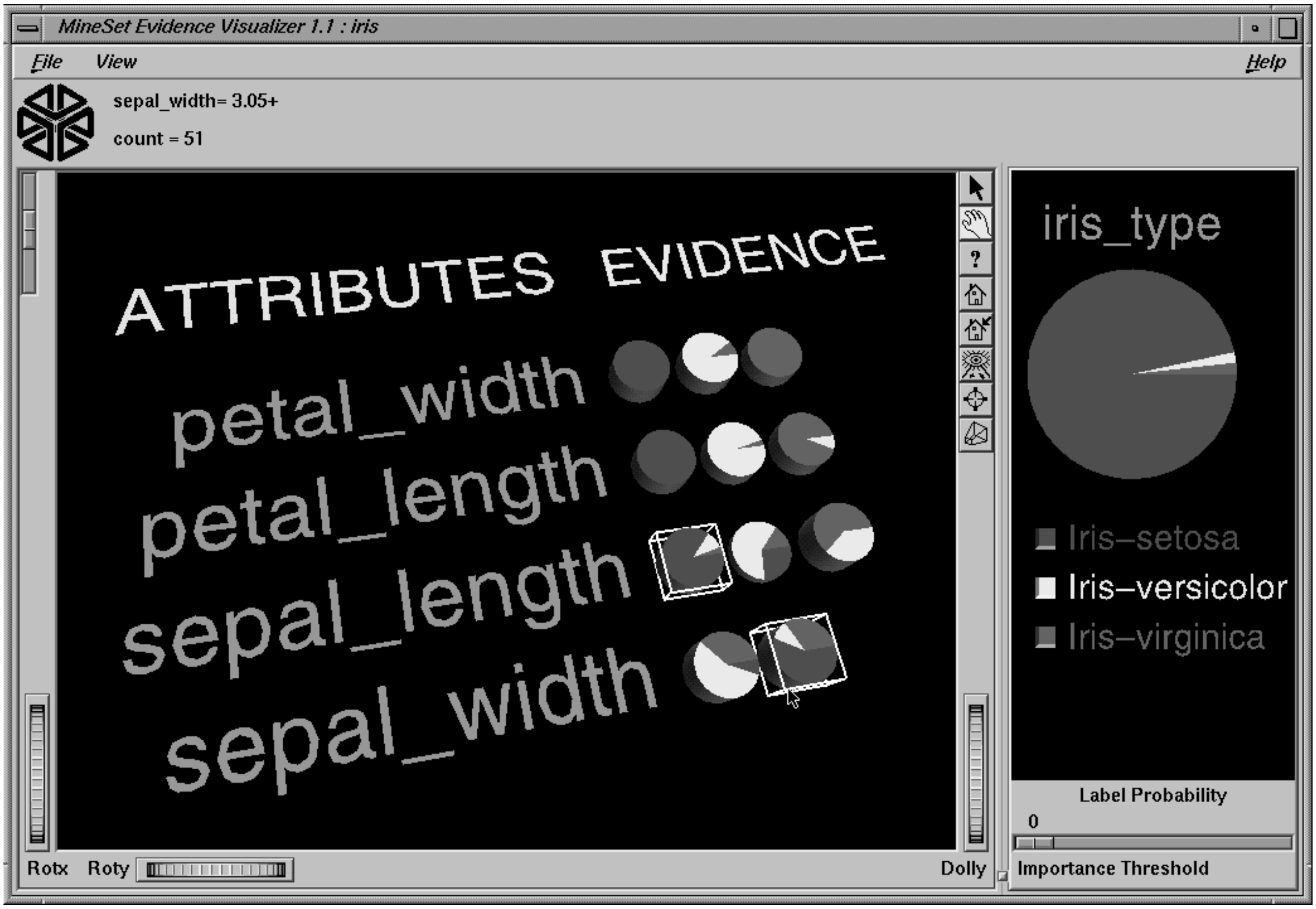
The *conditional probabilities*, depicted by pie charts in the visualization in the Evidence Pane on the left of the screen, show the relative probability of each attribute value given (conditioned on) each class label. The size of a pie slice indicates the amount of evidence the classifier adds to the prior probability after taking into account a given attribute value in a record. If the size of the slices are equal, the value is irrelevant, and the classifier adds the same amount of evidence to all classes.

Technically, the slice of the pie represents the normalized conditional probability of an attribute value A, given the class label L. The conditional probability, P(A | L), is the probability that a random record chosen only from records with label L takes the value A. Under the default settings, the probability is computed based on record counts. For example, P(0.75 < petal width ≤ 1.65 | iris-versicolor) is 91.6, because there are 36 records with label iris-versicolor, and 33 of them have a petal width in this range.

The Evidence Inducer, sometimes called Naive-Bayes (or Simple Bayes), builds a model that assumes the probabilities of each attribute value are independent given the class label. For example, this assumes that the four attributes (sepal_length, sepal_width, petal_length, and petal_width) are independent for each class of iris (*iris-setosa*, *iris-versicolor*, and *iris-virginica*). While this simplistic model is rarely true, the model is excellent for initial explorations of data and its classification prediction performance is very good in practical applications.

Each attribute value, or range of values, defines exactly one pie chart, which, in turn, gives the conditional probabilities for each class label. To classify a given record, one computes the probability of each class by multiplying its prior probability by the appropriate conditional probability from each row in the matrix. The final product gives the relative probability for each class and the highest value is the predicted class. If an attribute has an unknown value, it is ignored. (The unknown value does not add evidence to any of the classes.) The unknown values are denoted by a question mark (?).

This process of classification can be done interactively using the Evidence Visualizer. Simply select all the values for the attributes that you know. The probability pie on the right changes to show the distribution you would expect, given what you selected. For example, selecting the pie for sepal length < 5.45 inches and the pie for sepal_width > 3.05 inches shows that an iris with these characteristics belongs almost certainly to the class *iris-setosa* (see Figure 10-2).



sepal_width= 3.05+

count = 51

0

Figure 10-2 Selecting sepal_length < 5.45 and sepal_width > 3.05 Using the Iris Dataset

For some combinations of selected values, the probability pie on the right turns completely gray. This occurs when the values selected are contradictory according to the model. For example, in *iris.eviviz* there are no irises flowers that have petal_width < .75 inches and petal_length > 4.85 inches. Thus, selecting the two pies on the left representing these two values results in a gray pie on the right (see Figure 10-3).

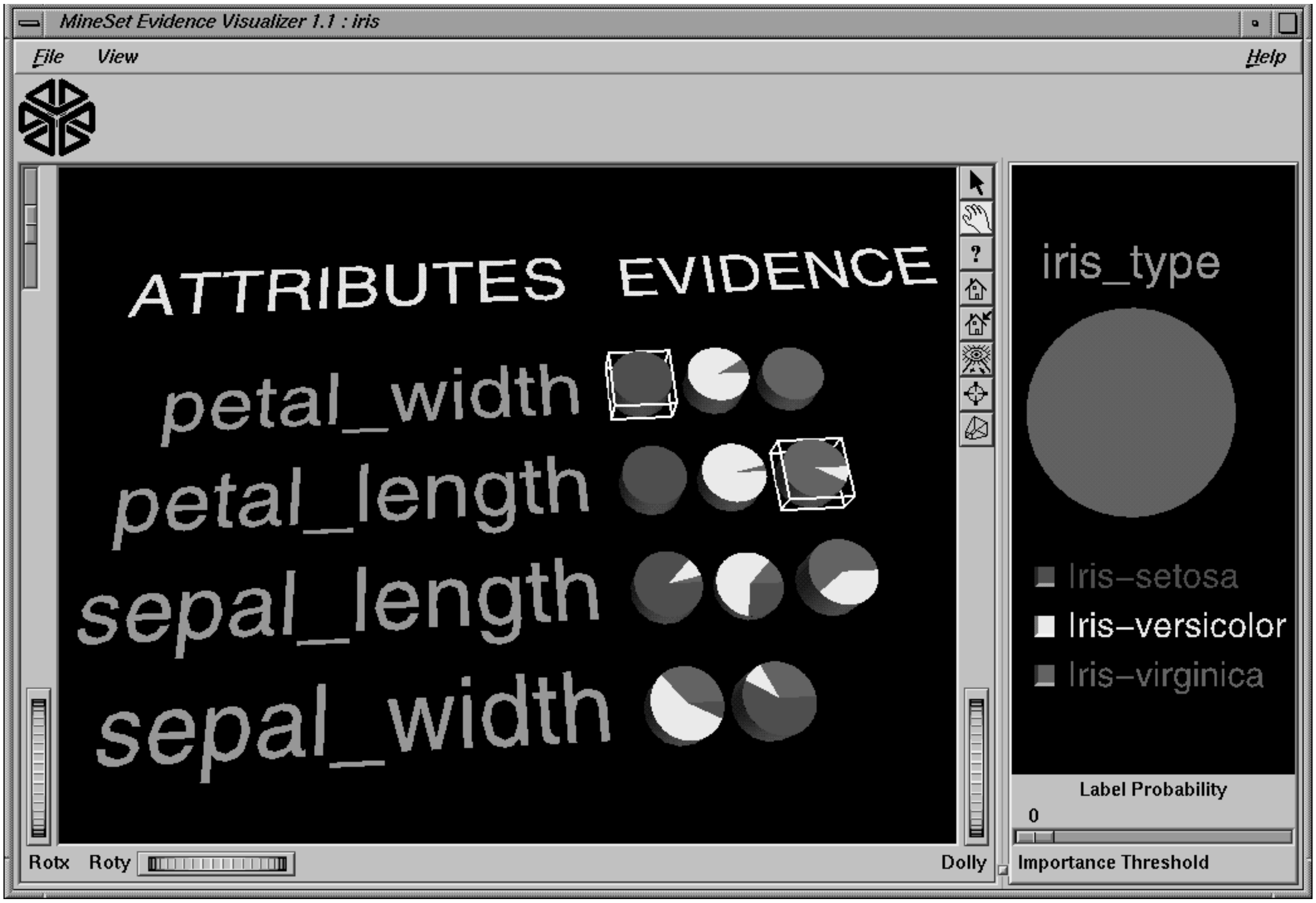
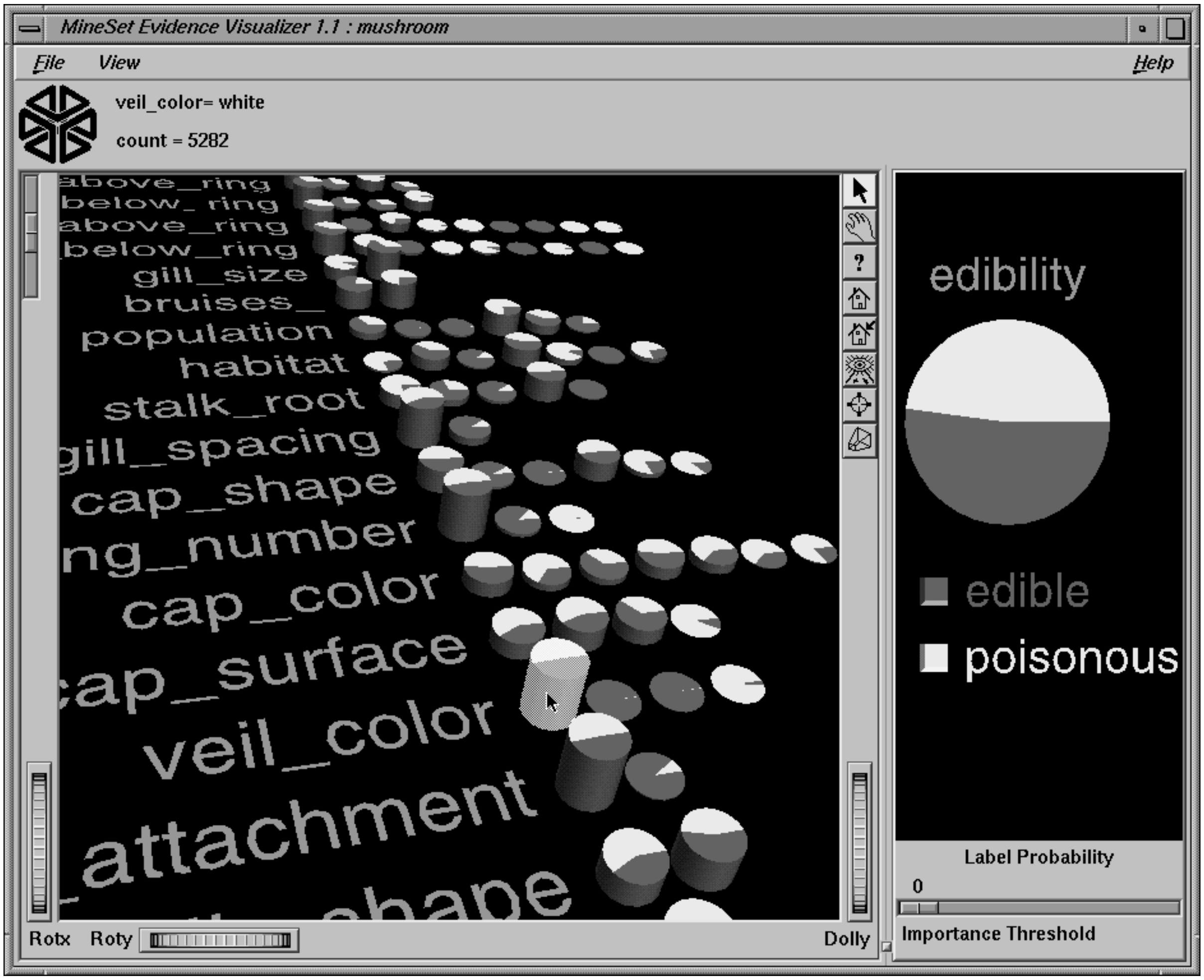


Figure 10-3 Selecting Two Contradictory Pies Results in a Gray Pie on the Right

Importance is a measure of predictive power with respect to a label. The Evidence Pane provides valuable insight not only into the importance of each attribute value affecting the class value, but also into the importance of specific attribute values. For example, in the mushroom dataset (described on page 312), the veil-color attribute is not very important because most of the time its attribute value is white (see Figure 10-4), which does not add much evidence to either class.



MineSet Evidence Visualizer 1.1 : mushroom

veil_color= white

count = 5282

Figure 10-4 Veil-Color Attribute in the Mushroom Dataset

However, if the veil color is brown or orange, the mushroom is likely to be edible, while if it is yellow, it is likely to be poisonous. Similarly, a “test for AIDS” might not be an important attribute for determining whether a patient has a deadly disease because most people would not test positive. However, the value POSITIVE for this test is highly informative because most patients that test positive do have a deadly disease (there might be test errors so this is not 100%).

Inducing Evidence Classifiers

The automatic induction of evidence classifiers is a process whereby counts are used to calculate the probabilities. Evidence classifiers are automatically induced (generated) from data. The data, which is made up of records and a label associated with each record, is called the *training set* (see Chapter 8).

The probabilities are generated using the following method:

1. All continuous attributes are discretized (binned), such that class distributions in these ranges are as different as possible. The number of ranges is determined automatically. To override the automatic binning, bin the given column with respect to the label using the Automatic Thresholds option under the Data Transformations’ *Bin Column* button.
2. The prior probabilities are the probabilities of each class in the training set.
3. The conditional probabilities are the probabilities of each attribute value conditioned on each class label in the training set.

The number of pies in a row is the number of discrete ranges produced by the inducer. If there is just one range, it means that this attribute by itself was not useful in predicting the label. The prior probabilities are what is initially displayed in the Label Probability Pane.

An optional Laplace correction can be applied to the probabilities, which avoids extreme probabilities (for example, probabilities of zeros and ones). We may prefer not to assign a probability of 1 to the event “a patient tested positive for AIDS has a deadly disease.” We may want to assign a probability close to 1 (but not 1), in order to allow for errors or unrepresentative samples. This is especially important if the number of records is small.

File Requirements

The Evidence Visualizer requires a training set, as described on page 237 of Chapter 8, “MineSet Inducers and Classifiers.” Files are generated by extracting data from a source (such as an ASCII file or a table in an Oracle, INFORMIX, or Sybase database). The Evidence Visualizer data file is output as a result of running the Evidence Inducer. The format of this file, which has a *.eviviz* extension, is described in Appendix F. When starting the Evidence Visualizer or when opening a file, you must specify the data filename. To apply the generated classifier, you should have a dataset of records with the same attributes as the training set, except that the label need not be present.

Running the Evidence Inducer

There are two ways to run the evidence inducer:

- From the Tool Manager

  To see the induction process, choose Client File from Tool Manager and type `/usr/lib/MineSet/data/iris.schema`. You’ll see four continuous attributes and one discrete attribute in the Data Transformation panel. Since there is only one discrete attribute, the label option automatically shows it. Select the Evidence Inducer, and ensure you have selected the *Classifier & Accuracy* mode. To run the Inducer, click *Go!*.

  You’ll see a small popup window with information and statistics, followed by another window showing the classifier structure in the Evidence Visualizer.

- From the command line

  To induce an evidence classifier from the command line, refer to Appendix E, “Command-Line Interface to MIndUtil: Classifiers, Discretization, Column Importance, and File Conversions.”

Starting the Evidence Visualizer

There are six ways to start the Evidence Visualizer:

- Run the Evidence Inducer from the Tool Manager under the Classifiers tab. After the inducer builds the classifier, it automatically invokes the Evidence Visualizer. See below for details about using the Tool Manager in conjunction with the Evidence Visualizer.
- Use the Tool Manager to start the Evidence Visualizer from the Visual Tools menu. (See Chapter 3 first for details on the Tool Manager's functionality, which is common to all MineSet tools.)
- Double-click the Evidence Visualizer icon on your Indigo Magic desktop. The startup screen requires you to select a data file by choosing File > Open.

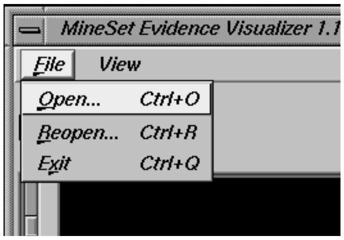


Figure 10-5 File > Open Menu Selection

- If you know what configuration file you want to use, double-click the icon for that configuration file. This starts the Evidence Visualizer and automatically loads the configuration file you specified. This works only if the configuration filename ends in *.eviviz* (which is always the case for configuration files created for the Evidence Visualizer via the Tool Manager).

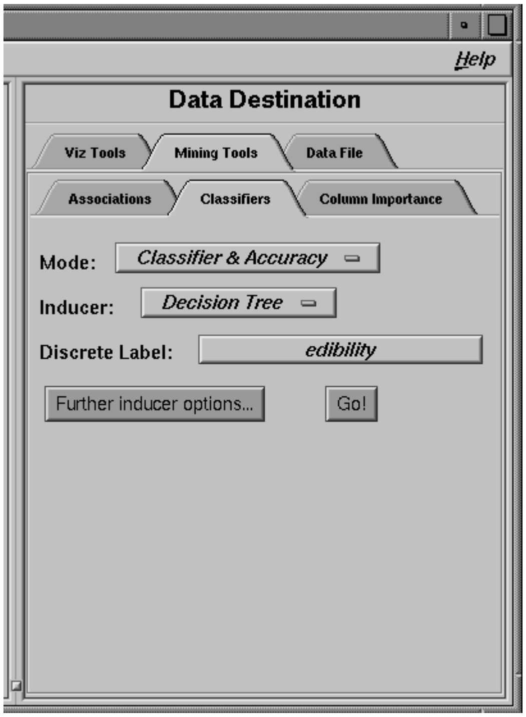
- If you know what configuration file you want to use, drag its icon onto the Evidence Visualizer icon. This starts the Evidence Visualizer and automatically loads the configuration file you specified.
- Start the Evidence Visualizer from the UNIX shell command line by entering this command at the prompt:

  ```
  eviviz [ dataFile ]
  ```

  Here, *dataFile* is optional and specifies the name of the configuration file to use. If you don't specify a configuration file, you then must use File > Open to specify one (see Figure 10-5).

Configuring the Evidence Inducer Using the Tool Manager

To access the options for configuring the Evidence Inducer, select the *Mining Tools* tab on the Data Destination panel (Figure 10-6). From the subsequent tabs, select Classifiers. Ensure that the inducer you select is Evidence (the default). Your selections in the Mode and Algorithm menus determine the options available in the Further Inducer Options menu. After you have made your selections in these menus, click *Go!* to run the inducer, which, in turn, creates the classifier.



Mode: Classifier & Accuracy

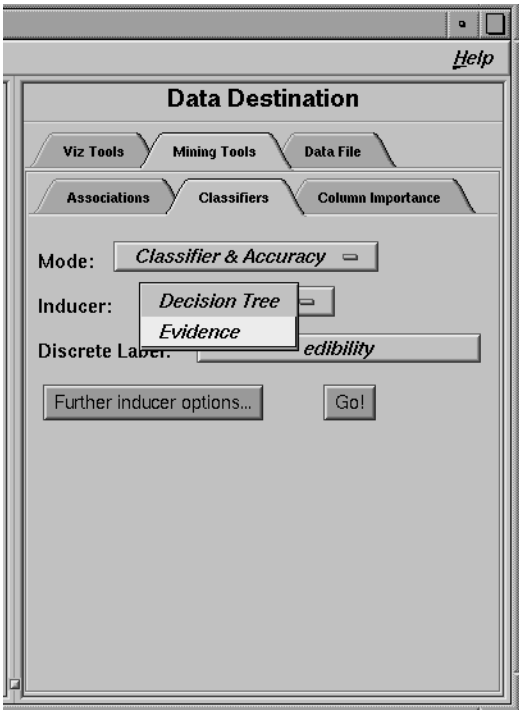
Inducer: Decision Tree

Discrete Label: edibility

Figure 10-6 Tool Manager With Data Destination Panel Showing Classifiers

Inducers

Figure 10-7 shows the two possible induction algorithms for creating the two MineSet classifiers.



Data Destination

Mode: Classifier & Accuracy

Inducer: Decision Tree

Discrete Label: edibility

Figure 10-7 The Algorithm Pulldown Menu of the Data Destination Panel

To build an Evidence classifier, select Evidence. The Decision Tree classifier is discussed in Chapter 9.

Discrete Labels

The Discrete Labels menu provides a list of possible discrete labels. Discrete attributes (binned values, character string values, or a few integers) have a limited number of values. You should select a label attribute with few values; for instance, two or three (see “Training Set” in Chapter 8). If there are no discrete attributes, the menu shows No Discrete Label, and the *Go!* button is disabled. You then must create a discrete attribute by binning or adding a new column using the Tool Manager’s Data Transformations panel.

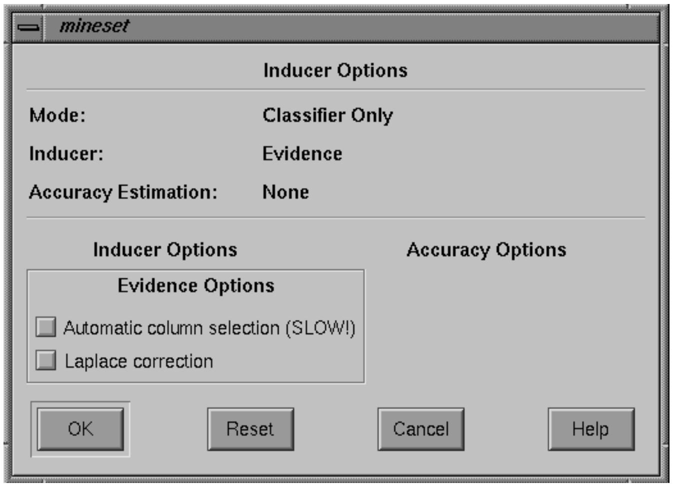
Classifier Name

The generated classifier is named with the prefix of the session filename (as determined in Tool Manager) with the suffix *-evi.class*. By default, all classifiers are stored on the server. These classifiers can be used for future classification of unlabeled records; that is, they can be used to predict the labels for unlabeled datasets (see “The Apply Classifier Button” in Chapter 3).

Refining the Inducer With Further Options

Selecting Further Inducer Options causes the Inducer Options dialog box to appear (see Figure 10-8). This dialog box consists of three panels:

- The top panel shows the choices you made in the Tool Manager’s Data Destination panel. The type of Accuracy Estimation is determined by the model.
- The bottom-left panel lets you specify further Algorithm Options.
- The bottom-right panel lets you specify the Accuracy Options (unless the mode you chose in the Data Destination panel was Classifier Only, in which case this area is empty). The options shown in this panel depend on the type of Accuracy Estimation you chose (see “Accuracy Estimation” in Chapter 8).



mineset

Inducer Options

Mode: Classifier Only

Inducer: Evidence

Accuracy Estimation: None

Inducer Options

Evidence Options

□ Automatic column selection (SLOW!)

□ Laplace correction

Accuracy Options

Figure 10-8 Classification Options Dialog Box Without Accuracy Estimate

Evidence Inducer Options

By choosing Further Inducer Options, you can fine-tune the Evidence inducer.

- Automatic column selection

  This applies a process that chooses only those columns that help prediction the most. Because extra columns can degrade the prediction accuracy of the evidence classifier, this process searches for a good subset of the columns automatically. Only those columns found to be useful are used. This process can take a long time, especially if there are many columns.

  The selection of columns is done by estimating the accuracy of different attribute sets using the wrapper approach. See Appendix I, “Further Reading and Acknowledgments,” for details.

- Laplace correction

  This biases the probabilities towards the average, thus avoiding extreme numbers (such as zero and one). The fewer the records in a bin, the more it will be changed towards the average.

Working in the Evidence Visualizer's Panes

If you started the Evidence Visualizer without specifying a configuration file, the main screen shows the copyright notice for the Evidence Visualizer. Only the File and Help pulldown menus are available. To view all menus and controls in the main window, open a configuration file. Use File > Open (see Figure 10-5) to see a list of configuration files.

When a valid configuration file is specified, the two panes in the main screen display graphics. For example, specifying *cars.eviviz* results in the output displayed in Figure 10-9.

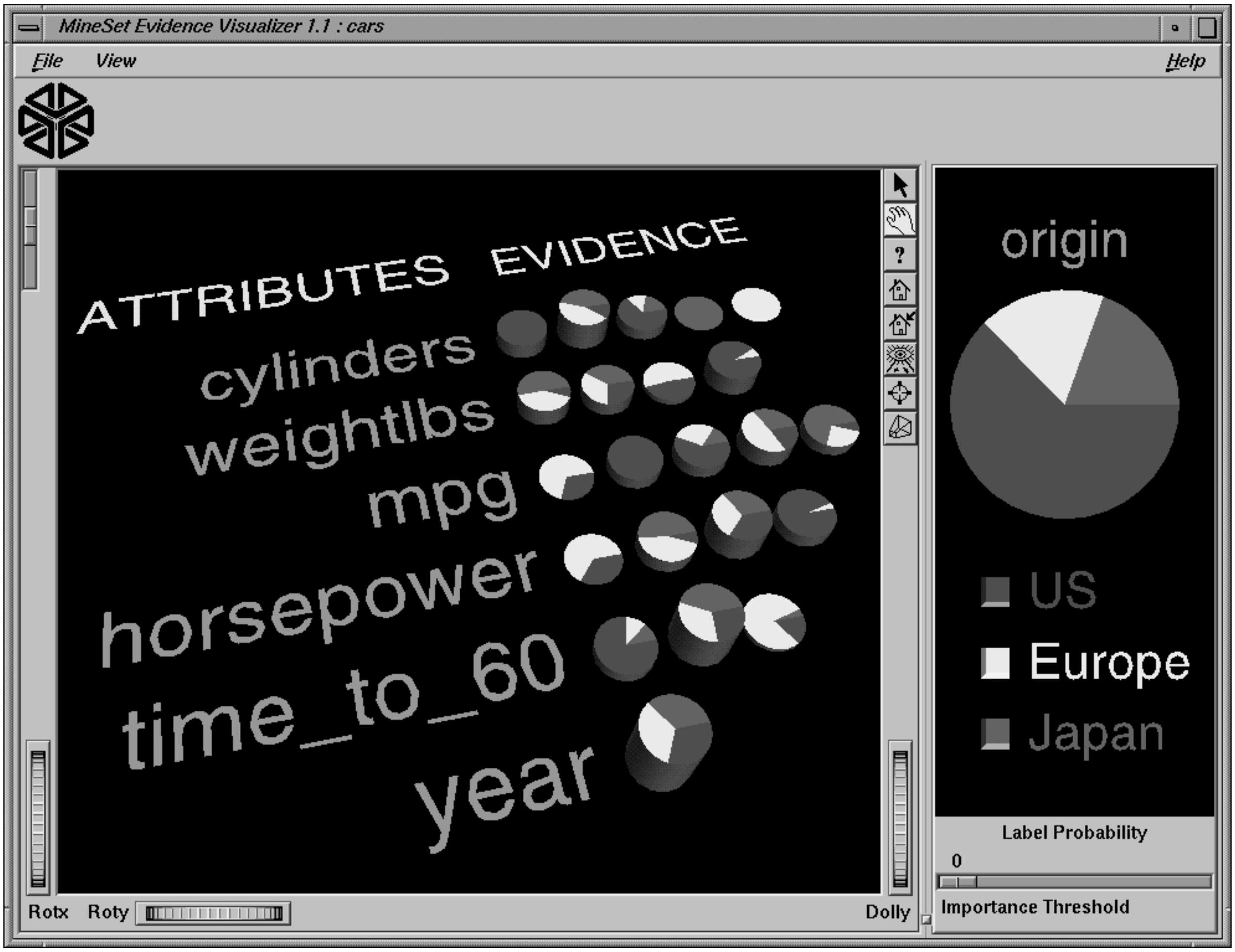


Figure 10-9 Evidence Visualizer Window for cars.eviviz

In the Evidence Pane on the left, one row of pie charts appears for each attribute in the dataset that the classifier is using. Each pie chart corresponds to a value for the attribute associated with the row. In the Label Probability Pane on the right, a list of all class labels appears under a large pie chart of the prior probability distribution. Note that the color of the slices correspond to the color associated with each class label. This prior probability represented by the pie shows the proportion of data with each class label.

Viewing Modes

Each of the Evidence Visualizer's main window panes has two modes of viewing: grasp and select.

Viewing Modes in the Label Probability Pane

The Label Probability Pane is located on the right of the Evidence Visualizer's main window. The top two buttons of those aligned vertically between the panes toggle between the grasp and select modes. Alternatively, the Esc key also toggles the viewing mode for both panes.

In grasp mode, the cursor appears as a hand that lets you pan and scale the scene's size.

- To pan (translate) the display, press the middle mouse button and drag it in the direction you want the display panned.
- To enlarge the scene, press the left mouse button, and drag the mouse downward.
- To shrink the scene, press the left mouse button, and move the mouse upward. Moving the cursor over the buttons next to the label values causes the size of the slice to be shown at the top.

Viewing Modes in the Evidence Pane

The Evidence Pane is located on the left of the Evidence Visualizer’s main window. The top two buttons of those aligned vertically between the panes toggle between the grasp and select modes. Alternatively, the Esc key also toggles the viewing mode for both panes.

In grasp mode, the cursor appears as a hand, so you can pan, rotate, and scale the scene’s size. The Label Probability pane contains only 2D geometry; thus, rotation is disabled.

- To rotate the display, press the left mouse button and move the mouse in the direction you want. (Also see “Thumbwheels” on page 303.)
- To pan (translate) the display, press the middle mouse button, and drag it in the direction you want the display panned.
- To enlarge the viewpoint, simultaneously press the left and middle mouse buttons and move the mouse downward. To shrink the viewpoint, simultaneously press the left and middle mouse buttons, and move the mouse upward. This is equivalent to the functions provided by the Dolly thumbwheel.

Selecting Items in the Label Probability Pane

In select mode, the cursor appears as an arrow. You can then select one of the class labels by clicking the button to the left of it. Once a value is selected, a white box appears around the button next to the label (see Figure 10-10). The size of that slice (the probability of predicting that label value) appears in the text output line at the top. To deselect a value, click it again.

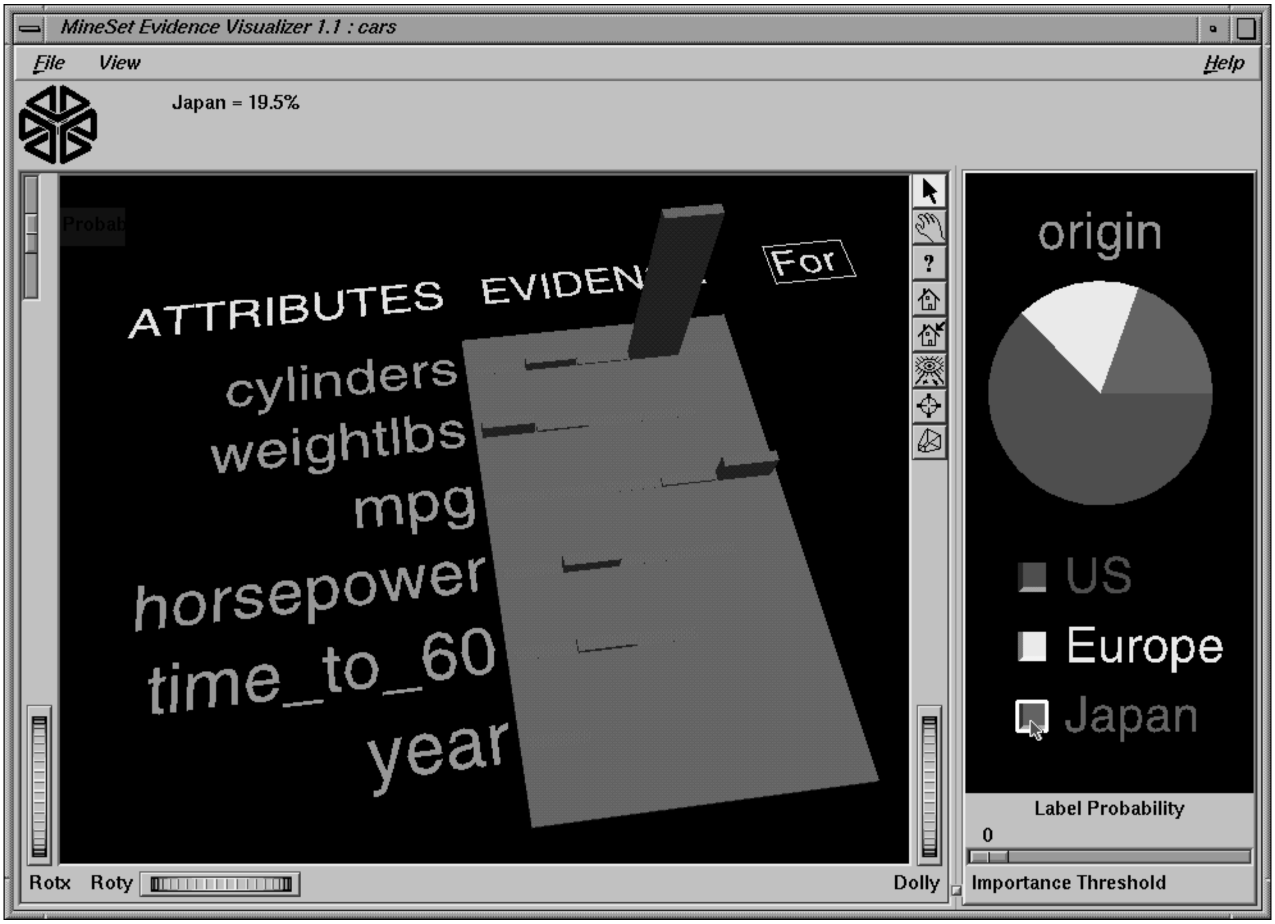


Figure 10-10 Label Value “Japan” Selected Using the Cars Dataset

If no label is selected, the Evidence Pane on the left displays pie charts. The pie charts show the effect a certain attribute value has on the distribution in the Label Probability Pane.

If a label is selected, the representation on the left displays bar charts. The height of each bar shows the evidence in favor of the selected label value. Technically, *evidence for* is the negative log of the quantity one minus the size of the slice matching the selected label in the corresponding pie of the pie chart representation.

As the default, the amount of evidence common to all the labels is subtracted. This means that the height of a bar for each value is reduced by the height representing the label for which the evidence is smallest. If you select a different label, the bars and their colors change to represent the new class label. Selecting the same label again deselects it, and the Evidence Pane again displays the pie charts. Uncheck the View > Subtract minimum evidence option if you do not want to subtract the common evidence.

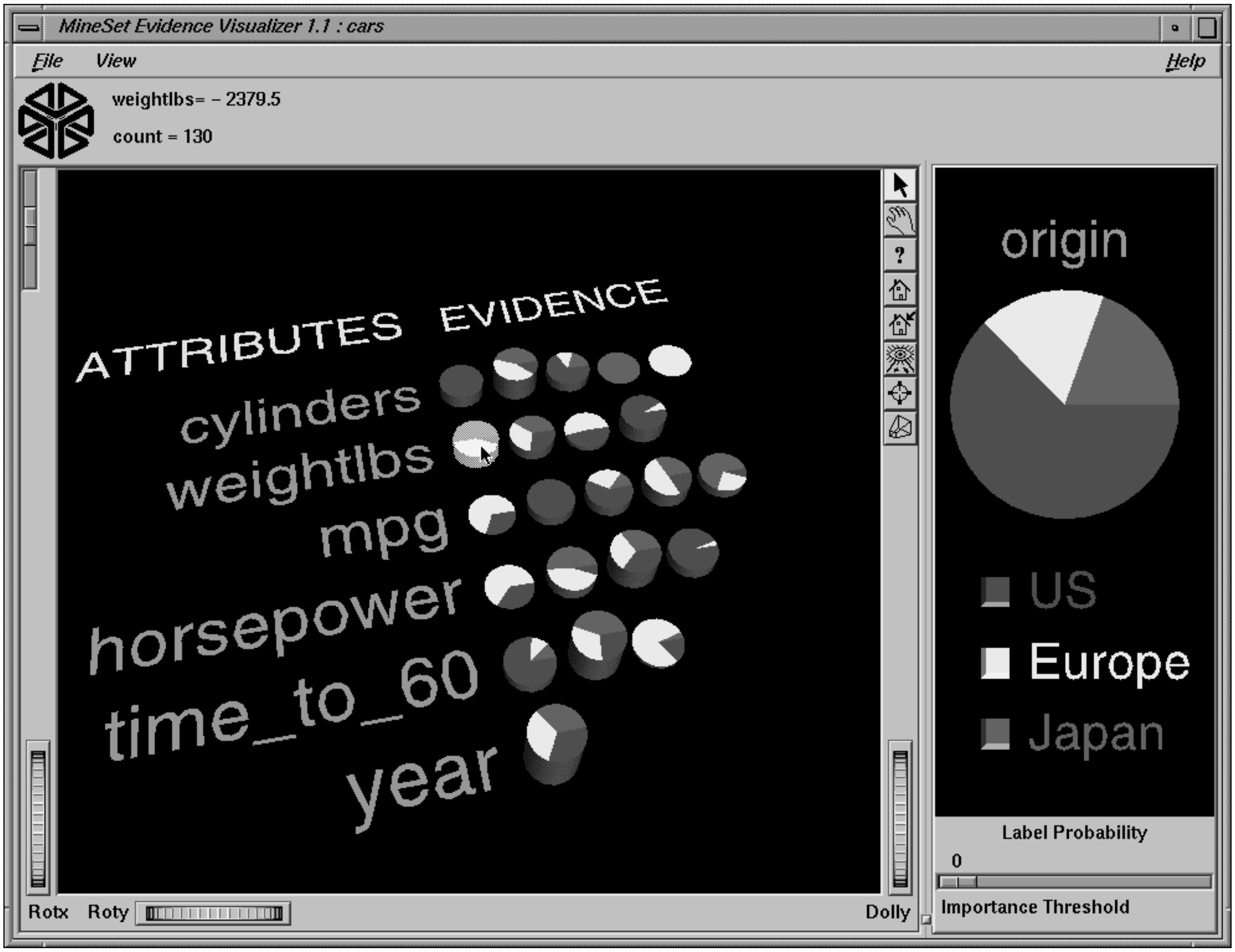
Selecting Items in the Evidence Pane

In select mode, the cursor appears as an arrow. You can highlight an object (either a pie chart or a bar) by moving the cursor over that object. Information about that object then appears above the Evidence Pane. The information is displayed as long as the cursor is over the object.

- If the object is a pie chart, then the message takes this format:

  <attribute name>: <value or range>
  count = <count>

  Here, *count* is the number of data points that fall in that range or have that value for that attribute (see Figure 10-11). The pie height is proportional to this number.



weightlbs= – 2379.5

count = 130

Figure 10-11 Pie Charts With the First Binned Range of weightlbs Highlighted

- If the object is a bar, then the message takes this format:

  (<attribute> = <value>) ==> Prob(<selected label>) = x% [low%-high%]
  Evidence=z
  <selected label> ==> Prob(<attribute> = <value>) = y% [low%-high%]

  Here, *x* is the probability that a record has the selected label given that it has the highlighted attribute value. The bracketed range, [low%-high%] gives the 95% confidence interval. Similarly, y% is the probability that a record has the highlighted attribute value given the selected label (see Figure 10-12). Note that the height of the bar shows evidence, not probability. The amount of evidence, z, is directly related to the bar heights. Evidence can be summed in order to determine which class is predicted (unlike probability, which must be multiplied).

  Technically, *evidence for* is defined as

$$-\log \left[1 - \frac{P(A \mid L)}{\sum_{i=1}^{N} P(A \mid L_i)} \right]$$

  while *evidence against* is defined as

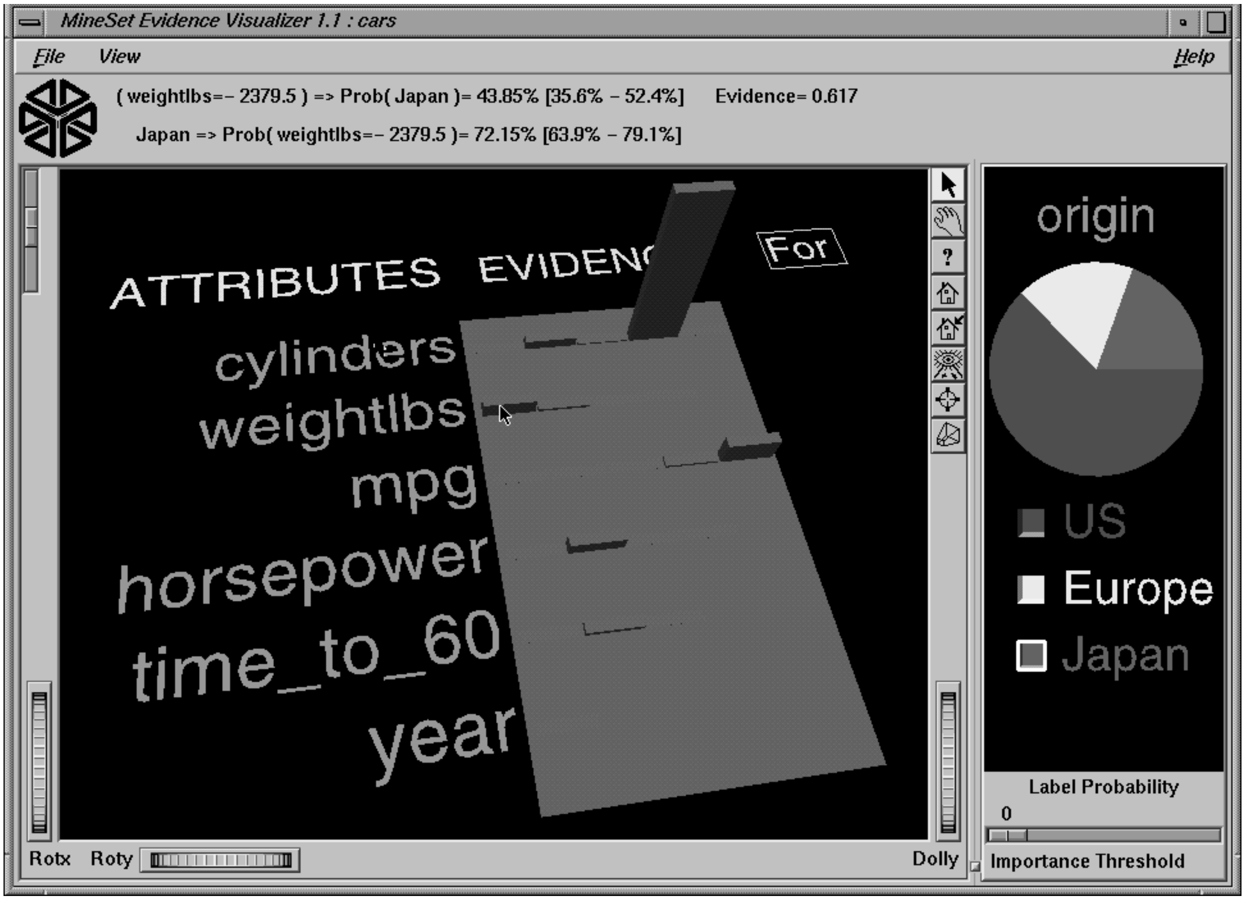
$$-\log \left[\frac{P(A \mid L)}{\sum_{i=1}^{N} P(A \mid L_i)} \right]$$

  A is the attribute value, L is the selected label value, and N is the number of label values. When computing the bar heights, a very small number is added inside the brackets of the above expressions to prevent the bars from becoming infinitely tall. The word “for” or “against” in the Evidence Pane has a box around it to indicate that it may be clicked on. Do this to toggle the representation.

  The height of the gray rectangular base (on which the bars stand) represents the amount of evidence contributed by the prior probability. For example, if the label is car cylinders, there are very few three cylinder cars, so the base is low when *evidence for* is showing, and high when *evidence against* is showing. You can add to this height the height of individual bars that are on top.

  Evidence for can be useful in determining which values are the most helpful in predicting a particular label value.

The amount of evidence (bar height) is not derived directly from either probability shown while highlighting. Instead, the evidence depends on the conditional probability relative to the other probabilities for all the other label values according to the equation above.



(weightlbs=– 2379.5) => Prob(Japan)= 43.85% [35.6% – 52.4%]   Evidence= 0.617

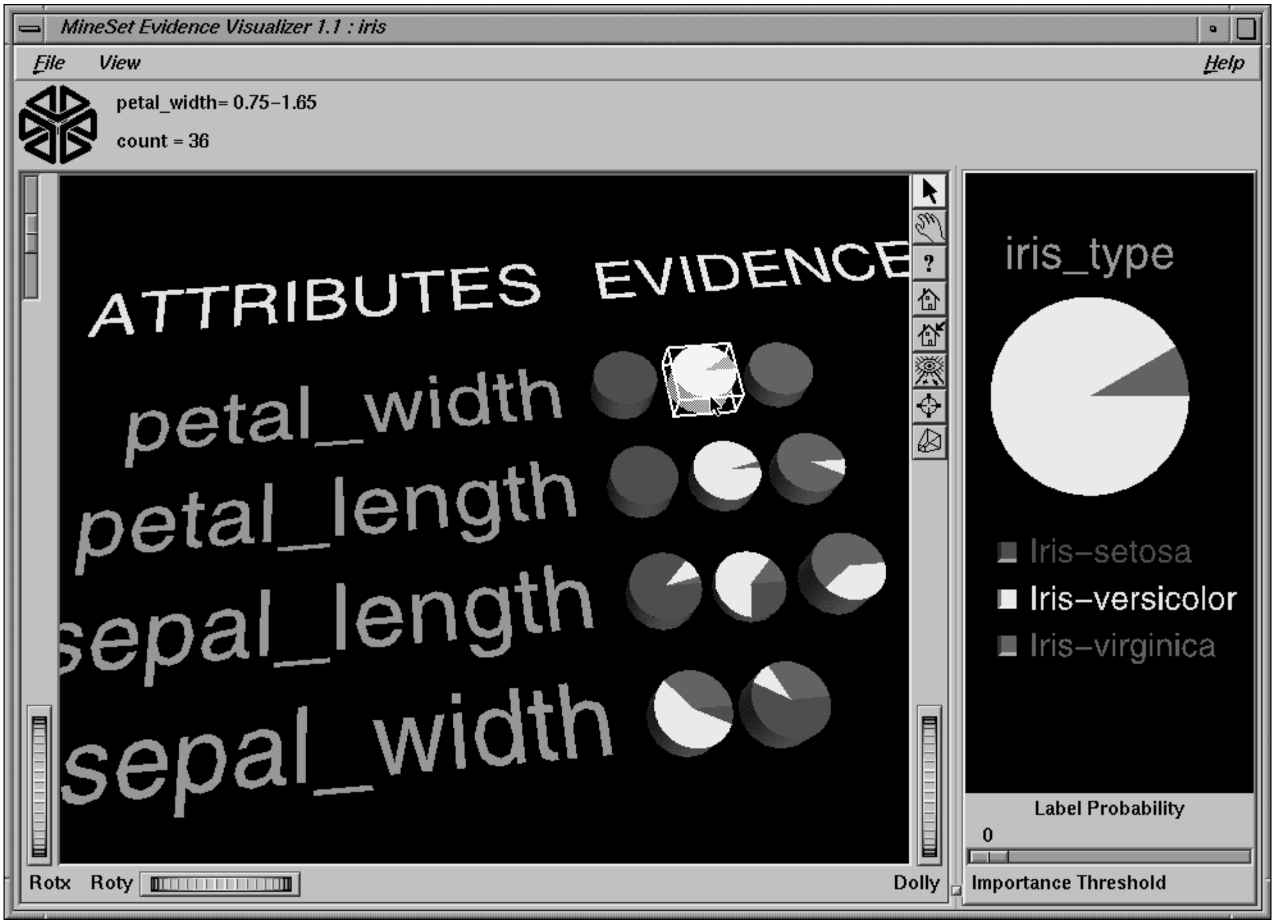
Japan => Prob(weightlbs=– 2379.5)= 72.15% [63.9% – 79.1%]

Figure 10-12 Bar Chart With the First Binned Range of weightlbs Selected

You can also select one pie chart or bar from an attribute row by clicking the left mouse button while the cursor is over one of the attribute values. This causes the object to be drawn with a white bounding box surrounding it (see Figure 10-13). Note that it is not possible to select a pie chart corresponding to an unknown value of an attribute (if one exists, it is in the first position of the row and denoted by a question mark). Trying to do so results in a beep. The large pie chart in the Label Probability Pane on the right changes to reflect the item you select; it now shows the posterior probability, given the attribute value that was just selected. The Evidence Visualizer arrives at this new probability distribution by multiplying the probabilities of all the selected objects together, then multiplying this result with the prior probability.

Note that this multiplication corresponds to a conditional independence assumption. When this assumption is not appropriate, and multiple values for attributes are chosen, the predicted class probabilities are likely to be too extreme, although the final classification might be correct. The estimated accuracy shown in the information and statistics window when you run the inducer can help you determine how reasonable this assumption is. If the accuracy is high, the assumption is reasonably robust in the domain.

Before clicking on a pie, the Evidence Visualizer appears as shown in Figure 10-1. This shows that given no additional information, there is an approximately equal likelihood that an iris will be designated type *iris-setosa*, *iris-versicolor*, or *iris-virginica*. If you click a pie for petal_width .75 - 1.65, the pie on the right changes to that shown in Figure 10-13. This indicates that if the petal width is between .75 and 1.65, the iris probably belongs to the class *iris-versicolor*. You then can select additional values to further change the distribution, but you can select at most one pie or bar from each row. The order in which you select pies or bars does not matter.



petal_width= 0.75–1.65

count = 36

Figure 10-13 Iris Dataset With the Value petal_width .75 - 1.65 Selected

When a particular label has been selected in the Label Probability Pane, the Evidence Pane shows bars rather than pies for each value of an attribute. The title over the bars reads `Evidence For`. The box around the `For` indicates that it can be selected (Figure 10-14).

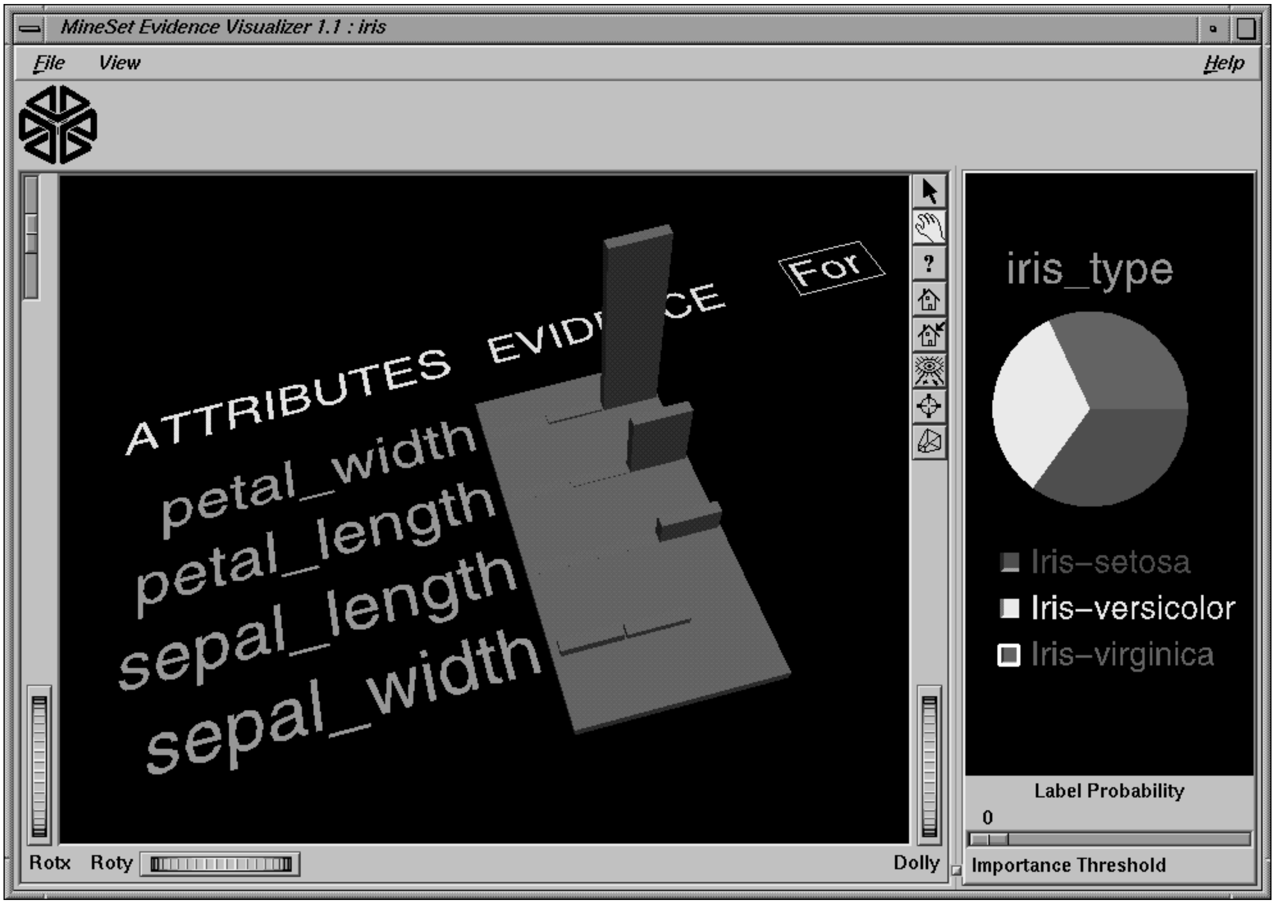


Figure 10-14 Bars Showing Evidence for iris-virginica

Clicking the `For` in the `Evidence For` title toggles it to display `Against`. As a result, the bar heights change to show evidence against the label (Figure 10-15).

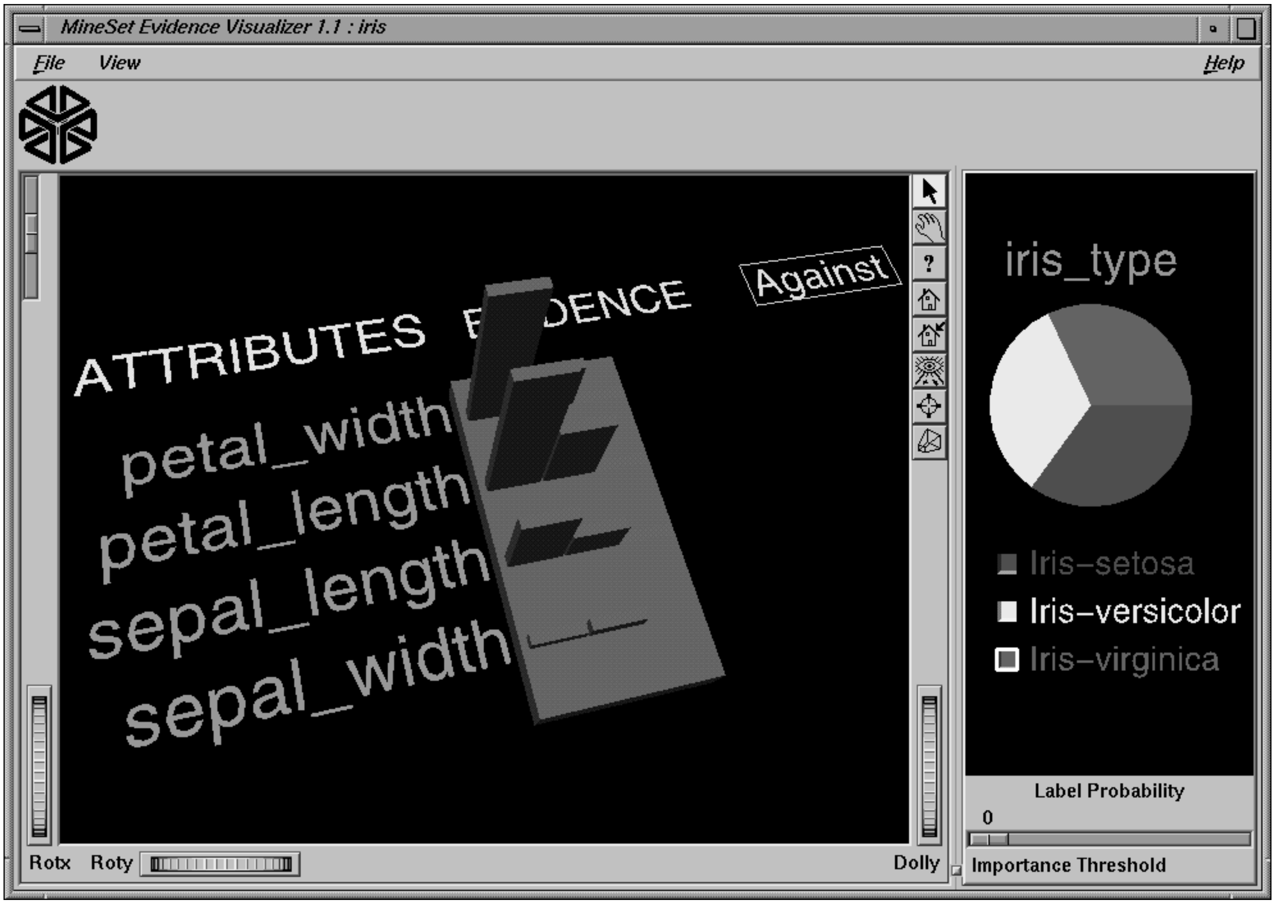


Figure 10-15 Bars Showing Evidence Against iris-virginica

Selecting bars has the same effect on the large probability pie in the Label Probability Pane to the right as did selecting pies. The bar height indicates the amount of evidence for or against the selected label contributed by that selected value. Since log probabilities are used to represent evidence, the bar heights are added to accumulate evidence (whereas probabilities must be multiplied).

External Controls

Several external controls surround the Evidence Pane: buttons, thumbwheels, and sliders. This section describes each type of control.

At the top right of the Evidence Pane area are eight buttons (Figure 10-16). These buttons are described below.

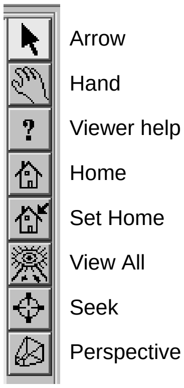


Figure 10-16 Evidence Pane Buttons

- *Arrow* puts you in select mode for both panes. When in this mode, the cursor becomes an arrow. Select mode lets you highlight, or select, entities in the Evidence Pane or select labels in the Label Probability Pane.
- *Hand* puts you in grasp mode for both panes. When in this mode, the cursor becomes a hand. Grasp mode lets you rotate, zoom, and pan the display in the Evidence Pane, or pan and zoom in the Label Probability Pane.
- *Viewer Help* brings up a help window describing the viewer itself.
- *Home* takes you to a designated location. Initially, this location is the first viewpoint shown after invoking the Evidence Visualizer and specifying a configuration file. If you have been working with the Evidence Visualizer and have clicked the *Set Home* button, then clicking *Home* returns you to the viewpoint that was current when you last clicked *Set Home*.

- *Set Home* makes your current location the home location. Clicking the *Home* button returns you to the last location where you clicked *Set Home*.
- *View All* lets you view the entire graphic display, keeping the angle of view you had before clicking this option. To get an overhead view of the scene, rotate the camera so that you are looking directly down on the entities, then click the *View All* button.
- *Seek* takes you to the point or object you click after selecting this button.
- *Perspective* is a button that lets you view the scene in 3D perspective (closer objects appear larger; farther objects appear smaller). Clicking this button again turns 3D perspective off.

  If Perspective is off, the Dolly thumbwheel becomes the Zoom thumbwheel. (The Dolly thumbwheel is described in "Thumbwheels" on page 303.)

Sliders

The Evidence Visualizer contains two sliders: Height Scale and Importance Threshold.

The Height Scale Slider (Figure 10-17), which is located in the upper left of the Evidence Visualizer, scales the height of the pies and bars. You can use this slider to magnify small differences.

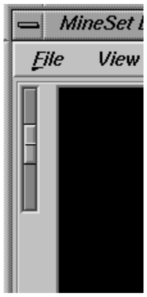


Figure 10-17 Evidence Visualizer Height Scale Slider

The Importance Threshold Slider, located at the bottom right of the Evidence Visualizer window (Figure 10-18), filters out attributes that are not as useful for classifying the selected label. This quality, assigned a value between 0 and 100 by the inducer, is called *importance*. This measure is on an absolute scale. To understand how importance is calculated, see “Column Importance and Relation to Classifiers” on page 321. As the slider is moved to the right, attributes that fall below the requisite importance value are removed from the scene. If the attributes are sorted by importance (the default), then the ones at the bottom are the first to be removed.

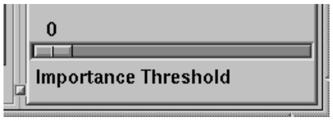


Figure 10-18 Evidence Visualizer Importance Threshold Slider

Thumbwheels

Three thumbwheels appear around the lower part of the main window border (see Figure 14). They let you dynamically move the viewpoint. Rotx and Roty rotate the scene about the *x* or *y* axis, respectively. The dolly thumbwheel moves the virtual cameral forward or backward.

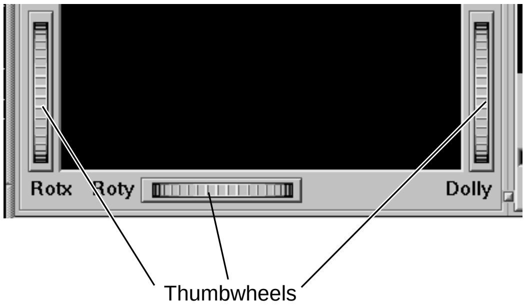


Figure 10-19 Evidence Pane Thumbwheels

Note: If *Perspective* is off, the Dolly thumbwheel becomes the Zoom thumbwheel.

Pulldown Menus

Three pulldown menus let you access additional Evidence Visualizer functions: File, View, and Help. If you start the Evidence Visualizer without specifying a configuration file, only the File and the Help menus are available.

The File Menu

The File menu (Figure 10-5) lets you open a new configuration file, reopen the current configuration file, or exit the Evidence Visualizer.

The View Menu

The View menu lets you control certain aspects of what is shown in the Evidence Visualizer pane (Figure 10-20).

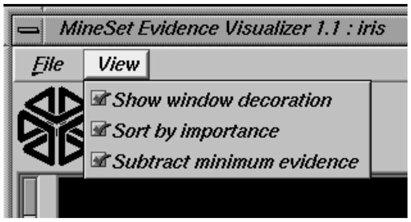

MineSet Evidence Visualizer 1.1 : iris
File
View
☑ Show window decoration
☑ Sort by importance
☑ Subtract minimum evidence

Figure 10-20 Evidence Visualizer’s View Menu

This menu contains three options:

- Show Window Decoration lets you hide or show the external controls around the main window.
- Sort By Importance lets you display the attributes sorted according to their usefulness in classifying with respect to the chosen label. If this option is turned off, then the attributes will appear in the same order they did under “Current Columns” in the Tool Manager.
- Subtract Minimum Evidence applies only when a label has been selected and the bars are shown. With this option on (the default), the height that is the minimum over all the label values is subtracted. This amount may be different for each value of each attribute, but for a given attribute value, the amount subtracted is constant across label values. Activating this option magnifies small differences by subtracting the least common denominator among all the label values.

The Help Menu

The Help menu provides access to five help functions (see Figure 10-21).

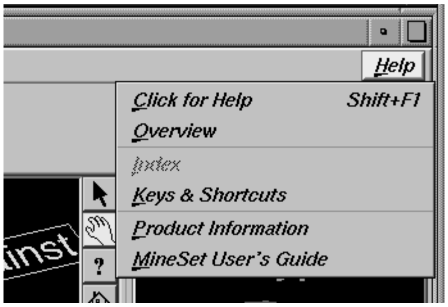


Figure 10-21 Evidence Visualizer’s Help Menu

- Click for Help turns the cursor into a question mark. Placing this cursor over an object in the Evidence Visualizer pane, and clicking the mouse, causes a help screen to appear; this screen contains information about that object. Closing the help window restores the cursor to its arrow form and deselects the help function. The keyboard shortcut for this function is Shift+F1. (Note that it also is possible to place the arrow cursor over an object and press the F1 function key to access a help screen about that object.)
- Overview provides a brief summary of the major functions of this tool, including how to open a file and how to interact with the resulting view.
- Index provides an index of the complete help system. This option is currently disabled.
- Keys & Shortcuts provides the keyboard shortcuts for all of the Evidence Visualizer's functions that have accelerator keys.
- Product Information brings up a screen with the version number and copyright notice for the Evidence Visualizer.
- MineSet User's Guide invokes the IRIS Insight viewer with the online version of this manual.

Sample Files

The following examples show cases in which classifiers might be useful. Each of these examples is associated with a sample dataset provided with MineSet. By running the inducer, you can generate the *.eviviz* files described below.

Note: The data files, which have a *.schema* extension, are located in */usr/lib/MineSet/data* on the client workstation. The classifier visualization files, which have a *.eviviz* extension, reside on the client workstation in */usr/lib/MineSet/eviviz/examples*.

Origin of Cars

The *cars* dataset contains information about different models of cars from the 1970s and early 1980s. Attributes include weight, acceleration, and miles per gallon (mpg). The file */usr/lib/MineSet/eviviz/examples/cars.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/cars.schema* with the label set to origin (Japan, U.S., Europe) and the cylinders column changed to type string. The cylinders were changed to type string in order to see all values and avoid the automatic discretization.

If you have a dataset of car attributes, you might want to know what characterizes cars of different origins.

From the distribution of label values in the pie on the right we can see that most cars in this dataset were made in the U.S. (62.1%) and a smaller number in Japan (19.5%) and Europe (18.4%). Clearly brand is the best predictor of origin, since each brand is associated with only one country. For this reason it has the highest importance, and is at the top of the list. By looking at the height of the pies it can be seen that many cars have four cylinders, most weigh less that 3000 lbs and most can reach 60 miles per hour in less than 20 seconds but more than 13.

Look at the distribution of slices for individual attribute values. If a car has engine size >169 cubic inches it is almost surely made in the U.S., and there is no chance it was made in Japan. Other pies show us that U.S. cars generally have six or eight cylinders, low miles per gallon, high horsepower (over 122), heavy weight (over 2981 pounds), and fast acceleration. Japanese cars have better gas milage, three or four cylinders (and a few six cylinders), and smaller engines. If you click “Europe” in the Label Probability Pane, you can see bars representing evidence for a car being European. For example, five cylinders strongly indicates that a car is European. The height of the corresponding pie, however, shows that there were only three cars with five cylinders in the data. If a car gets good mileage, there is a lot of evidence in favor of it being European. If a car’s mileage is >41, then there is an 83% chance that it’s European. If a car is European, there is only a 10.4% chance that its mileage is better than 41 mpg. But only 2% of Japanese cars—and no U.S. cars—have mpg in this range, so Europe gets the most evidence.

Suppose you wanted to predict where a car came from knowing only that its milage was 40 mpg and it could accelerate to 60 mph in 20 seconds. You can answer this by selecting the appropriate pies (or bars): mpg=30.95-41.15 and time_to_sixty=19.5+. The resulting probability distribution on the right shows a two thirds chance that the car is European and one third chance that it is from the U.S. Why is there no possibility it is Japanese? Because there were no Japanese cars in the training set with time_to_sixty>19.5. If you run the inducer again with the Laplace correction turned on, you get a different answer: 53% chance for European, 26% chance for U.S., and a 21% chance of being Japanese. The reason for this is that Laplace correction prevents any slice in the pies from going completely to zero. Certainly there is no fundamental reason why the Japanese could not make a car that accelerates to 60 mph in >19.5 seconds. Hence, when the probabilities (pies) are multiplied together, the possibility of predicting a Japanese car is not eliminated.

Gender Attribution

The *adult* dataset contains information about working adults. This dataset was extracted from the U.S. Census Bureau. It contains data about people older than 16, with a gross income of more than $100 per year who work at least one hour a week. You might want to know how to characterize males and females. The file */usr/lib/MineSet/eviviz/examples/adult-sex.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/adult.schema*, with the label set to *sex*, after removing the relationship column (which would have made the classifier trivial).

In the Evidence Visualizer, the Label Probability Pane shows that the prior probability of working males is higher than that of females.

- Marital status is the most important predictor of gender. If a worker is a *married-civilian-spouse* there is a greater probability of being male. A worker who is widowed and working, however, is much more likely to be female.
- The second attribute listed shows occupation. Study this to learn which occupations are popular with a particular gender. Male oriented trades are *Craft-repair, Transport-moving, Farming-fishing,* and *Armed-forces*. Female trades are *Private-house-service* and *Adm-clerical*. By clicking on the button next to “Female” in the Label Probability Pane, and then moving the mouse over occupation=*Adm-clerical*, one can see that 23% of females choose *Adm-clerical* as their job. Conversely, given that one’s job is *Adm-clerical*, there is a 67% chance that the gender is Female.

  Suppose you wanted to find out the probability of being female given that a person is *widowed* and has occupation=*Adm-clerical*. This can be done by clicking on the pies or bars representing these values and reading 95% from the test at the top when you move the mouse over the box next to “Female” (in pick mode).
- If the working class is either *self-employed-inc* or *self-employed-not-inc*, the probability that the person is a male is higher. Conversely, if the working class is *state-gov*, the conditional probability that the person is a female is higher, but the posterior probability (after taking into account the prior probability) is not higher (click it and look at the posterior probability on the right). The size of the female slice increased by selecting *state-gov*, but not so much that it would lead you to predict that a person was female, given only that they worked for the state.

  By rotating the view, you can see that most people work in private industry by looking at the height of the pie.
- By looking at the gross-income attribute, you can see that the higher the income range, the higher the probability of being male.
- Education generally does not indicate much about gender, except for doctorate degrees, where you are more likely to find males.
- Different occupations have different distributions for males and females.

- The race attribute shows that African-Americans have a higher percentage of females working than the percentage of other races in the conditional probability. Click the value to see that the posterior is about equal between males and females.
- Males in this dataset work more hours per week than do females.

Salary Factors

If you have a dataset of working adults, you might want to find out what factors affect salary. You might then divide the records into two classes: those making ≤ $50,000 a year and those making more. Each record then has an attribute with two values: "– 50,000" and "50,000+". You can run a MineSet classifier to help determine what factors influence salary. The file */usr/lib/MineSet/eviviz/examples/adult-salary.eviviz* shows the Evidence classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/adult.schema* with gross_income binned at the user-specified threshold of 50000 and the label set to gross_income_bin.

The attributes in the Evidence Visualizer are ranked by importance; thus, relationship, marital status, age, occupation, education, hours per week, and sex are considered most important.

- Relationship shows that husbands and wives are likely to make more money than unmarried workers or workers not in a family. Note also that even though the "Husband" pie shows approximately 3/4 yellow (over $50,000), this indicates only a larger amount of evidence for the class. Since the original population showed that most people are making less than $50,000 per year, clicking the button shows you through the pie on the right that the populations now are approximately equal.
- Marital status shows that most people are married (the second pie chart from the left is tall). Married workers earn more money than unmarried people.
- Age shows that age is a crucial factor. Until the age of 61, when many people retire, the probability of making over $50,000 increases as workers get older.
- Different occupations yield different probabilities. Executive and professional jobs raise the evidence for making over $50,000 per year.

- Education is an important factor. When considering just education, the highest evidence for earning over $50,000 is given to workers whose educational level includes a masters or doctoral degree, or matriculation from professional schools.
- Hours per week show that the more hours worked, the higher the evidence for earning more money.
- Sex shows that being a female gives evidence for making less than $50,000 per year.

Iris Classification

In this dataset, each record describes four characteristics of iris flowers: petal width, petal length, sepal width, and sepal length. Each iris was further classified into the types *iris-setosa*, *iris-versicolor*, or *iris-virginica*. The goal is to understand what characterizes each iris type.

Before running a classifier, click the Column Importance tab in the Tool Manager's Classifiers tab; then click *Go!*. You obtain a ranking of the importance of the features: petal_width, petal_length, and sepal_length. You can map these to the axes in the Scatter Visualizer, with the iris_type mapped to the color and see the clusters.

The file */usr/lib/MineSet/eviviz/examples/iris.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/iris.schema*.

In the Evidence Visualizer, we can see that petal_length and petal_width are excellent discriminatory attributes, while sepal_length and sepal_width are not as good. Move the importance threshold slider to the right to see that the sepal-based attributes disappear first.

Mushroom Classification

The file */usr/lib/MineSet/eviviz/examples/mushroom.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/mushroom.schema*.

The goal is to understand which mushrooms are edible and which ones are poisonous, given this dataset. There are over 8000 records in this set; thus, running this inducer might take several minutes. Note that under the default mode of the one-third holdout for accuracy estimation, a third of the records are kept for testing.

Each mushroom has many characteristics, including cap color, bruises, and odor. The Evidence Visualizer orders attributes by importance (that is, usefulness in predicting the label). Odor and color appear at the top of the list because the distributions in the pies is most different from value to value for these attributes. You can see a characterization of poisonous mushrooms by changing the pointer to an arrow (click the arrow icon at the top right of the main screen), then clicking the button by that class label in the right pane. High bars are associated with values that indicate the mushrooms are poisonous.

In the Evidence Visualizer, move the importance threshold slider to the right. The attributes with the lowest importance are removed from the scene. The most important attribute by far is odor, as its importance is 92; all other attributes have importance less than 48. Almost all values are good discriminators, but if there is no odor (none), then there is a mix of both classes. The Evidence Visualizer lets you see specific values that might be critical, even if the attribute itself is not always important. For example, *stalk_color_below_ring* is not a good discriminatory attribute because most of the time it takes on the value white. White offers no predictive power because there are equal amounts of edible and poisonous mushrooms with this value. When *stalk_color_below_ring* takes the value gray or buff, it provides excellent discrimination, but there are very few mushrooms with these values.

Party Affiliation

This dataset consists of voting records. The goal is to identify the party a congressperson belongs to given data about key votes. The dataset includes votes for each member of the U.S. House of Representatives on the 16 key votes identified by the *Congressional Quarterly Almanac* (*CQA*). The *CQA* lists nine types of votes: voted for, paired for, and announced for (these three are simplified to yes), voted against, paired against, and announced against (these three are simplified to no), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three are simplified to an unknown disposition).

Before running a classifier, look at the 16 votes to see if you can perceive which features are important. Then run the Evidence Visualizer.

The file */usr/lib/MineSet/eviviz/examples/vote.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/vote.schema*.

Breast Cancer Diagnosis

The breast cancer dataset contains information about females undergoing breast cancer diagnosis. Each record represents a patient with attributes such as cell size, clump thickness, and marginal adhesion. The final attribute is whether the diagnosis is malignant or benign. The file */usr/lib/MineSet/eviviz/examples/breast.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/breast.schema*.

In the Evidence Visualizer, you can see that *sample_code_number* was discretized into one range that is equally split, meaning that it does not indicate whether the breast cancer is benign or malignant.

Hypothyroid Diagnosis

The hypothyroid diseases dataset is similar to the one for breast cancer. The file */usr/lib/MineSet/eviviz/examples/hypothyroid.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/hypothyroid.schema*.

There are 3163 records in this dataset and most of them do not have hypothyroid (95.45%). While this means that one can predict “negative” and be correct with high probability, it’s those people that have hypothyroid that we are most worried about. In technical terms, the false negatives are very important.

In the Evidence Visualizer, you can see that *fti* is very important. The first two ranges (besides the unknown) give a lot of evidence for hypothyroid.

Pima Diabetes Diagnosis

This dataset is a diagnosis problem for diabetes using statistics gathered from an Indian tribe in Phoenix Arizona. The task is to determine whether a patient has diabetes, given some medical attributes, such as blood pressure, body mass, glucose level, and age.

The file */usr/lib/MineSet/eviviz/examples/pima.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/pima.schema*.

In the Evidence Visualizer, you can see that many attributes are irrelevant by themselves. As plasma_glucose increases, the probability of having diabetes increases. The number of pregnancies is also a good indicator when it is high (above 6), as is age (above 27).

DNA Boundaries

The file */usr/lib/MineSet/eviviz/examples/dna.eviviz* shows the structure of the Evidence Classifier induced for this problem. This file was generated by running the inducer on */usr/lib/MineSet/data/dna.schema*.

There are 3,186 records in this DNA dataset. The domain is drawn from the field of molecular biology. Splice junctions are points on a DNA sequence at which “superfluous” DNA is removed during protein creation. The task is to recognize exon / intron boundaries, referred to as EI sites; intron / exon boundaries, referred to as IE sites; or neither. The IE borders are referred to as “acceptors” and the EI borders are “donors.” The records were originally taken from GenBank 64.1 (*genbank.bio.net*). The attributes provide a window of 60 nucleotides. The classification is the middle point of the window, thus providing 30 nucleotides at each side of the junction.

From the Evidence Visualizer, you can see that attributes near the center are chosen as very important. Attributes further away from the splice junction are less important.

If you click and select the pie charts in the left pane corresponding to “left_01: G” and “left_02: A”, then the pie chart in the label probability pane on the right will change to show the probability distribution of each class as predicted by the evidence classifier. Given these two values, pie chart shows that the evidence model built assigns the highest probability to “intron / exon”, followed by “exon / intron” and “none”.

The accuracy improves slightly if you invoke automatic feature selection, although running time increases dramatically (sometimes hours). In such cases, run feature selection once, and continue mining only with the chosen features.

Chapter 11

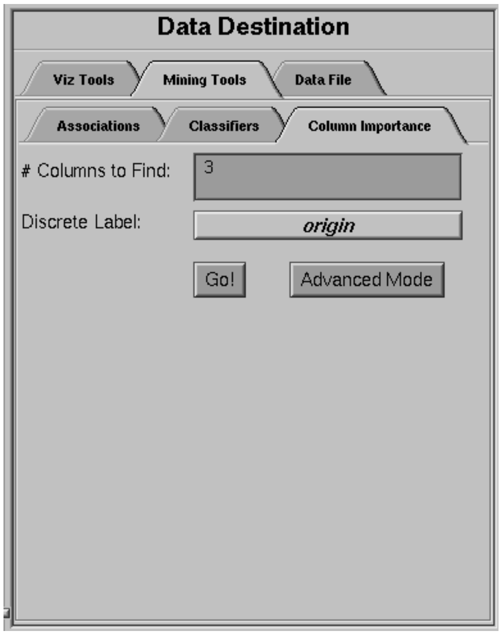
Column Importance

This chapter discusses the features and capabilities of the Column Importance mining tool, and the relationship between column importance and the importance ranking in the other data mining tools. Because of the differences in representation for classification models, different attributes may be judged more important for different models.

Note: This chapter assumes that you have read Chapter 8, “MineSet Inducers and Classifiers.”

Finding Important Columns

Column Importance (Figure 11-1) determines how important various columns are in discriminating the different values of the label column you choose. You might, for example, want to find out the best three columns for discriminating the label *good credit risk* so you can choose them for the Scatter Visualizer. When you select the label and click *Go!*, a popup window appears with the three columns that are the best three discriminators. A measure called “purity” (a number from 0 to 100) informs you how well the columns discriminate the different labels. Adding more columns can only increase the purity.



Columns to Find: 3

Discrete Label: origin

Figure 11-1 The Column Importance Tab

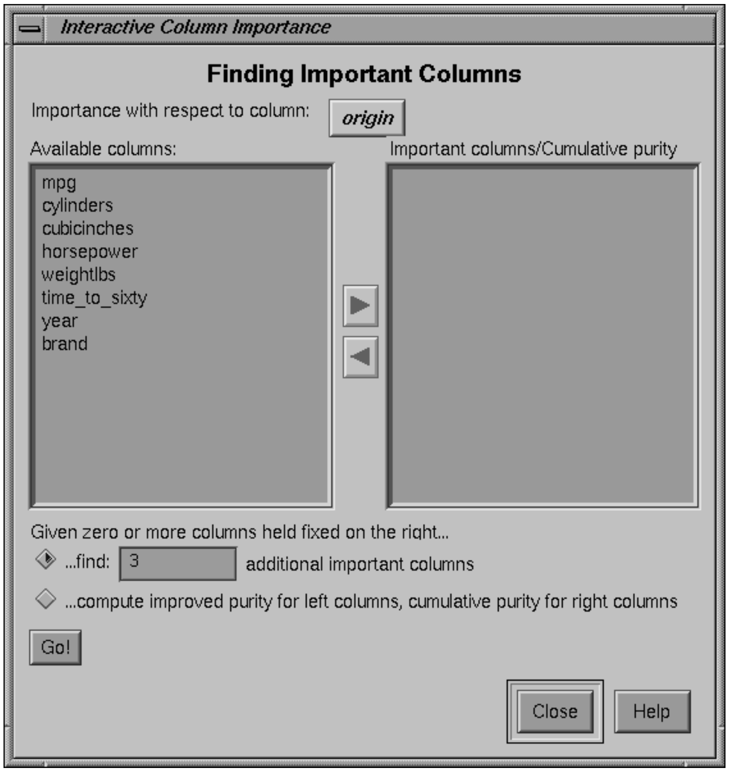
There are two modes of column importance:

- Simple Mode

  To invoke the Simple mode, choose a discrete label from the popup menu, and specify the number of columns you want to see, then click *Go!*.

- Advanced Mode

  Advanced mode lets you control the choice of columns. To enter Advanced mode, click *Advanced Mode* in the Column Importance panel. A dialog box appears, as shown in Figure 11-2. The dialog box contains two lists of column names: The left list contains the available attributes and the right list contains attributes chosen as important (by either the user or the column importance algorithm).



Interactive Column Importance

Finding Important Columns

Importance with respect to column: origin

Available columns:

mpg
cylinders
cubicinches
horsepower
weightlbs
time_to_sixty
year
brand

Important columns/Cumulative purity

Given zero or more columns held fixed on the right...

◆ ...find: 3 additional important columns

◇ ...compute improved purity for left columns, cumulative purity for right columns

Figure 11-2 Advanced Mode of Column Importance

Advanced mode can work two different ways: finding several new important attributes or ranking available attributes.

- Finding Several Important Attributes

  To enter this submode, click the first of the two radio buttons at the bottom of the dialog (*...find [number] additional important attributes*). If you click *Go!* with no further changes, the effect is the same as if you were in Simple mode, finding the specified number of important columns and automatically moving them to the right column. Near each column, the cumulative purity is given (that is, the purity of all the columns up to and including the one on the line. More attributes can only increase the purity.

  Alternatively, by moving column names from the left list to the right list, you can prespecify columns that you want included and let the system add more. For example, to select the *cylinders* column and let the system find three more columns, click the *cylinders* column name, then click the right arrow between the lists.

  Clicking *Go!* lets you see the cumulative purity of each column, together with the previous ones in the list. A purity of 100 means that using the given columns, you can perfectly discriminate the different label values in the dataset.

- Ranking Available Attributes

  Advanced mode also lets you compute the change in purity that each column would add to all those that were already marked important, that is, they are in the list on the right. For example, you might move *cylinders* to the list on the right, and then ask the system to compute the incremental improvement in purity that each column remaining in the left column would yield. The cumulative purity is computed for columns on the right.

  To enter this submode, click the second of the two radio buttons at the bottom of the dialog (*...compute improved purity for attributes on the left.*). This submode permits fine control over the process. If two columns are ranked very closely, you might prefer one over the other (for example, because it is cheaper to gather, more reliable, or easier to understand).

Column Importance Notes

Note that with other columns, the importance of features varies from their ranking alone. For example, while *net-income* might be a good column individually, it might not be as important together with *salary* because they are likely to be highly correlated. The best set of three columns is not necessarily composed of the columns that rank highest individually. If two columns give the income in dollars and in another currency, they are ranked equally alone; however, once one of them is chosen, the other adds no discriminatory power to the set of best features.

Column selection is useful for finding the best three axes for the Scatter Visualizer, as well as for finding a good discriminatory hierarchy (hierarchy that separates different label values) for the Tree Visualizer when you select the label to be the key used in the Tree Visualizer.

All floating point values (**double**s or **float**s) are prediscretized using the automatic discretization (see Chapter 3, “The Tool Manager”). If a column has no value given to it in the left list, the algorithm did not consider it; this is because it either had a single value (for example, when it is discretized into one interval), or the number of records that it would separate are not statistically significant.

Column Importance and Relation to Classifiers

This section describes the differences among Column Importance, the importance ranking chosen by the Evidence Inducer, and the splits chosen by the Decision Tree Inducer. As Column Importance uses all of the data, these descriptions assume that you are running the inducers in “Classifier Only” mode, so that the inducers are using all of the data as well.

The Discretization Process

The column importance algorithm and the Evidence Inducer discretize all continuous attributes using the automatic discretization algorithm (the same algorithm that is applied in automatic binning in the Tool Manager). The decision tree algorithm does not pre-discretize attributes (columns) and finds thresholds as the tree is built.

The main advantage of the automatic discretization is that it discretizes the continuous range into several intervals at once, while the decision tree makes only binary splits.

The main advantage of the decision tree algorithm is that it discretizes subsets of the data (those that reach a specific node where a test is done). Thus the discretization is “local” to those records as opposed to a “global” discretization.

The Importance Function

The Evidence Inducer and Column Importance rank attributes based on “mutual information” as the purity measure. The Decision Tree Inducer defaults to “normalized mutual information,” which penalizes multi-way splits (see the description of splitting criterion in “Decision Tree Inducer Options” on page 257). Thus, the Decision Tree Inducer prefers an attribute with few values over attributes with many values. The default for decision trees can be changed to “mutual information.”

Dependence on Other Attributes

The Evidence Inducer ranks each attribute independently. If several attributes are highly correlated, they have similar ranking. If you use the Advanced mode from Column Importance, the “...compute improved purity” option without any attributes chosen as important (that is, moved to the list on the right), the attribute ranking shown matches the sort order chosen by the Evidence Inducer.

Column Importance and the Decision Tree Inducer both provide more powerful importance capability than the Evidence Inducer. Both choose an importance ranking with respect to other attributes. In Column Importance, attributes are judged as important relative to the set of attributes in the list on the right. If two attributes are highly correlated and one is chosen, the other does not rank very highly. Similarly, in a decision tree, important attributes are chosen with respect to attributes on the path to the root node.

Decision trees provide the most flexible importance ranking because different attributes can be chosen at different subtrees. For example, one attribute can be chosen for the left child of the root and another for the right child of the root. With column importance, a single attribute must be chosen for all combinations of the previously chosen attributes. For some visualizations, such as the Scatter Visualizer, the Column Importance feature is most appropriate because you cannot represent a tree structure on the axes, yet you want to find a set of attributes that *together* give the most information.

Appendix A

Creating Data and Configuration Files for the Tree Visualizer

The first part of this appendix describes the types and formats of data supported by the Tree Visualizer. Data input to the Tree Visualizer must be provided as a single file containing raw data, usually in a tab-separated ASCII text form.

The second part discusses the configuration file, which describes how the Tree Visualizer reads in, and graphically displays, the data file.

Note that both the data and configuration files can be generated automatically by the Tool Manager (see Chapter 3).

Note: Read Chapter 4, “Using the Tree Visualizer,” before using this appendix.

The Data File

In its simplest form, the data file consists of a list of lines, each containing a set of fields separated by one tab. (Other separators are also allowed—see “Input Options” on page 340—but only one can separate each field). All lines must contain the same fields. (The interpretation of the fields is specified by the configuration file, described in the next section.) For example, using the retail store data (*store.treeviz* file) provided as part of the Tree Visualizer package, the first few lines of the input file look like this:

```
Eastern  Maryland  Baltimore  1816  appliances  72  115  138
Eastern  Maryland  Baltimore  1816  clothing  355  344  395
Eastern  Maryland  Baltimore  1816  electronics  156  182  209
Eastern  Maryland  Baltimore  1816  furniture  78  75  82
Eastern  Massachusetts  Boston  1331  appliances  48  68  81
Eastern  Massachusetts  Boston  1331  clothing  307  258  296
Eastern  Massachusetts  Boston  1331  electronics  38  183  210
Eastern  Massachusetts  Boston  1331  furniture  52  69  75
Eastern  Massachusetts  Boston  1220  appliances  37  63  75
Eastern  Massachusetts  Boston  1220  clothing  233  240  276
Eastern  Massachusetts  Boston  1220  electronics  175  208  239
Eastern  Massachusetts  Boston  1220  furniture  35  53  58
```

In this example, the first five columns are strings: region, state, city, store ID, and product. These are followed by three numbers, representing current sales, last year's sales, and the sales target. (The specific mapping to those values is defined in the configuration file, described in the section "The Configuration File.")

The data file cannot contain blank lines or comments. Missing or extra data on a line causes an error.

Note: One tab (the default separator) separates each field. Do not insert multiple tabs to line up the fields visually; doing so generates blank fields. It is possible to use other characters, such as a colon (:), as a separator. In this case, the first line appears as:

```
Eastern:Maryland:Baltimore:1816:appliances:72:115:138
```

The order of the columns must match the format of the configuration file. The order of the rows affects the layout of the final graphic, unless the configuration file specifies sorting. Generally, objects appearing earlier in the file appear to the left of objects appearing later in the file.

Any field in the data can also be a "?", indicating that the data is null (unknown). See Appendix G, "Nulls in MineSet."

Data Types

The Tree Visualizer supports integer, floating-point number, and string data types, as well as arrays of these types. The following five data types are supported:

- **int** represents a 32-bit signed integer.
- **float** represents a single-precision floating point number. The decimal point is optional. Numbers in exponential “e” notation are also accepted.
- **double** represents a double-precision floating point number. The decimal point is optional when representing a floating point number. Numbers in exponential “e” notation are also accepted. The superior precision of **double** can be useful for accurately representing large numbers, since **float** can represent only seven or eight significant digits accurately. This superior accuracy, however, consumes twice the memory space of **float**.
- **dataString** represents a string that is unlikely to appear multiple times. If it appears multiple times, several copies are made. A **dataString** can be used to store an address. Addresses are unlikely to be compared, and each record can have a different address.
- **string** represents a string of characters that can appear multiple times in the data file. Unlike a **dataString**, only a single copy of a given string is stored in memory, no matter how many times it appears in the data. This saves memory for strings appearing many times.

  Comparing **strings** is also much quicker than comparing **dataString**s. Reading in **strings** can be slower than reading in **dataStrings** because it is necessary to look for duplications. An example of **string** use is a division name that appears once for each department in the division. If you are unsure whether to use a **string** or a **dataString**, use a **string**.

Enumerations

You can create a special data type that maps consecutive integers to strings. These types are referred to as enumerations, or enums. For example, you can create a state enum that maps 0 to Alabama, 1 to Alaska, and so on. Often, enums are created by the Tool Manager to represent bins. For example, 0 might map to a bin representing <10, 1 to the bin 10-20, and so on.

Arrays

With the Tree Visualizer, you can use one-dimensional arrays of fixed or variable size.

In a fixed-sized array, all entries of the given type have the same number of values. For example, the budgets of the 50 United States, can be represented by a separate float column for each state, or by a single array with 50 floats.

A special form of a fixed array is an “enumerated array.” Like the normal fixed array, there are a fixed number of values in the array; however, the values are associated with an enumeration. For each value in the enumeration, there is a single entry in the array. For example, if there is an enumeration representing the 50 states, an enumerated array based on this enumeration has 50 values. (Note that in MineSet release 1.0, the enumerated array was referred to as a “keyed fixed array.”)

A variant of the “enumerated array” is the “null enumerated array.” This is a variant of the enumerated array with an additional entry at the beginning for null (represented as a “?”). For example, with the enumeration of the 50 states, the null enumerated array has 51 values, one for NULL, and the remaining 50 for the 50 states. The null array element could be used for entries where the state is unknown.

A variable-length array can have a different number of entries in each instance of the array. Often this is useful for representing organizations in which different parts have different depths. For example, one department could be represented by Gomez:Shapiro:Lacy (three entries), while another is Gomez:Wong:McMartin:Singe (four entries).

A variable-length entry with zero values can also be declared by passing an empty string. This can be used to specify data for the top level of a hierarchy.

When representing an organization with variable-length arrays, be careful. The Tree Visualizer computes the height for each level of the hierarchy separately, giving the highest bar on each level a user-specified height and normalizing the other bars accordingly. For example: Imagine a U.S.-based organization with a domestic and an international sales force. Domestic sales are divided up into states, which are divided into cities. International sales are divided into continents, which are divided into countries and cities. You can have locations such as domestic:California:Mountain View, and international:Europe:Italy:Rome. When displaying organizational hierarchies of this type, it is best to normalize heights at each level. If this is not done, small parts of the organization (for example, Mountain View) would be dwarfed by large parts of the organization (for example, domestic).

When the system tries to match up the levels, the normalization process might introduce anomalies. Usually, this is not the case at the highest level (domestic is matched with international); however, at lower levels this correspondence is no longer valid. Domestic cities (for example, Mountain View) are at the third level, but the third level for international is a country (for example, Italy). Comparing domestic cities against foreign countries usually has little validity. In this case, it is recommended that you introduce artificial levels to balance the hierarchies (for example, domestic:USA:California:Mountain View), thus matching cities.

Variable-length arrays might also be useful when some of the regions being compared are subdivided further than others. For example, an organization might have USA:California:San Francisco and USA:California:Los Angeles, but only USA:Wyoming. There is no need to construct an artificial third level just to keep the arrays balanced, as long as each level in the array matches the same level in other arrays.

Starting up the Tree Visualizer takes longer when variable-length arrays are read in than when fixed-length arrays or individual columns are read in. Unless the data is variable length, it is best not to use variable-length arrays.

As with the columns, arrays are represented as values separated by tabs or other separators. For a fixed-sized array, the same separator can be used for columns and for individual array elements (in which case, array elements are not visually distinguished from separate columns). You can also define a different separator. In the sales example (on page 325), for example, you can treat the location as a four-element array, rather than as four columns. It then could be represented like this:

```
Eastern:Maryland:Baltimore:1816     appliances    72    115    138
```

Here, the array is separated by colons, and the columns are separated by tabs. (For clarity, the rest of this document uses tabs to separate columns, and colons to separate array elements.)

For a variable-length array, you must use different separators for the array and for the columns; otherwise, it is impossible to determine where the variable-length array ends and the other columns begin.

The Configuration File

The configuration file format is flexible. Words in it must be separated by spaces, and it is case-sensitive. Except for the include statement and text within quoted strings, spacing and line breaks are irrelevant.

Comments begin with a pound (#) symbol at the beginning of a line; anything after this symbol to the end of the line (80 characters) is ignored.

Sections

The configuration file consists of a series of sections, each of which has this syntax:

```
sectionKeyword
{
    statements...
}
```

where *sectionKeyword* names the section. A semicolon (;) can follow the closing brace (}) but is not required. The order of the sections is significant, since sections can refer to variables defined in previous sections.

Options Files

As each section is encountered, a special configuration file (referred to as a "options file") is also read in. Options files have names in the form:

sectionName`.treeviz.options`

Options files normally contain options statements. These files are searched in the following order:

1. The directory */usr/lib/MineSet/treeviz*. This directory usually contains system defaults.
2. The *~/.MineSet* directory (where the tilde, ~, indicates your home directory). You can set up personal defaults in this directory.
3. The current directory. This lets you set up defaults for each directory.

Files with the same name can appear in more than one of the above-named directories; in this case, the order given is the one in which the directories are read. If the same option is found in multiple files, the last option read is used. Note that the appropriate section in the configuration file is read after all the options files have been read in; thus, options in the configuration file override those in the options files.

Statements

A statement has the following syntax:

statementKeyword `info ;`

where *statementKeyword* defines the statement, and *info* varies according to the keyword. A statement can be another section (using the brace format defined under "Sections").

Variable Names

A variable name can appear in two formats:

- In the first format, it is a letter followed by a number of letters, digits, or underscores. It cannot be a keyword, and should not be placed in quotation marks.
- In the alternate form, the variable name should be surrounded by back quotes (`‘`). In this form, the variable name can match a keyword, and can contain even non-alphanumeric characters. The primary purpose of this second form is for configuration files generated automatically by the Tool Manager.

There is no scoping of variable names; a given variable name can be declared only once in the configuration file.

Option Statements

Many sections have options statements, which have this syntax:

```
options key info, key info... ;
```

where *key* defines the specific option, and *info* depends on the key. In some cases, the *key* can be more than one word. To maximize the number of allowable variable names, most option keys are meaningful only within the appropriate option statement; keys do not conflict with variable names. You can declare several options on the same line, separating them by commas or placing them in several options statements. For example, the following two examples are equivalent:

```
options home angle 30, shrinkage 10.0;
```

and

```
options home angle 30;
options shrinkage 10.0;
```

If two conflicting values for the same option appear, the last value is taken.

Include Statements

The configuration file can contain lines of the form:

```
include "filename"
```

These lines can appear anywhere in the configuration file, but each must be on its own line. The filename must be in quotation marks; anything after the closing quotation mark is ignored. Include statements can be nested. If a relative pathname (one not beginning with a slash) is specified, the file is first sought relative to the directory containing the current configuration file. (If include statements are present, this might not be the same as the initially loaded configuration file.) If it is not found in the current configuration file, the include is sought in the current directory. If the file is not found, an error message appears.

Sinclude Statements

A statement similar to an include is sinclude, which has the syntax:

```
sinclude "filename"
```

This is identical to the include statement, except that no error appears if the file does not exist; instead, the sinclude statement is ignored.

Strings and Characters

Strings and characters in the configuration file follow C conventions. Strings are in double quotes ("), and characters are in single quotation marks (‘). All standard backslash conventions are followed (for example, \n represents a new line).

Keywords

The currently recognized keywords are listed in Table A-1. Variables cannot have these names unless they are surrounded by back quotes (‘). Tokens appearing only in option statements are not keywords, and can be used for variable names.

Table A-1 Keywords for the Tree Visualizer

aggregate	disk	int	normalize
any	divide	isSummary	off
ascending	double	key	on
average	enum	label	options
back	execute	landscape	scale
base	expressions	legend	separator
buckets	file	levels	sort
color	filter	max	string

Table A-1 (continued) Keywords for the Tree Visualizer

colors	float	message	sum
count	height	min	view
dataString	hierarchy	modulus	
descending	input	none	

Expressions

Expressions are accepted in several places in the input. Expressions follow standard C syntax. The following operations are supported:

```
+  -  *  /  %  ==  !=  >  <  >=  <=  &&  ||  !  &  |  ^  ?:
```

Also, the following functions are available:

- **divide**(x, y, z) divides *x* by *y*, unless *y* is zero. If *y* is zero, the result is *z*; this is equivalent to the C construct *y==0 ? z : x/y.*
- **modulus**(x, y, z) is similar to **divide**, but for modulus.
- **hierarchy**(*string*) is valid only within a hierarchy. It produces a string describing the components of the hierarchy, separated by *string*. For example:

  ```
  hierarchy(":")
  ```

  might produce

  ```
  Western:California:Mountain View
  ```

  The hierarchy function is most useful in the execute statement, passing the hierarchy information to the command being executed.
- **isSummary**() returns 1 if the expression is being applied to base information; otherwise, it returns zero. Often, this is useful with the **?:** operator, particularly in message and execute statements.

Type handling is similar to that in C. Expressions using **int** and **float** promote both sides to float. Expressions using **int** and **double**, or **float** and **double** promote both sides to double. The result of a relational expression (for example, ==, <) is always an **int**. Type casting is also supported.

Unlike in C, strings can be compared using relational expressions; the strings are compared alphabetically.

The Input Section

The first section of a data file is normally the input section. It defines the name and format of the file. A typical input section might look like this:

```
input {
   file "store";

   string region;
   string state;
   string city;
   string storeId;
   string product;
   float sales;
   float lastYear;
   float target;

   options separator ‘:’;
}
```

This example states that the input file is called *store*, and that there are eight fields: five of type **string**, three of type **float**.

When the input section is entered, the options file, *input.treeviz.options*, is read in.

File Statements

The file statement names the data file to be read. This statement is required. Its syntax is:

```
file "filename";
```

Filename must be in double quote marks. If it is a relative pathname (no leading slash), it is first sought in the directory containing the current configuration file. If include statements are present, this might not be the same as the initially loaded configuration file. If it is not found in the current configuration file's directory, the file is sought in the current directory.

Data Statements

The data statements declare the columns in the data file. The columns must be declared in the order they appear in the data file. The format of most data statements is

```
type name;
```

where *type* is **int**, **float**, **double string**, or **dataString**; *name* is the variable name. Unlike in C, only one variable can be declared per statement.

Enumerations

The syntax for declaring an enumeration is:

```
enum type name { value, value...};
```

For example:

```
enum string state {
   "Alabama",
   "Alaska",
   ...
   “Wyoming"
};
```

The word “string” indicates that the enumeration maps integers to strings; they can also be mapped to other types.

Note that for compatibility with MineSet 1.0, “enum” can be replaced with “key.”

Once an enumeration is declared, a column can be declared to be of that enumeration using the following syntax:

```
enum enumname columnname;
```

For example:

```
enum state st;
```

declares `st` to be a variable of state enumeration. The input file corresponding to this column must contain values from 0-49 (or “?” representing null); however, the output shows the state name.

Enumerations also can be used to declare enumerated arrays (see “Enumerated Arrays”).

Fixed Arrays

Arrays are also declared using data declarations. The simplest form is the fixed array. The declaration syntax is

```
type name [ number ] ;
```

For example:

```
float revenue [50];
```

You can also override the separator by declaring it as

```
type name [ number ] separator ‘char’;
```

For example:

```
float revenue [50] separator ‘:’;
```

If no separator is specified, the default column separator (usually a tab) is used.

Enumerated Arrays

To declare an enumerated array, first declare the enumeration (see the “Enumerations” subsection). Then declare the array using the following syntax:

```
type name [ enum keyname ];
```

or

```
type name [enum keyname ] separator ‘char’;
```

For example:

```
float revenue [enum state];
```

As with the normal fixed array, you can also specify a separator. Note that for compatibility with MineSet 1.0, the word "enum" can be omitted from within the brackets. To declare a null enumerated array, use the syntax:

```
type name [ null enum keyname ];
```

or

```
type name [ null enum keyname ] separator `char';
```

For example:

```
float revenue [null enum state];
```

indicates that the array contains one additional value at the beginning, corresponding to null.

Variable-Length Arrays

To declare a variable-length array, do not include a number in the brackets ([ ]). With a variable-length array, you must include a separator that is different from the one specified as a column separator. The syntax is:

```
type name [] separator ‘char’;
```

For example:

```
string category [] separator ‘:’;
```

Input Options

The input section of a data file has several options. All options statements begin with the word *options* and have one or more comma-separated options.

- The separator option defines the separator between columns in the data file. The default separator is a tab. The syntax is:

  ```
  options separator ‘char’;
  ```

  For example:

  ```
  options separator ‘:’;
  ```

  Note: Arrays can override the separator.

- The monitor option allows a dynamic update of the data displayed. When the specified file is changed (for example, through the *touch* command), the data file (not the configuration file) is reread. The data file should not be used to trigger the updates. This prevents the data file being read at the same time it is being updated. The syntax of the monitor option is

  ```
  options monitor "filename";
  options monitor "filename" timeout;
  ```

  where *filename* is the file to watch, and the optional *timeout* specifies the number of seconds to wait after the file changes. If the user interacts with the application in any way during this timeout (via the mouse or keyboard), the timeout restarts. Updating the file can take a few seconds. By specifying a timeout, the chances of an update occurring while the user is interacting with the tool are minimized (but the update is delayed). If no timeout is specified, the update occurs immediately.

  The file being monitored must exist at the start of the program. When this file is being updated, it must not be removed and re-created; instead, only its modify time should be updated (for example, through the *touch* command). If the file is deleted, subsequent updates are not shown.

Suppose a program extractor extracts data from a database into a data file. If you want the program to update the data file every 10 minutes, the script might look like this:

```
extractor > dataFile;        # create first data file
touch trigger;               # create the trigger file
while (sleep 600)            # sleep 10 minutes
do
   extractor > dataFile;     # create new data file
   touch trigger;            # force a reread
done &                       # this loop goes in the
                             # background
treeviz configFile;          # run treeviz
kill $!                      # when treeviz exits, kill
the
                             # update loop
```

The monitor option can be used only if the file alteration monitor */usr/etc/fam* is installed (this can be found in the subsystem *desktop_eoe.sw.fam*).

The input section of a configuration file might look like this:

```
input
{
   file "dataFile"
   #data declarations here
   options monitor "trigger" 15;
}
```

- The backslash option controls whether backslashes in the input data are treated specially or like other characters. The syntax is:

  ```
   options backslash off;
   options backslash on;
  ```

  The default is off. If backslash processing is on, separators in the input data preceded by backslashes are treated as regular characters rather than separators. Also, within strings standard C-style backslash processing is done.

The Expression Section

The expression section of a data file lets you define additional columns that are expressions of existing columns. For example, one column can be defined as the sum of two other columns. The expressions are calculated before the definition of the hierarchy. In many cases, it is more appropriate to apply the expressions after creating the hierarchy; the expressions then should be defined within the hierarchy section (described later), and the expressions section can be omitted.

The following is a sample expression section. This section assumes two existing columns of type **float**, "male" and "female"; these represent spending by males and females on various goods. Two columns are added: "total" represents the total dollars spent, and "pctFemale" represents the percentage of dollars spent by females.

```
expressions
{
float total = male+female;
float pctFemale = divide (female*100, total, 50.0);
}
```

Note: The pctFemale calculation uses "total," defined in the previous statement. Also, note the use of the divide function rather than the / operator. This results in 50% for the case where there are no dollars spent at all; using the / operator generates a divide by zero error in such a case.

The format of the expressions section is:

```
expressions
{
   expressionDeclaration;
   ...
}
```

where *expressionDeclaration* has the following syntax:

```
type name = expression ;
```

Since the expressions section has no options, no options file is read in for it.

The Hierarchy Section

The hierarchy section of a data file describes how the previously read table is converted into a hierarchy. Here is a sample hierarchy section:

```
hierarchy
{
   levels region, state, city, storeId;
   key product;
   aggregate
   {
      sum sales;
      sum lastYear;
      sum target;
   }
   expressions
   {
      float pctLastYear = divide(sales*100, lastYear, 100.0);
      float pctTarget = divide(sales*100, target, 100.0);
   }
}
```

The parts of the hierarchy section are described below.

When entering the hierarchy section, the *hierarchy.treeviz.options* options file is read in.

Levels Statements

The levels statement defines how the table is converted into a hierarchy. The format is:

```
levels name, name...;
```

where *name* represents a column previously defined in the input or the expressions section. How the hierarchy is created depends on the types of the columns defined.

If the columns represent simple types (for example, strings or numbers), each column is converted into a single level of the hierarchy. The top level of the hierarchy is a single, all-inclusive node. The next level contains one node for each unique value in the first column. The third level contains one node for each unique value in the second column, and so on. Hierarchies created in this way are always balanced: All branches in the hierarchy go to the same depth (namely one greater than the number of columns specified in the levels statement).

In the case where the column is an array, there can be only a single column specified in the levels statement. Each value in the array is mapped to one level in the hierarchy. The top level is a single node representing the total aggregation. The next level contains one node for each unique value of the first value in the array; the third level contains one node for each unique value of the first two values of the array, and so on.

If the array is of fixed type, this hierarchy is balanced. If a variable array is used, the hierarchy is not necessarily balanced (some branches can go deeper than others).

A variable-length array can be used to specify the hierarchy, even if the hierarchy is balanced to a fixed depth. When using columns or fixed arrays to specify the levels, you can specify data associated only with those levels at the bottom (or leaf) nodes. In this case, all higher nodes in the hierarchy must be aggregated. However, rather than relying on automatic aggregation, you might want to supply your own data for each level of the hierarchy (if, for example, the calculation can not be done automatically by the Tree Visualizer). In that case, use variable-length arrays to specify levels and provide separate data for each level. For example, the data file might contain lines such as:

```
Domestic:Western  43
Domestic:Eastern  57
Domestic    85
Intl:Europe  52
Intl:Asia  39
Intl   94
      133
```

Note: The last line has an empty value for the location; the number 133 is translated to the top of the hierarchy.

Key Statements

The key statement specifies those keys that are used to select the bars at each node in the hierarchy. The key corresponds to the bars displayed in the final view. The syntax of the key statement is:

```
key name [sort [ascending|descending]};
```

where *name* is the name of one of the previously defined columns. It cannot be the name of a column used in the levels statement. Only a single key statement can be made.

By default, the bars generated by the key statement appear in the order first encountered. If the key is an enumerated array, the bars appear in the order of the enumeration; otherwise they appear in the order in which values are first encountered in the data file. Adding the word `sort` at the end of the key statement sorts the bars. Sorting depends on the type: Strings are sorted alphabetically, and numbers are sorted numerically. Enumerations are sorted on the index of the enumeration, not the string that the enumeration refers to. If, however, the key is an enumerated array, the sorting takes place according to the enumeration string (to sort based on the enumeration index, leave it unsorted). Optionally, the word `sort` can be followed by `ascending` or `descending` to specify the sort order; the default is ascending.

If the key column is a simple type (for example, a **string**), the unique values of that key are looked up in the original table. The order of the values is the same as the one in which the key values appear in the original input table. Although it is not required, the same keys are often repeated in the same order. For example, in the following table, the fifth column is the key, and has the values “appliances,” “clothing,” “electronics,” and “furniture.”

```
Eastern  Maryland  Baltimore  1816 appliances  72  115  138
Eastern  Maryland  Baltimore  1816 clothing  355  344  395
Eastern  Maryland  Baltimore  1816 electronics  156  182  209
Eastern  Maryland  Baltimore  1816 furniture  78  75  82
Eastern  Massachusetts  Boston  1331 appliances  48  68  81
Eastern  Massachusetts  Boston  1331 clothing  307  258  296
Eastern  Massachusetts  Boston  1331 electronics  38  183  210
Eastern  Massachusetts  Boston  1331 furniture  52  69  75
Eastern  Massachusetts  Boston  1220 appliances  37  63  75
Eastern  Massachusetts  Boston  1220 clothing  233  240  276
Eastern  Massachusetts  Boston  1220 electronics  175  208  239
Eastern  Massachusetts  Boston  1220 furniture  35  53  58
```

The key can also be any column of the enumerated array type. In this case, the enumeration is used as the key for specifying the bars. Other columns in the input can also be enumerated array types, as long as they use the same enumeration. For example, this table can also be input as

```
Eastern  Maryland  Baltimore  1816
              72:355:156:78  115:344:182:75  138:395:209:82
Eastern  Massachusetts  Boston  1331
              48:307:38:52  68:258:183:69  81:296:210:75
Eastern  Massachusetts  Boston  1200
              837:233:175:35  63:240:208:53  75:276:239:58
```

For clarity, each line has been wrapped onto two lines; however, in the file these should be on single lines. The input section for this data appears as

```
input
{
   file "...";
   key string product {
   "appliances", "clothing", "electronics", "furniture"
   }
   string region;
   string state;
   string city;
   string storeId;
   float sales [ enum product ] separator ‘:’ ;
   float lastYear [ enum product ] separator ‘:’ ;
   float target [ enum product ] separator ‘:’ ;
}
```

Note: Since the arrays are fixed, the use of a colon separator for the arrays is not required; however, it might make it easier for a human to read the input.

In this example, the hierarchy section appears as follows:

```
hierarchy
{
   levels region, state, city, storeId;
   key sales;
   ...
}
```

Since *sales* is an enumerated array, it used its key type (product) as the key to generating the bars; thus, each graph in the final view has four bars. Note that *lastYear* and *target* must use the same key type for their array.

Arrays other than enumerated arrays can not be specified as the key.

Aggregate Subsection

The aggregate subsection of the hierarchy section describes how values are aggregated at higher levels of the hierarchy. An example is:

```
aggregate
{
   sum sales;
   sum lastYear;
   sum target;
}
```

This indicates that *sales*, *lastYear*, and *target* are to be summed at higher levels of the hierarchy (each level summing the values in the level below it). In addition to the **sum** aggregation, the aggregations **average**, **min**, **max**, **count**, and **any** are allowed. All are self-explanatory, except for **any**, which indicates that any of the values can be used. This aggregation is used if you expect the same value (for example, a string) to appear everywhere in the hierarchy and if you just want it to populate the entire hierarchy.

A special case is when the key is an enumerated array. Here, the key is normally also aggregated.

In the case where a variable-length array specifies data for all levels of the hierarchy simultaneously (as opposed to merely specifying the data at the leaf nodes), the aggregate section cannot be used.

The two forms an aggregate statement can take are

```
agg name;
name1 = agg name2;
```

In both cases, the aggregate (*agg*) is one of **sum**, **average**, **min**, **max**, **count**, and **any**. The first form was illustrated above; it aggregates a column, and the result is given the same name as the original column being aggregated. The second form aggregates the column *name2*, but gives the result the name *name1*. This second form is useful if the same value is being aggregated multiple times. Since using the first form creates two aggregations with the same name, the second form can be used to differentiate the aggregations.

For example, if you have a column named *expenses* and want to aggregate it to show the maximum and minimum expenses, you can use

```
aggregate
{
   maxExpenses = max expenses;
   minExpenses = min expenses;
}
```

Aggregate Base Subsection

This subsection specifies how values in the base are aggregated. It can be used only if the aggregate subsection is not present. (If the aggregate section is present, the base is aggregated using the aggregations specified in it).

A sample aggregate base subsection is:

```
aggregate base
{
    sum sales;
    sum lastYear;
}
```

An aggregate statement takes the form

```
agg name;
```

where the *aggregate* (*agg*) is one of sum, average, min, max, count, and any, (similar to the aggregate section). The aggregation is applied to all the bars on that base to give the appropriate value for the base. After the base is aggregated, its values correspond to all of the columns used in specifying the bars. Any column not specified in the aggregate base section has a value of zero. Because the base values correspond to the bar values, the second form of the aggregate statement (using the =), cannot be used in the aggregate base section.

Expressions Subsection

An expressions subsection of the hierarchy section is similar to the expressions section described earlier, except that it is applied after the hierarchy is created and aggregated. The syntax is identical, but it is declared within the hierarchy section, not external to it.

To give an example of the difference between calculating the expressions before and after creating the hierarchy, take the example of male and female dollars spent. Assume you want to calculate the percentage of dollars spent by women. The expressions might be:

```
expressions
{
   float total = male+female;
   float pctFemale = divide (female*100, total, 50.0);
}
```

Assume you calculated these variables before creating the hierarchy. Then, when aggregating the data up the hierarchy, summing the percentages is not useful. Averaging the percentages results in a believable number; however, it averages percentages of large dollars with percentages of small dollars, and produces incorrect results. (To make this clearer, suppose that on one product, males spent $99, and females spent $0. On another product, males spent $0, and females spent $1. On the first product females spent 0%, and on the second they spent 100%. Averaging these gives 50%, but in reality, females spent only 1% of the dollars spent on the two products combined.)

The base data should be aggregated first, then the expressions should be applied. (In the example, after aggregating, the result is a combined spending of $99 for males, and $1 for females; if the percentage is calculated after the aggregation, the correct value of 1% results.)

Sort Statements

By default, the order of the nodes within each level of the hierarchy is based on the order of the data in the input file. However, sometimes it is desirable to sort the hierarchy. The sort statement can appear in one of two forms:

```
sort name [, ascending|descending];
sort key [, ascending|descending];
```

In the first form, one column name (not used in the level statement) is used for sorting. The column can be the result of an aggregation or an expression. In the second form, the value used in the level statement is the one used in laying out the hierarchy.

The hierarchy can be sorted in ascending or descending order. If neither option is specified, the default is descending order if the first form of the sort is used (this places the largest columns on the left); the default is ascending order if the second form is used (this typically sorts alphabetically).

Note that sort statements affect the sorting of only the branches of the hierarchy; they do not affect the bars within each node of the hierarchy.

Hierarchy Options

There are two options in the hierarchy section: *skipMissing* and *organization*. The format for the *skipMissing* option is

```
options skipMissing;
```

If this option is off (the default) and some values of the key are not present for a given hierarchy node, dummy entries are created with values of 0. This guarantees that all graphs in the hierarchy have the same number of bars, and the same layout. If this option is on, no such entries are generated. This results in variable-length tables in the hierarchy, and bars exist only for items in the input. The position of these bars, however, is not meaningful. This option is not useful if the key is an enumerated array (for which all values are supplied).

The *skipMissing* option increases memory usage and should be avoided, if possible.

The format for the *organization* option is

```
options organization same;
```

```
options organization contains;
```

```
options organization unknown;
```

The organization option provides hints about the hierarchy organization that allow for more efficient algorithms. This option is most useful if no hierarchy aggregation is done. The *same* value specifies that all nodes in the hierarchy contain entries for the same item (for example, all nodes could contain “appliances,” “clothing,” “electronics,” and “furniture”). The *contains* value indicates that a parent node contains entries for all values that its children contain. For example, if a node contains “appliances,” its parent node must also contain “appliances,” although not all of its child nodes must contain appliances. The *unknown* value means that no assumptions are to be made regarding the contents of individual nodes.

If no organization is specified, the Tree Visualizer determines the organization as follows.

- If there is no aggregate subsection, *unknown* is used.
- If there is an aggregate section, but the *skipMissing* option is provided, *contains* is used; otherwise, *same* is used. Since this is normally correct when an aggregate subsection is provided (unless *skipMissing* is used but nothing is missing), there normally is no need to provide an organization if the aggregate subsection is present.

If the organization specified does not match the data, the results are unspecified. For example, *same* should not be specified, unless all nodes have the same entries.

The View Section

The view section of a data file describes how the hierarchy is displayed, including the mapping of heights, colors, labels, and so forth. A sample view section is:

```
view hierarchy landscape
{
   height sales, normalize levels, max 2.0;
   height legend label "Height: Total sales";
   base height max 1.0;
   disk height target, legend label "Disk height: Target
             sales";
   color pctTarget, scale 0 100 200 500;
   color colors "red" "gray" "green" "blue";
   color legend label "Color: % of target" "0%" "100%"
             "200%" "500%";
   options columns 4;
   message "$%,.2f, %.0f%% of target, %.0f%% of last year",
             sales, pctTarget, pctLastYear;
}
```

The first words of the view section (before the opening brace) describe the type of view. The only view type supported is **view hierarchy landscape**; thus, these words must introduce the view section.

When entering the view section, the *viewHierarchyLandscape.treeviz.options* options file is read in. Note that there is not a simple *view.treeviz.options* options file, the full name *viewHierarchyLandscape* must be used.

Height Statements

The height statement describes how the columns are mapped to the height of objects. It consists of a series of clauses separated by commas. Alternatively, it can be specified as multiple height statements. Thus: the following three examples are equivalent:

-
  ```
  height sales, normalize levels, max 2.0;
  ```

-
  ```
  height sales;
  height normalize levels;
  height max 2.0;
  ```

-
  ```
  height sales, normalize levels;
  height max 2.0;
  ```

The first clause normally contains the name of a column that is to be mapped to height ("sales," in the example). The column must be of a number type (**int**, **float**, or **double**); **float** is the most efficient. If no height column is specified, all bars are flat, and the remaining height clauses have no effect.

normalize Clause

The normalize clause determines the maximum value of the height variable; it normalizes all values relative to that height. Thus, if the maximum value is 30.0, and that bar was given a height of 1.0 (in arbitrary units, as discussed in "The max Clause"), a value of 15.0 would be mapped to a value of 0.5.

The syntax of the normalize clause can be

normalize
: This normalizes all values against one another, throughout the hierarchy.

normalize levels
: This performs independent normalization at each level of the hierarchy.

normalize none
: This performs no normalization, and is the default.

The second form is particularly useful in cases where the data is aggregated up the hierarchy. For example, assume the sales data is aggregated up the company. Comparing the sales of the company as a whole to the sales of a single individual has little meaning; in a large company, the heights of the bars for the individuals are so small as to be indistinguishable from zero. It makes more sense to compare sales people to sales people, offices to offices, regions to regions, and so on. Normalizing levels does this.

Regardless of which form of normalization is used, the base (if shown) is always normalized independently of the bars. By default, the same normalization mechanism for the bars is used for the base.

The max Clause

The max clause is used with the normalize clause to specify the height of the tallest bars. If no max clause is specified, the height is 1.0 in arbitrary units. If after looking at the view, you see that the heights are too low or too high, use the max clause to adjust them. The syntax of the max clause is

```
max float
```

where *float* is a floating point number (the decimal point is optional). For example, to double the heights, specify

```
max 2
```

The max clause must be used with the normalize clause.

The scale Clause

If normalization is not used, the height variable is mapped directly to the height (in the arbitrary height units). The scale clause can scale these values; all values are multiplied by the scale. The syntax of the scale clause is:

```
scale float
```

Do not use the scale clause with the normalize or max clauses.

The filter Clause

Large datasets can contain many graphics. This results in poor performance. In many cases, the data values are small and of little informative value. The filter clause prefilters the data based on the height variable, so that only the nodes with the highest bars are shown. The syntax of the filter clause is:

```
filter > float%
```

The > and % characters must be typed literally. For example:

```
filter > 5%
```

This example filters out all charts containing no bars greater than 5% of the maximum bar height, except for those containing descendants in the hierarchy containing such bars. Note that if a chart contains just one bar that meets this criterion, the entire chart is shown.

The filter value can be changed interactively through the filter panel (see "The Filter Panel" in Chapter 4).

The filter clause is permitted only on the height statement.

The legend Clause

The legend clause defines the meaning of the height mappings. Any string can be placed in the height legend. The legend clause has the following syntaxes:

legend off This turns off the height legend (this is the default).

legend on This turns on the height legend.

legend label *string*
This changes the legend. If legend label is used, legend on is unnecessary.

By default, the legend has the following syntax:

```
height:varname
```

where *varname* is the name of the variable that is mapped to height.

It is possible to declare separate legends for the height, the base height, and the disk height.

Base Height Statements

The base height statement specifies how the height of the base is calculated. The format is similar to the height statement, except that it is preceded by the word “base.” If the base height statement is omitted, the height of the base is calculated using the same values as in the height statement (the same variable, normalization mechanism, max value, and so on). You also can specify only some of the clauses for the base, in which case everything else is the same as the height statement. For example:

```
height sales, normalize levels, max 2.0;
base height max 1.0;
```

In this case, the base height is based on *sales*, and it is normalized by *levels*. The maximum height, however, is only 1.0 instead of 2.0. Usually, the visual effect is better if the base height max is less than the max for the bars.

The filter clause is not permitted on the base height statement.

The on and off Clauses

The initial value of the base height can be turned on and off via the on and off clauses. To turn it off, use

```
base height off
```

To turn it on, use the default:

```
base height on
```

The base height can be changed interactively using the Base Height option in the Display menu. The on and off clauses are valid only with base height. Do not use them with the height or disk height statements.

Disk Height Statements

You can place a disk on each bar to indicate an additional item of data. This is done with the disk height statement. The disk height statement's syntax is similar to that of the height statement, but it is preceded by the word "disk." For a disk to be displayed, there must be a clause specifying the column to be mapped to the disk. Other clauses are optional; if these are omitted, the height statement's defaults are used.

If the height statement has a normalize clause, and the disk height statement has no normalize or max clause, then the disks are normalized with the bars (they are drawn to the same scale). If the disk height statement has either a normalize clause or a max clause, the disks are normalized independently of the bars. For example:

```
height sales, normalize levels, max 2.0;
disk height target;
```

In this case, the bars are mapped to the variable "sales," and the disks are mapped to "target." Both are normalized, with the maximum value of sales or target on each level mapped to a value of 2.0. If instead this example is written as

```
height sales, normalize levels, max 2.0;
disk height target, normalize levels;
```

the bars are mapped so the highest bar at each level is 2.0, and the highest disk on each level is 2.0, but the bars and disks are not mapped to the same scale. This can be used, for example, if the bars represent dollars and the disks represent head count.

The filter clause is not permitted on the disk height statement.

Color Statements

The color statement describes how values are mapped to colors. The format is similar to that of the height statement, consisting of several clauses that can be separated by commas or entered as multiple statements.

Color Naming

Color names follow the conventions of the X Window System™, except that the names must be in quotation marks. Examples of valid colors are "green," "Hot Pink," and "#77ff42." The last one is in the form "*#rrggbb*", in which the red, green, and blue components of the color are specified as hexadecimal values. Pure saturation is represented by ff, a lack of color by 00. For example, "#000000" is black, "#ffffff" is white, "#ff0000" is red, and "#00ffff" is cyan. (A list of available colors is found in the file */usr/lib/X11/rgb.txt*.)

The Color Variable

As with height, you also can specify a single column to be mapped to a color. The column must be a number type. Unlike for height, there is no normalization of colors.

The key Clause

Instead of specifying a variable, the word key can be specified. This assigns a different color based on each key, normally for each bar. For example, if the 50 states were the keys, key assigns a different color to the bar for each state. Since the base is not keyed, when the key clause is used, the base is always gray.

The colors Clause

The colors clause specifies the colors to be used. The colors clause syntax is:

```
colors "colorname" "colorname"...
```

The format for *colorname* is described “Color Naming.” Note that there are no commas between the colors. This is because commas are used to separate clauses in the color statement. A sample colors clause is

```
colors "red" "gray" "blue"
```

Colors in the list are subsequently referred to by their index, starting at zero. In the above example, red is color 0, gray is color 1, and blue is color 2.

If there is no colors statement, colors are chosen randomly; however, if there is a colors statement, at least as many colors must be specified as are to be mapped. If a key is used, there must be one color for each key value.

The scale Clause

The scale clause allows assignment of values to a continuous range of colors. For example, when displaying a percentage, red can be assigned to 0%, gray to 50%, and blue to 100%. Intermediate values are interpolated; for example, 25% is pinkish, and 55% is a slightly bluish gray.

The syntax for the scale clause is

```
scale float float ...
```

The first value is mapped to color 0, the second to color 1, and so forth. The colors statement must contain at least as many colors as are to be mapped to the largest index.

Values in this statement must be in increasing order. Any value less than the first color is assigned the value of the first color. Any value greater than the last value is assigned the last color. Intermediate values are interpolated.

For example, assume the pctFemale column indicates what percentage of the group is female, and you want to map a group that is 100% female to red, 100% male to blue, and 50% each to gray. The colors statement for this is:

```
colors pctFemale, colors "blue" "gray" "red", scale 0 50 100;
```

The buckets Clause

The buckets clause is similar to the scale clause without interpolation. All values are rounded down to the highest value in the clause, and that exact color is used. Values less than the first value use the first color.

The syntax for the buckets clause is

```
buckets float float ...
```

The syntax and assignment of colors is the same as for the scale clause.

If, in the `pctFemale` example, you used the buckets clause instead of the scale clause, the statement would be

```
colors pctFemale, colors "blue" "gray" "red", buckets 0 50
100;
```

All values greater or equal to 100 are colored red. Values greater than or equal to 50, but less than 100, are gray. All other values are blue.

The legend Clause

The legend clause creates a legend of the colors. By default, a legend is on for the bar colors, and off for base and disk colors, although separate legends are permitted for each. The legend clause syntax can be any of the following:

```
legend off
legend on
legend "string" "string" ...
legend label "string"
legend "string" "string" ... label "string"
```

The **legend off** clause turns the legend off. The **legend on** clause turns the legend on. It can be omitted if other legend statements are included. Specifying only **legend on** generates the default legend.

The default legend includes a single label to the left (with the name of the column that is mapped to color), and a list of colored labels on the right (with values obtained from the scale clause, the buckets clause, or from the keys). To override the strings in the colored labels, specify the strings as: `legend "string" "string`.

To override the label on the left, specify it following the word label. To eliminate this label, specify an empty string; that is:

```
legend label ""
```

Base Color Statements

The base color statement controls the color of the base. Its syntax is similar to the color statement, except that it is preceded by the word “base.” If this word is omitted, the base has the same color as the bars. If the base color statement is present, any omitted clauses default to the values of the color statement.

Disk Color Statements

The disk color statement controls the color of the disk. The syntax is similar to the color statement, except that it is preceded by the word “disks.” If the disk color statement is omitted, the disk has the same color as the bars. If the statement is present, any omitted clauses default to the values of the color statement.

Since disks are drawn only if a disk height statement is present, a disk color statement has no effect without a disk height statement.

Label Statements

Label statements specify the labels used when labeling objects in the scene. Normally, these statements can be omitted. By default, each bar is labeled with its key; each base is labeled with its position in the hierarchy. The syntaxes of the label statements are:

`label` *name*

`base label` *name*

`line label` *name*

`back label` *name*

where *name* is the name of the column to be used as the label. The first form is used as the label on the bars. The second form is the label on the bases. The third form labels the lines connecting the bases. The fourth places labels behind the bases. (Note that bases often obscure the back labels, so this form is less useful; however, there might be occasions where it is appropriate.)

Message Statements

The message statement specifies the message displayed when the pointer is moved over an object or when an object is selected. The syntax is similar to that of the C **printf** statement. A sample message statement is

```
message "%s: $%f, %.0f%% of target, %.0f%% of last year",
      product, sales, pctTarget, pctLastYear;
```

This could produce the following message:

```
furniture: $2425.37, 23% of target, 87% of last year
```

The formats must match the type of data being used:

- Strings must use %s.
- Ints must use integer formats (such as %d).
- Floats and doubles must use floating point formats (such as %f).

For details of the **printf** format, see the printf (1) reference (man) page (type **`man printf`** at the shell prompt).

A special format type has been added to **printf**. If the percent sign is followed by a comma (for example, “%,f”), commas are inserted in the number for clarity. Currently, only the United States convention of d,ddd,ddd.dddd is supported, with the decimal point represented by a period, and commas separating every three places to the left of the decimal point. For example, if the above format were:

```
message "%s: $%,f, %,.0f%% of target, %,.0f%% of last year”,
      product, sales, pctTarget, pctLastYear;
```

it would produce the message:

```
furniture: $2,425.37, 23% of target, 87% of last year
```

The $, *, h, l, ll, L, and n **printf** format options are not supported.

All values, including the format string, are expressions. Thus, if you had a pctFemale column, but wanted a more gender-neutral message, you could use

```
message pctFemale>50?"%f%% females":"%f%% males",
      pctFemale>50?pctFemale:100-pctFemale;
```

If pctFemale is 70, the message “70% females” is displayed; if pctFemale is 30, the message “70% males” is displayed. In this case, you can also achieve the same result with a single format string:

```
message "%f%% %s", pctFemale>50?pctFemale:100-pctFemale,
      pctFemale>50?"females":"males";
```

By default, the same message is used for the base as for the bars. It is possible to specify a different message by using a base message statement, which has the same syntax.

If no message is specified, a default message containing the names and values of all the columns is used.

The Execute Statement

The execute statement lets you execute a shell command by double-clicking an object. The syntax is similar to that of the *message* command; however, since hierarchy information is not displayed on a separate line, it is useful to include the hierarchy information and to pass the key information as arguments.

Here is a sample execute statement that uses *xconfirm* to show a window with information about the item. (The first line, the string, is broken into multiple lines to fit into a single page. In an actual file, it should be on a single line. Multi-line strings are not supported.

```
execute "xconfirm -t ‘%s’ -t ‘sales of %s’ -t ‘$%,.0f’
      -t ‘target $%,.0f (%.0f%% of target)’
      -t 'last year $%,.0f, %.0f%% of last year'>/dev/null",
   hierarchy(" "), isSummary()?"everything":product,
   sales, target, pctTarget, lastYear, pctLastYear;
```

This might produce a dialog with the message

```
Eastern Connecticut Milford
sales of clothing
$348
target $427 (81% of target)
last year $372 (94% of target)
```

Note the use of hierarchy(" ") to produce a blank-separated description of the hierarchy. Also note the isSummary()?”everything”:product; this produces the word “everything” if the base was selected, but otherwise produces the product. An alternative to this is using separate execute and base execute statements.

If there is no execute statement, double-clicking an object has the same effect as single-clicking it.

The View Options

The view section has many options. Like other options statements, the options can be separated by commas, or they can appear in separate lines.

Sky and Ground Colors

The sky and ground color can be specified using the following syntax:

```
options sky color colorname
options sky color colorname colorname
options ground color colorname
options ground color colorname colorname
```

The syntax for color names is the same as that for color naming.

For both the sky and the ground it is possible to specify either one or two colors. If only one color is specified, the sky or ground is solid. If two colors are specified, the sky or ground is shaded between the colors. For the sky, the first color is for the top of the sky, the second for the bottom. For the ground, the first color is for the far horizon, the second for the near ground.

For example, to have a solid black background, specify:

```
options sky color "black", ground color "black";
```

Bar Layout

By default, bars in each chart are laid out as close to a square as possible. You can override this using either the rows or the columns option:

```
options rows number
options columns number
```

Only one of these can be specified.

Overview

Although the overview can be brought up using the Show menu, it can also be configured to come up automatically at startup. The overview syntax is:

```
options overview on
options overview off
```

The first form causes the overview to be displayed at startup. The second form (the default) turns the overview off. Regardless of the setting, the overview can be invoked from the Show menu.

Shrinkage

Hierarchies normally have a large aspect ratio, having greater width than depth. In their unaltered form, it is impossible to view the entire hierarchy, except from such a far distance that no detail would be visible. To see the hierarchy more clearly, distant objects can be shrunk more than perspective normally dictates. The shrinkage option lets you control the shrinkage for a given graph. The shrinkage option syntax is any of the following:

```
options shrinkage auto
options shrinkage float
options shrinkage off
```

The first form (the default) automatically calculates a shrinkage value. Its results are usually reasonable, but not necessarily optimal in unusual hierarchical layouts. Thus, you might want to explicitly set the shrinkage using the second form. For hierarchies in which some parts are deeper than others, automatic calculation does not work well. The best shrinkage value depends on the graph being displayed, as well as various layout options such as margins. You should experiment with each graph. Start with a value of 10.0, then make adjustments. Smaller values result in a narrower hierarchy and increased distortion. The shrinkage value must be positive; avoid values smaller than 5.0.

Shrinkage can be turned off. This is recommended only for very small hierarchies, as it produces hierarchies with very large aspect ratios.

Root Label

By default, the root node of the hierarchy gets a label based on the name of the configuration file. You can override this by using the **root label** option. The format is

```
options root label string
```

This option also affects the string displayed when an object is selected, as well as the result of the of **hierarchy**() function.

Note that the root label option has no effect if the base label statement was used (that statement defines the base label for the root as well as for all other bases).

Font

The font option controls the font used for drawing the labels. The syntax is

```
options font "fontname"
```

where *fontname* can be any font in the directory */usr/lib/DPS/outline/base*.

It also can be the string *default*. This attempts to use Helvetica (if available), or the default Inventor font (if Helvetica is not available). Note that different systems can have different fonts installed.

Base Label Color

The base label color option controls the color of the labels in front of the bases. The syntax is

```
options base label color "color"
```

Bar Label Color

The bar label color option controls the color of the labels in front of the bars. The syntax is

```
options bar label color "color"
```

Line Color

The line color option controls the color of the lines connecting the nodes in the hierarchy. The syntax is

```
options line color "color"
```

Zero

The zero option lets you determine whether bars, disks, and bases of height zero are drawn solid, as an outline, or hidden completely. In the last case, space is left for the object, but it is not drawn. The default value is solid. This option can be changed at run time using the Display menu (see “The Display Menu” in Chapter 4).

The syntax for the zero option is

```
options zero solid
options zero outline
options zero hidden
```

Null

The null option lets you determine whether bars, disks, and bases of height null (see Appendix G, “Nulls in MineSet”) are drawn solid, outline, or hidden completely. In the last case, space is left for the object, but it is not drawn. The default value is outline. This option can be changed at run time using the Display menu (see “The Display Menu” in Chapter 4). The syntax is

```
options null solid
```

```
options null outline
```

```
options null hidden
```

Other Options

There are 10 other options to control the layout of the display, level of detail, and other parameters. Generally, it is not necessary to adjust these parameters. The values of many of the options are in arbitrary units. Adjust the options by increasing or decreasing the value. For the default values of these parameters, see the file */usr/lib/MineSet/treeviz/viewHierarchyLandscape*.

- `options speed` *`float`*

  Controls the speed during free-form (middle-mouse) horizontal navigation (forward, backward, and side to side). The larger the value, the faster the motion.

- `options climb speed` *`float`*

  Controls the speed when moving up and down using Shift + middle mouse. The larger the value, the faster the motion.

- `options leaf leaf margin` *`float`*

  Controls the distance between adjacent nodes in the hierarchy. Larger values move the nodes farther away.

- `options root leaf margin` *`float`*

  Controls the distance between a node and its children. Larger values move the nodes farther away.

- `options leaf edge margin` *`float`*

  Adds margin space next to nodes at the edge of a subhierarchy.

- `options initial position` *`float float float`*

  Provides the initial *x*, *y*, and *z* position from which the scene is viewed. A value of 0 0 0 positions the viewer at the root of the hierarchy; since the user is looking forward, the root probably is not visible. Increasing *x*, *y*, and *z* moves the camera to the right, up, and back, respectively. A typical position has a zero *x*, positive *y*, and positive *z*. If unspecified, the initial position depends on the layout of the hierarchy.

- `options initial angle` *`float`*

  Provides the initial angle, measured in degrees, from which the hierarchy is viewed. The value must be between 0 and 90. A value of 0 looks at the scene horizontally; a value of 90 looks straight down.

- `options bar label size` *`float`*

  Specifies the size of the labels in front of the bars. Larger values result in larger labels.

- `options base label size` *`float`*

  Specifies the size of the labels in front of the bases. Larger values result in larger labels.

- `options lod [bar` *`float float`*`] [bar label` *`float`* `[`*`float`*`]] [base` *`float float`*`] [base label` *`float`*`[`*`float`*`]] [disk` *`float`*`] [motion` *`float`*`]`

  Controls the level of detail. The parameters can appear in any order, be omitted, or placed in multiple **lod** options. These options control the changing form, or disappearance of, objects, thus providing better system performance.

Except for the **motion** parameter, all float values represent the size of the object when the form change or disappearance takes place. The smaller the value specified, the smaller and farther away the object is when the change takes place. Smaller values provide nicer graphics but slower system performance. The numbers of the different parameters cannot be compared directly because the size of the object also determines when the change takes place. A value of 0.0 means no level of detail changes for that parameter. This setting can significantly slow the rendering process.

bar controls when a bar is drawn with less detail. The first value specifies when the object is drawn as a pair of planes; the second value specifies when the object is drawn as a single line.

bar label controls when the labels on the bars disappear. If two values are specified, the first value specifies when the label is drawn in a lower-quality, fast font; the second value controls when it disappears.

base controls when the bases, and the bar charts in front on top of them, disappear. The first number is based on the width of the base; the second on the height of the base plus the tallest bar on it.

base label controls when the label in front of the base disappears. If two values are specified, the first value specifies when the label is drawn in a lower-quality, fast font; the second value controls when it disappears.

initial depth controls the initial depth to which the hierarchy is viewed. When you are at the top of the hierarchy, you see only the number of hierarchical levels specified by the slider. The nodes in the rows are arranged to optimize their visibility. When navigating to nodes lower in the hierarchy, additional rows are made visible automatically. The nodes above them automatically adjust their locations to accommodate the newly added nodes; thus, some nodes might seem to move. Note that the overview shows all nodes in the hierarchy, not just the top nodes, so the layout of the overview might not match the layout of the main view. The X in the overview approximates the corresponding location in the main view; there is no exact mapping between the two layouts.

An initial depth of zero, or one greater than the depth of the hierarchy, shows the entire hierarchy.

Once the Tree Visualizer is running, the depth can be changed through the filter panel.

disk controls when the disk disappears.

motion controls changes in some of the level of detail calculations when the scene is animated. A value greater than 1.0 defaults to 1.0. A value of 1.0 specifies that motion has no effect on the level of detail. Smaller values change the level of detail at a proportional distance. For example, a value of 0.5 means that during animation, level of detail changes occur at half the normal distance.

Appendix B

Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer

The first part of this appendix describes the types and formats of data supported by the Map Visualizer. Data input to the Map Visualizer must be provided as a single file containing raw data, usually in a tab-separated ASCII text form.

The second part discusses the configuration file, which describes how the Map Visualizer reads in, and displays, the data file.

Both the data and configuration files can be generated automatically by the Tool Manager (see Chapter 3).

Note: Read Chapter 5, “Using the Map Visualizer,” before using this appendix.

The Data File

In its simplest form, the data file consists of a list of lines, each containing a set of fields separated by one tab. (Other separators are also allowed, but only one can separate each field. See “Input options” on page 387.) All lines must contain the same fields. The interpretation of the fields is specified by the configuration file, described in “The Configuration File” on page 376. Using the U.S. population data (*examples/population.usa.data* file), provided as part of the Map Visualizer package, the first few lines of this input file appear as shown below:

```
AL    0 0 0 1000 9000 127901 309527 590756 771623 964201
996992 1262505 1513401 1828697 2138093 2348174 2646248
2832961 3061743 3266740 3444354 3894025 4040587    51705
AR    0 0 0 0 1000 14000 30000 98000 210000 435000 484000
803000 1128000 1312000 1574000 1752000 1854000 1949000
1910000 1786000 1923000 2286000 2351000    53187
```

```
AZ    0 0 0 0 0 0 0 0 0 0 10000 40000 88000 123000 204000
334000 436000 499000 750000 1302000 1775000 2717000
3665000    114000
CA    0 0 0 0 0 0 0 0 93000 380000 560000 865000 1213000
1485000 2378000 3427000 5677000 6907000 10586000 15717000
19971000 23668000 29760021    158706
```

In this example, the first column is a two-character string identifying the graphical object: the state. (This string locates a record in a *.gfx* file containing information about the shape of the graphical object.) The tab separator is followed by a grouping of 23 numeric values, which represent the state's population from 1770 through 1990, in 10-year increments. The next tab separator is followed by a single numeric value, which specifies the state's area in square miles.

The data file cannot contain blank lines or comments. Missing or extra data on a line causes an error.

Note: One tab (the default separator) separates each field. Do not insert multiple tabs to line up the fields visually; this generates blank fields. The order of the columns must match the format specified by the configuration file.

Any field in the data can also be a "?", indicating that the data is null (unknown). See Appendix G, "Nulls in MineSet."

Data Types

The Map Visualizer supports integer, floating point number, and string data types, as well as arrays of these types. The following five data types are supported:

- **int** represents a 32-bit signed integer.
- **float** represents a single-precision floating point number. The decimal point is optional. Numbers in exponential “e” notation are also accepted.
- **double** represents a double-precision floating point number. The decimal point is optional when representing a floating point number. Numbers in exponential “e” notation are also accepted. The superior precision of **double** can be useful for accurately representing large numbers, since **float** can represent only seven or eight significant digits accurately. This superior accuracy, however, consumes twice the memory space of **float**.
- **dataString** represents a string that is unlikely to appear multiple times. If it appears multiple times, multiple copies are made.
- **string** represents a string of characters that can appear multiple times in the data file. Unlike a **dataString**, only a single copy of a given string is stored in memory, no matter how many times it appears in the data. This saves memory for strings appearing many times.

  Comparing **strings** is also much quicker than comparing **dataString**s. Processing is somewhat slower when looking for duplicate strings as they are read in. An example of **string** use is for a division name that appears once for each department in the division. If you are unsure whether to use a **string** or a **dataString**, use a **string**.

Fixed Arrays

With the Map Visualizer, you can use one- or two-dimensional arrays of fixed size. In a fixed-sized array, all entries of the given type have the same number of values. Arrays contain the data values across one or two independent variables, that is, those dimensions controlled by the sliders.

A variant of the “enumerated array” is the “null enumerated array.” This is a variant of the enumerated array with an additional entry at the beginning for null, which is represented by “?”.

The Configuration File

The configuration file format is flexible. Words in it must be separated by spaces, and it is case-sensitive. Except for the include statement and text within quoted strings, spacing and line breaks are irrelevant.

Overview

The configuration file's structure and grammar are explained in the following sections.

Sections

The configuration file consists of a series of sections, each of which has the following syntax:

sectionKeyword
```
{
   statements...
}
```

where *sectionKeyword* names the section. A semicolon (;) can follow the closing brace (}) but is not required. The order of the sections is significant, since sections can refer to variables defined in previous sections.

Defaults Files

As each section is encountered, a special configuration file (referred to as a *defaults file*) is also read in. The defaults file has the same name as the section. Defaults files contain options statements. These files are searched in the following order, as specified by the X-resource *Mapviz*configPath* in the file */usr/lib/X11/app-defaults/Mapviz*.

1. The directory */usr/lib/MineSet/mapviz*. This directory contains system defaults.
2. The *~/.MineSet* directory (where the tilde, ~, indicates your home directory). You can set up personal defaults in this directory.
3. The current directory. This lets you set up defaults for each directory.

Files with the same name can appear in more than one of the above-named directories; in this case, the order given is the one in which the directories are read. If the same option is found in multiple files, the last option read is used. Note that the appropriate section in the configuration file is read after all the defaults files; thus, options in the configuration file override those in the defaults files.

Statements

A statement has the following syntax:

statementKeyword info ;

where *statementKeyword* defines the statement, and *info* varies according to the keyword. A statement can be another section (using the brace format defined under “Sections” on page 376).

Variable Names

A variable name can appear in two formats:

- In the first format, it is a letter followed by a number of letters, digits, or underscores. It cannot be a keyword, and should not be placed in quotation marks.
- In the alternate form, the variable name should be surrounded by back quotes (‘). In this form, the variable name can match a keyword, and can contain even non-alphanumeric characters. The primary purpose of this second form is for configuration files generated automatically by the Tool Manager.

There is no scoping of variable names; a given variable name can be declared only once in the configuration file.

Option Statements

Many sections have options statements, which have the syntax:

```
options key info, key info... ;
```

where *key* defines the specific option, and *info* depends on the key. In some cases, the *key* can be more than one word. To maximize the number of allowable variable names, most option keys are meaningful only within the appropriate option statement; keys do not conflict with variable names. You can declare several options on the same line, separating them by commas or placing them in several options statements. If two conflicting values for the same option appear, the last value is taken.

Include Statements

The configuration file may contain lines of the form

```
include "filename"
```

These lines can appear anywhere in the configuration file, but each must be on its own line. The filename must be in quotes; anything after the closing quote is ignored. The number of nested includes is unlimited. If a relative pathname (one not beginning with a slash) is specified, the file is first sought in the directory containing the current configuration file. If include statements are present, this might not be the same as the initially loaded configuration file. If it is not found in the current configuration file, the include is sought in the current directory. If the file is not found, an error message appears.

Sinclude Statements

A statement similar to an include is sinclude, which has the syntax:

```
sinclude "filename"
```

This is identical to the include statement, except that no error is given if the file does not exist; instead, the sinclude statement is ignored.

Strings, Characters, and Comments

Strings and characters in the configuration file follow C conventions. Strings are in double quotation marks (“), and characters are in single quotation marks (’). All standard backslash conventions are followed (for example, `\n` represents a new line).

Comments begin with a pound (#) symbol at the beginning of a line; anything after this symbol to the end of the line (80 characters) is ignored.

Keywords

The currently recognized keywords are listed below. Variables can not have these names unless they are surrounded by back quotes (‘). Tokens appearing only in option statements are not keywords, and can be used for variable names.

Table B-1 Keywords for the Map Visualizer

buckets
color
colors
datapoints
dataString
date
divide
double
enum
execute
expressions
file
float
from
height
input
int
key
label
legend
level
map
message
modulus
monitor
null
objects
off
on
options
outlines
scale
separator
slider
string
summary
title
to
view

Expressions

Expressions are accepted in several places in the input. Expressions follow standard C syntax The following operations supported:

```
+  -  *  /  %  ==  !=  >  <  >=  <=  &&  ||  !  &  |  ^  ?:
```

Also, the following functions are available:

- **divide**(x, y, z) divides *x* by *y*, unless *y* is zero. If *y* is zero, the result is *z*; this is equivalent to y==0 ? z : x / y.
- **modulus**(x, y, z) is similar to **divide**, but for modulus.

Type handling is similar to that in C. Expressions using **int** and **float** promote both sides to float. Expressions using **int** and **double**, or **float** and **double** promote both sides to double. The result of a relational expression (for example, ==, <) is always an **int**. Type casting is also supported.

Unlike in C, strings can be compared using relational expressions; the strings are compared lexicographically.

The following sections explain the use and syntax of the Map Visualizer configuration file's input, expression, and view geography sections.

The Input Section

The first section of a data file is normally the **input** section. It defines the name and format of the data file. A typical input section might look like this:

```
input
   {
   file
      "/usr/lib/MineSet/mapviz/examples/population.usa.data";
   enum int Year from 1770 to 1990 by 10;
   string states;
   float population[enum Year] separator ' ';
   float sqMiles;
   }
```

This example specifies that the input data file is called *population.usa.data*, and that there are three tab-separated (the default) fields as follows:

- one of type **string**
- one a fixed-length vector of type **float**, with each value separated by a space
- one a scalar value of type **float**

When the input section is entered, the defaults file, */usr/lib/MineSet/mapviz/input.mapviz.options,* is read in.

File Statements

The **file** statement names the data file to be read. This statement is required. Its syntax is

```
file "filename";
```

The file name must be in double quotation marks. If it is a relative pathname (no leading slash), it is first sought in the directory containing the current configuration file. If include statements are present, this might not be the same as the initially loaded configuration file. If it is not found in the current configuration file’s directory, the file is sought in the current directory.

Enum Statements

Enum statements declare enumeration variables that index into array fields. The enum statement has three forms.

- The first form is

  `enum` *`type name`* `from` *`value1`* `to` *`value2`* `by` *`increment`*`;`

  This declares an enum with values starting at *value1* and incremented by *increment* until they reach or exceed *value2*. For example, the statement:

  ```
  enum int age from 20 to 70 by 10;
  ```

  declares age as an array dimension with the values 20, 30, 40, 50, 60, and 70.

Type must be a number type (**int**, **float**, or **double**) or **date** (see “Dates” on page 383).

- The second enum statement form is

  `enum` *`type name`* `from` *`value1`* `to` *`value2`* `across` *`numberOfValues`*`;`

  This declares an enum with values ranging from *value1* to *value2*. The *numberOfValues* is an integer specifying the number of values. For example, the statement

  ```
  enum int age from 20 to 70 across 6;
  ```

  declares age as an enum with the values 20, 30, 40, 50, 60, and 70.

  Type must be a number type (**int**, **float**, or **double**) or **date** (see “Dates” on page 383).

- The third enum statement explicitly lists the enumeration values. Its form is

  `enum` *`type name`* `{` *`value1, value2, ..., valueN`* `};`

  Type can be any type or date (see “Dates” on page 383).

Dates

The enum statement includes special support for a date type that handles date and time values starting Jan 1, 1753. The date type is valid only within enum statements. A date enum statement can have the following syntaxes:

```
enum date “format” name from “value1” to “value2” across
        numberOfValues;
enum date “format” name { value1, value2, ..., valueN };
enum date “format” name from “value1” to “value2” by
        “increment”;
```

The *format* string specifies the format of the values; it is useful for controlling how dates are displayed in the animation control panel. The syntax of the *format* string is similar to the scanf function in C. Various units of time are represented by special characters preceded by the percent symbol (%). For example:

```
enum date cq “Calendar Q%Q, %Y” from “Calendar Q1, 1980” to
“Calendar Q3, 1985” by “1 quarter”;
```

The “Calendar Q” in the *format* string matches the “Calendar Q” in *value1* and *value2*. The %Q in the *format* string indicates that the next number in *value1* and *value2* is the calendar quarter. The comma and space in the *format* string match the commas and spaces in the values. Finally, the %Y in the *format* string specifies that the year values are next.

Table B-2 lists the characters that can follow the percent symbol and the units of time they represent.

Table B-2 Characters That Can Follow the Percent Symbol in the format String

Character	Time Unit	Precision
Y	year	4
Q	calendar quarter	1
M	month	2
N	month name	>= 3
D	day	2
h	hour	2
m	minute	2
s	second	2

With the exception of N, each character matches an integer of the specified precision. N matches 3 or more characters giving the English name of the month.

The from-to-by form of the enum statement includes an increment value. For dates, the increment is a quoted string containing an integer, an optional space, and one of the special characters in Table B-2 or one of the symbols **year**, **quarter**, **month**, **day**, **hour**, **minute**, and **second**. The plural forms of these symbols are also accepted. Note that these symbols are not keywords, since they have special meaning only in the increment string. The following are examples of valid increments:

```
“1 year”
“7 days”
“4h”
```

Data Statements

The data statements declare the columns in the data file. The columns must be declared in the order they appear in the data file. The format of most data statements is

```
type name;
```

where *type* is **int**, **float**, **double, string**, or **dataString**; *name* is the variable name. Unlike in C, only one variable can be declared per statement.

Fixed Arrays

Fixed arrays can also be declared using simple numeric data declarations; however, if you also are going to declare a slider, you must use the enum declaration form. The declaration syntax is

```
type name [ number ] ;
```

For example:

```
float revenue [50];
```

You can also override the separator by declaring it as

```
type name [ number ] separator ’char’;
```

For example:

```
float revenue [50] separator ’:’;
```

If no separator is specified, the default separator (usually a tab) is used.

Fixed arrays can also be two-dimensional, such as

```
enum string products {“bread”,“milk”,”cheese”,”cereal”,
”apples”,”lettuce”,”juice”,”toothpaste”,”soap”,”eggs”};
```

```
enum year from 1985 to 1994 by 1;
```

```
float prices[enum products][enum year];
```

or

```
float prices[10][20];
```

which might be used for an array of prices for a set of 10 products over a 20-year period.

Using the *prices* array, for example, if you specified in the Tool Manager that data was to be retrieved from the database in "wide" mode (with a bin for null values), the enumerated *products* are declared as:

```
float prices[null enum products][enum year];
```

and the first column contains the prices for unknown products (products not in the enumerated list of ten known products) declared in the *enum string products* statement.

Input options

The input section of a data file has several options. All **options** statements begin with the word “options” and have one or more comma-separated options.

- The separator option defines the separator between columns in the data file. The default separator is a tab. The syntax is

  ```
  options separator ′char′;
  ```

  For example:

  ```
  options separator ′:′;
  ```

  Note: Arrays can override the separator.

- The monitor option allows a dynamic update of the data displayed. When the specified file is changed (for example, through the UNIX *touch* command), the data file (not the configuration file) is reread. Note that although the data file could be used to trigger the updates, it is better to use a different file so that the data file is not read while it is being updated. The syntax of the monitor option is:

  ```
  options monitor "filename";
  options monitor "filename" timeout;
  ```

  where *filename* is the file to watch, and the optional *timeout* specifies the number of seconds to wait after the file changes. If the user interacts with the application in any way during this timeout (via the mouse or keyboard), the timeout restarts. Updating the file can take a few seconds. By specifying a timeout, the chances of an update occurring while the user is interacting with the tool are minimized. This might delay the update. If no timeout is specified, the update occurs immediately.

  The file being monitored must exist at the start of the program. When this file is being updated, it must not be removed and re-created; instead, only its modify time should be updated (for example, through the *touch* command). If the file is deleted, subsequent updates are not shown.

Suppose a program extractor extracts data from a database into a data file. If you want the program to update the data file every 10 minutes, the script you write might look like this:

```
extractor > dataFile;           # create first data file
touch trigger;                  # create the trigger file
while (sleep 600)               # sleep 10 minutes
do
extractor > dataFile;           # create new data file
touch trigger;                  # force a reread
done &                          # this loop goes in the
                                # background
mapviz configFile;              # run mapviz
kill $!                         # when mapviz exits, kill
                                # the update loop
```

The monitor option can be used only if the file alteration monitor */usr/etc/fam* is installed (this can be found in the subsystem *desktop_eoe.sw.fam*).

The input section of configuration file might look like this:

```
input
{
   file "dataFile:
   #data declarations here
   options monitor "trigger" 15;
}
```

- The backslash option controls whether backslashes in the input data are treated specially or like other characters. The syntax is:

```
 options backslash off;
 options backslash on;
```

The default is off. If backslash processing is on, separators in the input data preceded by backslashes are treated as regular characters rather than separators. Also, within strings standard C-style backslash processing is done.

The Expressions Section

The expressions section pf a data file lets you define additional columns that are expressions of existing columns. For example, one column can be defined as the sum of two other columns. The following is a sample expression section. This section assumes two existing fixed-length columns of type **double**: “male” and “female”; these represent spending by males and females on various goods across time (one independent dimension). Two columns are added: “total” represents the total dollars spent, and “pctFemale” represents the percentage of dollars spent by females.

```
expressions
{
double total[enum month] = male+female;
double pctFemale[enum month] = divide(female*100,total,50.0);
}
```

Note: The pctFemale calculation uses “total,” defined in the previous section. Also, note the use of the divide function rather than the / operator. This results in 50% for the case where there are no dollars spent at all; using the / operator generates a divide by zero error in such a case. (The divide function is described in the “Expressions” section.)

The format of the expressions section is

```
expressions
{
   expressionDeclaration;
   ...
}
```

where *expressionDeclaration* has the following syntax:

```
type name = expression ;
```

The format of *expression* has already been described.

Since the expressions section has no options, no defaults file is read in for it.

The View Section

The view section of a data file describes how the graphic objects are displayed, including the mapping of heights, colors, labels, and so forth. A sample view section is

```
view map
{
   map objects "usa.states.hierarchy";
   slider Year;
   height population;
   height legend label "Height: U.S. Population (1770-1990)";
   color density, scale 0 250 500 750 1000;
   color colors "white" "#ffc0c0" "#ff8080" "#ff4040" "red";
   color legend label "Color: Pop. Density" "0/sq-mile"
            "250/sq-mile" "500/sq-mile" "750/sq-mile"
            "1000/sq-mile";
   message "population %,.0f   %,.1f per sq mile",
   population, density;
   execute "xconfirm -t 'Population %,.0f'
                  -t 'averaging %,.1f per sq mile'
                  -t 'across %,.0f sq-miles' > /dev/null",
        population, density, sqMiles;
   }
```

The first words of the view section (before the opening brace) describe the type of view. The only view type supported is **view map**; thus, these words must introduce the view section.

When entering the view section, the *viewMap.mapviz.options* defaults file is read in. Note that there is no simple view defaults file, so the full name *viewMap.mapviz.options* must be used.

Title Statement

The **title** statement inserts a title string at the bottom of the main window. The syntax is

```
title string;
```

where *string* is a string enclosed by double- quotation makes.

Map Statement

The map statement specifies how the graphical objects are to be drawn in the main window. The map statement has three possible syntaxes: one required, the other two optional. The required syntax is

```
map objects hierarchy_filename;
```

where "objects" is a keyword, and *hierarchy_filename* is a filename enclosed in double quotation marks. This statement names the *.hierarchy* file describing the 3D graphical objects that exhibit heights and colors.

The following **map** statements are optional:

- `map outlines hierarchy_filename;`

  Declares graphical objects that are drawn as flat lines on which the **map objects** objects are placed. See the samples provided in *examples/population.usa.cities.mapviz*.

- `map level column_name;`

  Specifies an alternative level of the geographical hierarchy for initial display. For example, in the *examples/population.usa.mapviz* file, the unstated default is

  ```
  map level states;
  ```

  and the main window initially displays individual states. If, instead, the configuration file specified

  ```
  map level eastWest;
  ```

  the main window initially displays the United States as two halves: East and West.

Slider Statement

The **slider** statement identifies a key to be used as a slider dimension. Its syntax is

```
slider [enum] enumName;
```

where *enumName* is the name of an enum variable declared in the input section. Note that the **enum** keyword is optional.

There can be 0, 1, or 2 slider statements. The first slider statement applies to the horizontal slider. The second slider statement applies to the vertical slider. If there is no slider statement, the resulting display does not include animation.

No slider statement is required if “height” and “color” map to non-array variables. One slider statement can be included if “height” and “color” map to one-dimensional arrays. Two slider statements can be included if “height” and “color” map to:

- two-dimensional arrays, or
- one-dimensional arrays, where dimensions are enum variable names that one of the sliders controls.

Height Statement

The **height** statement describes how the columns of data are mapped to the height of objects. It consists of a series of clauses separated by commas. The first clause normally contains the name of a column to be mapped to height (“population,” in the example in the section “The View Section” on page 390). The column must be of a number type (**int**, **float**, or **double**), of which **float** is the most memory-efficient. If the column is a fixed-length array, the **view** section also must contain at least one, and no more than two, **slider** statements.

If no height column is specified, all bars are flat, and the remaining height clauses have no effect.

The **scale** clause lets you scale the height values. Normally, the height variable is mapped directly to the height of the graphical objects, so that the tallest object (with the largest numeric value) rises towards the top of the view window. With the optional scale clause, all values are multiplied by the scale. The scale clause syntax is

```
scale float
```

The **legend** clause defines the meaning of the height mappings. Any string can be placed in the height legend. The legend clause has the following syntaxes:

legend off — This turns off the height legend (this is the default).

legend on — This turns on the height legend. The legend can be changed by using the legend label form, in which case **legend on** is unnecessary. The legend's default syntax is

```
height:varname
```

where *varname* is the name of the variable that is mapped to height.

legend label *string*

where *string* is the name of the variable that is mapped to height. The legend can be changed by using the legend label form. If **legend label** is used, **legend on** is unnecessary.

Color Statement

The color statement describes how values are mapped to colors. The format is similar to that of the height statement, consisting of several clauses that can be separated by commas or entered as multiple statements.

Color naming follows the conventions of the X Window System, except that the names must be in quotation marks. Examples of valid colors are “green,” “Hot Pink,” and “#77ff42.” The last one is in the form “*#rrggbb*”, in which the red, green, and blue components of the color are specified as hexadecimal values. Pure saturation is represented by ff, a lack of color by 00. For example, “#000000” is black, “#ffffff” is white, “#ff0000” is red, and “#00ffff” is cyan.)

The **color** variable lets you specify a single column to be mapped to a color (as with height). The column must be a number type.

The **colors** clause specifies the colors to be used. The colors clause's syntax is

```
colors "colorname" "colorname"...
```

The format for *colorname* is described above. Note that there are no commas between the colors. This is because commas are used to separate clauses in the color statement. A sample colors clause is

```
colors "red" "gray" "blue"
```

Colors in the list are subsequently referred to by their index, starting at zero. In the above example, red is color 0, gray is color 1, and blue is color 2.

If there is no colors statement, colors are chosen randomly; however, if there is a colors statement, at least as many colors must be specified as are to be mapped.

The **scale** clause allows assignment of values to a continuous range of colors. For example, when displaying a percentage, red can be assigned to 0%, gray to 50%, and blue to 100%. Intermediate values are interpolated; for example 25% is pinkish, and 55% is a slightly bluish gray.

The syntax for the scale clause is

```
scale float float ...
```

The first value is mapped to color 0, the second to color 1, and so forth. The colors statement must contain at least as many colors as are to be mapped to the largest index.

Values in this statement must be in increasing order. Any value less than the first color is assigned the value of the first color. Any value greater than the last value is assigned the last color. Intermediate values are interpolated.

For example, assume the pctFemale column indicates what percentage of the group is female, and you want to map a group that is 100% female to red, 100% male to blue, and 50% each to gray. The colors statement for this is:

```
colors pctFemale, colors "blue" "gray" "red", scale 0 50 100;
```

The **buckets** clause is similar to the scale clause without interpolation. All values are rounded down to the highest value in the clause, and that exact color is used. Values less than the first value use the first color.

The syntax for the buckets clause is

```
buckets float float ...
```

The syntax and assignment of colors is the same as for the scale clause.

If, in the pctFemale example, you used the buckets clause instead of the scale clause, the statement would be:

```
colors pctFemale, colors "blue" "gray" "red", buckets 0 50
100;
```

All values greater or equal to 100 are colored red. Values greater than or equal to 50, but less than 100, are gray. All other values are blue.

The **normalize** clause controls a form of color normalization, analogous to height normalization. By default, color normalization is off. The syntax is

```
normalize off;
normalize on;
```

When color normalization is on, the color scale (or buckets) list of values must range between 0 and 100. These color values then represent relative percentages of the range from the minimum to the maximum for a given viewed scene. For example,

```
color totalSales;legend off
color scale 0 100, colors “white” “red”, normalize on;
```

generates colors in the range of “white” to “red,” where “white” corresponds to the minimum “totalSales” and “red” corresponds to the maximum “totalSales” for the particular set of graphical objects being viewed. See */usr/lib/MineSet/mapviz/examples/variations.articles.france.mapviz* for a more elaborate example.

The **legend** clause creates a legend of the colors. By default, the color legend is off. The legend clause syntax can be any of the following:

```
legend off
legend on
legend "string" "string" ...
legend label "string"
legend "string" "string" ... label "string"
```

The **legend off** clause turns the legend off. The **legend on** clause turns the legend on. It can be omitted if other legend statements are included. Specifying only **legend on** generates the default legend.

The default legend includes a single label to the left (with the name of the column that is mapped to color), and a list of colored labels on the right (with values obtained from the scale clause, the buckets clause, or from the keys). To override the strings in the colored labels, specify the strings as:

```
legend "string" "string
```

To override the label on the left, specify it following the word label. To eliminate this label, specify an empty string; that is

```
legend ""
```

Execute Statement

The **execute** statement lets you execute a shell command by double-clicking an object. The syntax is similar to that of the *message* command.

Here is a sample execute statement that uses *xconfirm* to show a window with information about the item. Note that the command line (string) is shown as three lines. In an actual file, this should be on a single line. Multi-line strings are not supported.

```
execute "xconfirm -t '%s' -t 'population %,.0f' -t '%,.0f per
    sq mile' -t '%,.0f sq-miles' > /dev/null", states,
    population, density, sqMiles;
```

This might produce a dialog with the message:

```
CA
64 per sq mile
266,807 sq-miles
```

If there is no execute statement, double-clicking an object has the same effect as single-clicking it.

Message Statement

The **message** statement specifies the message displayed when an object is selected. The syntax is similar to the C **printf** statement. A sample message statement is

```
message "%s: $%f, %.0f%% of target, %.0f%% of last year",
      product, sales, pctTarget, pctLastYear;
```

This could produce the following message:

```
furniture: $2425.37, 23% of target, 87% of last year
```

The formats must match the type of data being used:

- Strings must use %s.
- Ints must use integer formats (such as %d).
- Floats and doubles must use floating point formats (such as %f).

For details of the **printf** format, see the printf (1) reference (man) page (type **`man printf`** at the shell prompt).

A special format type has been added to **printf**. If the percent sign is followed by a comma (for example, “%,f”), commas are inserted in the number for clarity. Currently, only the United States convention of d,ddd,ddd.dddd is supported, with the decimal point represented by a period, and commas separating every three places to the left of the decimal point. For example, if the above format were:

```
message "%s: $%,f, %,.0f%% of target, %,.0f%% of last year",
         product, sales, pctTarget, pctLastYear;
```

it would produce the message:

```
furniture: $2,425.37, 23% of target, 87% of last year
```

The $, *, h, l, ll, L, and n **printf** format options are not supported.

All values, including the format string, are expressions. Thus, if you had a pctFemale column, but wanted a more gender-neutral message, you can use:

```
message pctFemale>50?"%f%% females":"%f%% males",
         pctFemale>50?pctFemale:100-pctFemale;
```

If pctFemale is 70, the message "70% females" is displayed; if pctFemale is 30, the message "70% males" is displayed. In this case, you can also achieve the same result with a single format string:

```
message "%f%% %s", pctFemale>50?pctFemale:100-pctFemale,
         pctFemale>50?"females":"males";
```

If no message is specified, a default message containing the names and values of all the columns is used.

Summary Statement

The **summary** statement specifies the initial setting of the Show Data Points pulldown menu option. The syntax is

```
summary datapoints on;
```

or

```
summary datapoints off;
```

The **summary** statement is optional, and the default setting is *off*.

The Hierarchy File

The hierarchy file defines the object hierarchy, allowing objects to be displayed at different levels of aggregation. It enables the drill up and drill down capabilities of the Map Visualizer (see “File Requirements” in Chapter 5). The hierarchy file is specified in the *.mapviz* configuration file with the `map object` ***`hierarchy_filename`*** statement (see “The View Section” on page 390 and “Map Statement” on page 391).

Here are the first few lines of the *usa.states.hierarchy* file:

```
states        regions        eastWest        USA
usa.states.gfx        usa.states.gfx
     usa.states.gfx        usa.states.gfx
AL        E_S_CENTRAL        USA_E        USA_ALL
AR        W_S_CENTRAL        USA_W        USA_ALL
AZ        MOUNTAIN        USA_W        USA_ALL
CA        PACIFIC        USA_W        USA_ALL
CO        MOUNTAIN        USA_W        USA_ALL
CT        NEW_ENGLAND        USA_E        USA_ALL
DE        MID_ATLANTIC        USA_E        USA_ALL
```

This defines how states combine into regions, sectors, and into a single object encompassing all states.

The first record is a list of column names of the hierarchy; each name must be separated by a single tab (‘\t’) character. One of the column names must match a type **string** column in the data file, as declared in the configuration file’s **input** section on page 380). In this example, the first column name, *states*, is also the name of a data column in the example *population.usa.mapviz*. The number of column names in this record must be the same as the number of columns of hierarchy data, beginning at the third record of the *.hierarchy* file. If there is only one column name (for example, *gfx_files/canada.provinces.hierarchy*), then there are only two records in the *.hierarchy* file.

The second record is a list of *.gfx* file pathnames, where each pathname is separated by a single tab (‘\t’) character. Each column name in the first record must have a matching *.gfx* file pathname.

If there is a single column name (and *.gfx* file pathname), then only these two records must be in the file. If there are multiple column names and pathnames, then starting at the third record in the *.hierarchy* file is an N-column table of keywords of graphical objects, where N is the number of column names in the first record. Looking at the sample file, the first column contains "states" keywords, the second column "regions" keywords, the third the "eastWest" keywords, and the fourth the "USA" keyword. The matching *.gfx* files contain the positions and shapes of each of the column's graphical objects.

The third and remaining records in the hierarchy file are the hierarchy data. These records define how objects at one level correspond to objects at other levels.

The .gfx File

The .gfx files define the geometry of each object used by the Map Visualizer when displaying the objects. Each *.gfx* file contains multiple records, one for each object being displayed. Each record contains:

- the gfx keyword name
- the gfx full name
- the vertex pair count
- the shape hint
- the vertex pairs

The following steps guide you through the procedure for building *.gfx* files.

1. Using a digitizing scanner, convert a geographical image into an RGB image file format. Note that the image itself is not used by the Map Visualizer; it is just used as a template for defining the graphical objects in Step 6 on page 401.
2. Launch the i3dm application in */usr/demos/bin/*. (If this application is not currently installed, it can be installed from the IRIX™ 5.3 or 6.2 distribution, in the subsystem demos.sw.tools.) This creates windows on your screen: a Menu window on the left, an Input window across the bottom, and four windows (labeled *TOP*, *Pers*, *Front*, and *Right*) on the right. All *i3dm* windows must remain displayed (not iconified) for i3dm to work.
3. Move the cursor to the Front window.
4. Press the right mouse button to display options. Continue holding the right mouse button, and scroll to the Image Background option, then to the Load Image option. The Input window (at the bottom of your screen) prompts you for a name to apply to this image.
5. Enter the name of the RGB image file. The image appears in the Front window.
6. Delineate the shape of each object in the image by pointing and clicking at significant points on the boundary of each object. Do this in a clockwise sequence for each object. Each identified point is called a "vertex" and is represented by numeric x- and y-axis values. These values are assigned by the i3dm application and exist in a relative frame of reference for that RGB image file. The following procedure is used to delineate each object's shape:
   - Use the middle mouse button to drag the image in the Front window so that the object you are going to delineate is completely exposed. If this is not possible, see step 8.
   - Go to the Menu window, and click the right mouse button on the Create pulldown menu.
   - Choose the Line option.
   - Start the point-and-click process of selecting vertices with the left mouse button in the Front window. Note that the greater the number of vertices you identify, the more accurate the resulting graphical image is.

- Note the red line crosshairs as you move the cursor over the image. As you click the left mouse button to declare each vertex, a small red box appears at that point. The box of the previous vertex changes to a small “x,” and a yellow line connects the new vertex to the previous vertices. As you move clockwise around the object, stop selecting vertices immediately before you are about to close the shape (that is, before clicking on the first vertex you selected when starting to delineate the object).
- Go to the Menu window, and click the right mouse button on the Attrib pulldown menu.
- Scroll to the Name option. The Input window (at the bottom of your screen) prompts you for a name.
- Enter a unique identifier for the object you have just delineated. Do not use spaces. The becomes the object’s gfx keyword name. For example, in *population.usa.mapviz* the gfx column is specified as the first column in the data file. This first column contains strings such as “CA” and “NY.” These are the keyword names for the states. These keyword names are the gfx keyword names in the associated gfx file.
- Go to the Menu window, and click the right mouse button on *Done*.

7. Repeat Step 6 for every other object in the same image. If the object adjoins a previously identified object, you must reuse common vertices by selecting them with the middle mouse button instead of the left mouse button. Using the middle mouse button while the crosshairs are positioned close to a previously selected vertex ensures that the newly selected vertex is identical to the previously selected one.

   Caution: If a graphical object is too large to fit into the Front window, you must identify the vertices in sections. After all the objects are declared and the vertex information written to an ASCII file, you must edit this output file to join the sections of each subdivided object.

8. When all objects are identified, save the recorded vertices in a file. To do this:

   - Go to the Menu window and press the right mouse button on the File pulldown menu.
   - Scroll down to the File i3dm format option and choose it. The Input window (at the bottom of your screen) prompts you for a filename.
   - Enter a filename, specifying the *.i3dm* suffix.

9. Exit the i3dm application. To do this

   - Go to the Menu window, and choose the *File* pulldown menu.
   - Scroll to the Exit option, and choose it.

10. Convert the i3dm format file into a gfx file format by using the convert.i3dm utility, using the following syntax:

    ```
    /usr/lib/MineSet/mapviz/convert.i3dm inputFilename
    outputFilename.gfx
    ```

    For each object, the utility prompts you to

    - confirm the object's keyword name (which defaults to the Attrib name you supplied in Step 6, substep 6, above, when identifying the vertices)
    - declare the object's full name (which is the name the user sees in the Map Visualizer's Selection window when using the mouse to select a geographical object)
    - declare if the object has a concave shape that requires special handling

    Note: Declaring an object to be concave results in an accurate graphical display, but at the cost of slower performance. One strategy is to declare no objects as concave, examine the display to determine which objects are inaccurately drawn, then manually edit the gfx files for those objects, changing the string "convex" to "concave." Another strategy is to declare all objects as "concave" (assuming there are few objects), then determine if the resulting performance is acceptable.

Appendix C

Creating Data and Configuration Files for the Scatter Visualizer

The first part of this appendix describes the types and formats of data supported by the Scatter Visualizer. Data input to the Scatter Visualizer must be provided as a single file containing raw data, usually in a tab-separated ASCII text form.

The second part discusses the configuration file, which describes how the Scatter Visualizer reads in, and displays, the data file.

Both the data and configuration files can be generated automatically by the Tool Manager (see Chapter 3).

Note: Read Chapter 6, “Using the Scatter Visualizer,” before using this appendix.

The Data File

In its simplest form, the data file consists of a list of lines, each containing a set of fields separated by one tab. (Other separators are also allowed, but only one can separate each field. See “Input Options” on page 419.) All lines must contain the same fields. The interpretation of the fields is specified by the configuration file, described in the next section. Using the store sales data provided as part of the Scatter Visualizer package (file */usr/lib/MineSet/scatterviz/examples/store-type.data*), the first few lines of the input file appear as:

```
LIQUOR STORE    4300,4460,4800,4900,4700,4200,4250,4200
2700,2800,2750,3000,2900,2600,2500,2650
1600,1650,1900,1950,2000,2200,2300,2300
GROCERY STORE    700,900,600,800,877,755,800,600
3000,2900,3100,2800,2899,2950,3400,3300
10000,11000,9000,9800,9700,9650,9770,9700
```

In this sample file listing, each line consists of four fields, separated by tabs. The first field is a string that identifies a store type. The second field is an array of eight numbers, separated by commas, which might be sales of alcohol over an eight-day period. The third and fourth fields are also arrays of eight numbers that could represent sales of tobacco and food, respectively, over the same eight-day period.

The sample data file has other fields in the same format, but these are not shown. These additional fields correspond to sales of other products (see the configuration file */usr/lib/MineSet/scatterviz/examples/store-type.scatterviz* for a listing of all the fields).

The data file cannot contain blank lines or comments. Missing or extra data on a line causes an error.

Note: One tab (the default separator) separates each field. Do not insert multiple tabs to line up the fields visually; this generates blank fields. The order of the fields must match the format specified by the configuration file.

Data Types

The Scatter Visualizer supports integer, floating-point number, and string data types, as well as arrays of these types. The following five data types are supported:

- **int** represents a 32-bit signed integer.
- **float** represents a single-precision floating point number. The decimal point is optional. Numbers in exponential "e" notation are also accepted.
- **double** represents a double-precision floating point number. The decimal point is optional when representing a floating point number. Numbers in exponential "e" notation are also accepted. The superior precision of **double** can be useful for accurately representing large numbers, since **float** can represent only seven or eight significant digits accurately. This superior accuracy, however, consumes twice the memory space of **float**.

- **dataString** represents a string that is unlikely to appear multiple times. If it appears multiple times, several copies are made. A **dataString** is typically used to store an address. Addresses are unlikely to be compared, and each record can have a different address.
- **string** represents a string of characters that can appear multiple times in the data file. Unlike a **dataString**, only a single copy of a given string is stored in memory, no matter how many times it appears in the data. This saves much memory for strings appearing many times.

  Comparing **strings** is also much quicker than comparing **dataString**s. Processing is somewhat slower when looking for duplicate strings as they are read in. An example of **string** use is for a division name that appears once for each department in the division. If you are unsure whether to use a **string** or a **dataString**, use a **string**.

Arrays

With the Scatter Visualizer, you can use fields that are one- or two-dimensional arrays of fixed size. In a fixed-sized array field, all entries of the given field are arrays with the same number of values. Arrays contain the data values across one or two independent variables (those dimensions controlled by the sliders). In the listing from the file *store-type.data*, the second, third, and fourth fields are arrays.

Null Values

Any field or array element in the data file can also have the value “?” (question mark), indicating an unknown or null value (see the discussion of nulls in Appendix G).

The Configuration File

The configuration file format is flexible. Words in it must be separated by spaces, and it is case-sensitive. Except for the include statement and text within quoted strings, spacing and line breaks are irrelevant.

Sections

The configuration file consists of a series of sections, each of which has the form:

sectionKeyword

```
{
statements...
}
```

where *sectionKeyword* names the section. The order of the sections is significant, since sections can refer to variables defined in previous sections.

Defaults Files

As each section is encountered, a special configuration file (referred to as a *defaults file*) is also read in. Defaults files normally contain options statements. These files are searched in the following order:

1. The directory */usr/lib/MineSet/scatterviz*. This directory usually contains system defaults.
2. The *~/.MineSet* directory (where the tilde, ~, indicates your home directory). You can set up personal defaults in this directory.
3. The current directory. This lets you set up defaults for each directory.

Files with the same name can appear in more than one of the above-named directories; in this case, the order given is the one in which the directories are read. If the same option is found in multiple files, the last option read is used. Note that the appropriate section in the configuration file is read after all the defaults files; thus, options in the configuration file override those in the defaults files.

Statements

A statement has the following form:

```
statementKeyword info ;
```

where *statementKeyword* defines the statement, and *info* varies according to the keyword.

Variable Names

A variable name can appear in two formats:

- In the first format, it is a letter followed by a number of letters, digits, or underscores. It cannot be a keyword, and should not be quoted.
- In the alternate form, the variable name should be surrounded by back quotes (‘). In this form, the variable name can match a keyword, and can contain even non-alphanumeric characters. The primary purpose of this second form is for configuration files generated automatically by the Tool Manager.

There is no scoping of variable names; a given variable name can only be declared once in the configuration file.

Options Statements

Many sections have options statements, which have the form

```
options optionName info, optionName info... ;
```

where *optionName* defines the specific option, and *info* depends on the option. In some cases, *optionName* can be more than one word. To maximize the number of allowable variable names, most option names are meaningful only within the appropriate options statement; option names do not conflict with variable names. You can declare several options on the same line, separating them by commas or placing them in several options statements. If two conflicting values for the same option appear, the last value is taken.

Include Statements

The configuration file can contain lines of the form

```
include "filename"
```

These lines can appear anywhere in the configuration file, but each must be on its own line. The filename must be in quotation marks; anything after the closing quote is ignored. The number of nested includes is unlimited. If a relative pathname (one not beginning with a slash) is specified, the file is first sought in the directory containing the current configuration file. If include statements are present, this might not be the same as the initially loaded configuration file. If it is not found in the current configuration file, the include is sought in the current directory.

Sinclude Statements

A statement similar to an include is sinclude, which has the form

```
sinclude "filename"
```

This is identical to the include statement, except that no error is given if the file does not exist; instead, the sinclude statement is ignored.

Strings and Characters

Strings and characters in the configuration file follow C conventions. Strings are in double quotation marks ("), and characters are in single quotation marks ('). All standard backslash conventions are followed (for example, \n represents a new line).

Comments

Comments begin with a pound (#) symbol at the beginning of a line; anything after this symbol to the end of the line (80 characters) is ignored, up to the end of the line.

Keywords

The keywords recognized by the Scatter Visualizer are listed in Table C-1. Variables cannot have these names unless they are surrounded by back quotes (‘). Tokens appearing only in option statements are not keywords, and can be used for variable names.

Table C-1 Scatter Visualizer Keywords

across	average	axis	buckets
by	color	colors	dataString
date	divide	double	entity
execute	expressions	file	float
from	include	input	int
key	label	legend	max
message	min	modulus	monitor
off	on	options	scale
separator	sinclude	size	slider
string	sum	summary	time
to	view		

Currently, the keywords **execute**, **min**, **monitor**, and **time** are not used by the Scatter Visualizer.

Expressions

Expressions are accepted in several places in the input. Expressions follow the syntax of C. The following operations are supported:

```
+  -  *  /  %  ==  !=  >  <  >=  <=  &&  ||  !  &  |  ^  ?:
```

Also, the following functions are available:

- **divide**(x, y, z) divides *x* by *y*, unless *y* is zero. If *y* is zero, the result is *z*; this is equivalent to y==0 ? z : x / y.
- **modulus**(x, y, z) is similar to **divide**, but for modulus.

Type handling is similar to that in C. Expressions using **int** and **float** promote both sides to float. Expressions using **int** and **double**, or **float** and **double** promote both sides to double. The result of a relational expression (for example, ==, <) is always an **int**. Type casting is also supported.

Unlike in C, strings can be compared using relational expressions; the strings are compared lexicographically.

The Input Section

The first section of a configuration file is normally the input section. It defines the name and format of the data file. A typical input section might look like this:

```
input  {
   file "company.data";
   string company;
   slider int income from 20000 to 60000 by 10000;
   slider date “%N %Y” purchaseDate from “Jan 1990” to “Dec
   1992” by “1 month”;
   options array separator ‘,’;
   float lifeSales[income][purchaseDate];
   float autoSales[income][purchaseDate];
   float homeSales[income][purchaseDate];
   string location;
   }
```

This example states that the input file is called *company.data*, and that there are five fields: *company*, *lifeSales*, *autoSales*, *homeSales*, and *location*. The *company* and *location* fields are of type **string**, while the other three fields are two-dimensional arrays of type **float**. Two slider dimensions are declared:

- *income*, which is of type **int**, ranges from 20000 to 60000 in increments of 10000; and
- *purchaseDate*, which is of type **date** and ranges from January 1990 to December 1992 in increments of 1 month.

The arrays *lifeSales*, *autoSales*, and *homeSales* contain values for each income and purchase date. Individual values within the arrays are separated by commas.

When the **input** section is entered, the defaults file *inputDefaults* is read in.

File Statements

The **file** statement names the data file to be read. This statement is required. Its form is:

```
file "filename";
```

filename must be in double quotation marks. If it is a relative pathname (no leading slash), it is first sought in the directory containing the current configuration file. If include statements are present, this might not be the same as the initially loaded configuration file. If it is not found in the current configuration file’s directory, the file is sought in the current directory.

Enumeration Statements

Enumeration statements declare enumerations, or enums, that index into array fields. The enum statement has three forms.

- The first enum statement form is

  ```
  enum type name from value1 to value2 by increment;
  ```

  This declares an enum with values starting at *value1* and incremented by *increment* until they reach or exceed *value2*. For example, the statement

  ```
  enum int age from 20 to 70 by 10;
  ```

  declares age as an enum with the values 20, 30, 40, 50, 60, and 70.

  Type must be a number type (**int**, **float**, or **double**) or **date** (see “Dates” on page 415).

- The second enum statement form is

  ```
  enum type name from value1 to value2 across
  numberOfValues;
  ```

  This declares an enum with values ranging from *value1* to *value2*. The *numberOfValues* is an integer specifying the number of values. For example, the statement:

  ```
  enum int age from 20 to 70 across 6;
  ```

  declares age as an enum with the values 20, 30, 40, 50, 60, and 70.

  Type must be a number type (**int**, **float**, or **double**) or **date** (see “Dates” on page 415).

- The third enum statement explicitly lists the enum values. Its form is:

  ```
  enum type name { value1, value2, ..., valueN };
  ```

  Type can be any type or date (see “Dates” on page 415).

Dates

The enum statement includes special support for a date type that handles date and time values starting Jan 1, 1753. The date type is valid only within enum statements. A date enum statement can have the following syntaxes:

```
enum date “format” name from “value1” to “value2” across
        numberOfValues;
enum date “format” name { value1, value2, ..., valueN };
enum date “format” name from “value1” to “value2” by
        “increment”;
```

The *format* string specifies the format of the values; it is useful for controlling how dates are displayed in the animation control panel. The syntax of the *format* string is similar to the **scanf** function in C. Various units of time are represented by special characters preceded by the percent symbol (%). For example,

```
enum date cq “Calendar Q%Q, %Y” from “Calendar Q1, 1980” to
“Calendar Q3, 1985” by “1 quarter”;
```

The “Calendar Q” in the *format* string matches the “Calendar Q” in *value1* and *value2*. The %Q in the *format* string indicates that the next number in *value1* and *value2* is the calendar quarter. The comma and space in the *format* string match the commas and spaces in the values. Finally, the %Y in the *format* string specifies that the year values are next.

Table C-2 lists the characters that can follow the percent symbol and the units of time they represent.

Table C-2 Characters That Can Follow the percent Symbol in the format String

Character	Time Unit	Precision
Y	year	4
Q	calendar quarter	1
M	month	2
N	month name	>= 3
D	day	2
h	hour	2
m	minute	2
s	second	2

With the exception of N, each character matches an integer of the specified precision. N matches 3 or more characters giving the English name of the month.

The from-to-by form of the enum statement includes an increment value. For dates, the increment is a quoted string containing an integer, an optional space, and one of the special characters in Table C-2 or one of the symbols **year**, **quarter**, **month**, **day**, **hour**, **minute**, and **second**. The plural forms of these symbols are also accepted. Note that these symbols are not keywords, since they have special meaning only in the increment string. The following are examples of valid increments:

```
“1 year”
“7 days”
“4h”
```

Data Statements

The data statements declare the fields in the data file. The fields must be declared in the order they appear in the data file. The format of most data statements is

```
type name;
```

where *type* is **int**, **float**, **double, string**, or **dataString**; *name* is the variable name.

A data field can also be based on an enumeration. The syntax is

```
enum enumName name;
```

The field must contain **ints** corresponding to the values of the enum. For example, if the enum ageGroup is declared as

```
enum string ageGroup {"below 30", "30-39", "40-49", "50-59",
        "60 or above"};
```

the field age can be declared as

```
enum ageGroup age;
```

The field should contain **ints** between 0 and 4, where 0 is displayed as “below 30,” 1 as “30-39”, and so forth.

Only one variable can be declared per statement.

Arrays

Arrays are also declared using data declarations. The declaration syntax for one-dimensional arrays is one of the following:

```
type name [ number ] ;
type name [ enumName ] ;
type name [ null enumName ] ;
```

For example:

```
float revenue [50];
```

The declaration syntax for two-dimensional arrays is one of the following:

```
type name [ number1 ][ number2 ] ;
type name [ enumName1 ][ enumName2 ] ;
type name [ null enumName1 ][ null enumName2 ] ;
```

For example:

```
float revenue [50][10];
```

When enums are used, the number of values in the array is taken from the declaration of the enum. For example, given the statements

```
enum int age from 20 to 70 by 10;
float clothingPurchases[age];
```

the array *clothingPurchases* must have six values, corresponding to the enum values 20, 30, 40, 50, 60, and 70.

The keyword **null** indicates an extra value at the beginning of the array, corresponding to null. Thus, the statements

```
enum int age from 20 to 70 by 10;
float clothingPurchases[null age];
```

declare *clothingPurchases* as an array with seven values: the first value corresponding to null or unknown age values, and the remaining six values corresponding to age values 20, 30, 40, 50, 60, and 70.

You can override the separator between values in an array by declaring it as:

```
type name [ number ] separator ‘char’;
```

For example:

```
float revenue [50][10] separator ‘:’;
```

If no separator is specified, the default separator (usually a tab) is used.

Input Options

All **options** statements begin with the word “options” and have one or more comma-separated options.

- The separator option defines the separator between fields in the data file. The default separator is a tab. The syntax is

  ```
  options separator ‘char’;
  ```

  For example:

  ```
  options separator ‘:’;
  ```

  Note: The separator is used also to separate values within arrays; however, arrays can override the separator.
- The backslash option controls whether backslashes in the input data are treated specially or like other characters. The syntax is:

  ```
  options backslash off;
  ```

  ```
  options backslash on;
  ```

  The default is off. If backslash processing is on, separators in the input data preceded by backslashes are treated as regular characters rather than separators. Within strings, this causes standard C-style backslash processing.

The Expressions Section

The **expressions** section of a configuration file lets you define additional fields that are expressions of existing fields. For example, one field can be defined as the sum of two other fields.

The format of the expressions section is

```
expressions
{
expressionDeclaration;
...
}
```

where *expressionDeclaration* has the following form:

```
type name = expression ;
```

The following is a sample expression section. This section assumes two existing array fields of type **double**: “male” and “female”; these represent spending by males and females on various goods across time (one independent dimension). Two fields are added: “total” represents the total dollars spent, and “pctFemale” represents the percentage of dollars spent by females.

```
expressions
{
double total[36] = male+female;
double pctFemale[36] = divide (female*100, total, 50.0);
}
```

Note: The pctFemale calculation uses “total,” defined in the previous statement. Also, note the use of the divide function rather than the / operator. This results in 50% for the case where there are no dollars spent at all; using the / operator generates a divide by zero error.

The expressions section has no options; thus, no defaults file is read in for it.

The View Section

The **view** section of a configuration file describes how the data is displayed, including the mapping of sizes, colors, axes, and so on. The default values for these options are in */usr/lib/MineSet/scatterviz/view.scatterviz.options*. Its form is

```
view
{
viewStatement;
...
}
```

A sample view section is

```
view {
   slider month;
   entity brand;
   axis male$, color “blue”;
   axis female$, color “red”;
   size total$, max 5;
   color pctFemale, scale 0 50 100, colors “blue” "gray"
   "red";
   message "brand %s, total sales %,.0f",brand, total$;
   }
```

When entering the view section, the *viewDefaults* file is read in.

Slider Statement

The **slider** statement identifies an enum to be used as a slider dimension. Its syntax is one of the following:

```
slider enumName;
slider null enumName;
```

The enum name is declared in the input section. If the keyword **null** is present, the slider includes a position at the beginning corresponding to null or unknown values of the enum. Arrays indexed by the slider must be declared to match the *null* in the slider statement.

There can be 0, 1, or 2 slider statements. The first slider statement applies to the horizontal slider, the second to the vertical slider. If there is no slider statement, the resulting display does not include animation.

Entity Statement

The **entity** statement lets you specify a variable that uniquely identifies the entities in the display. The entity statement consists of a series of clauses, separated by commas:

```
entity clause1, clause2,...
```

Alternatively, the clauses can be given in separate entity statements.

The Entity Variable

The first clause of the **entity** statement normally contains the name of the entity variable (*brand* in the example on page 421).

The Label Clause

This clause defines how the entities are labeled. It has the following forms:

- label off

  This turns off the labels.

- label on

  This turns on the labels. The default labels use the entity variable as the label for each entity.

- label *variable*

  This turns on the labels and uses the given variable to label the entities. When this form is used, it is not necessary to specify *label on*.

The Label Color Clause

This clause turns on the labels and specifies their color. It has the form:

```
label color “colorname”
```

where *colorname* is the name of a color in a special format. (Color naming is explained in “Color Statement” on page 425.) The default label color is gray.

The Legend Clause

The legend clause explains what the entities are. Any string can be placed in the entity legend. The legend clause has the following forms:

- legend off

  This turns off the entity legend.

- legend on

  This turns on the entity legend (this is the default). The default legend is

  ```
  Entity: varname
  ```

  where *varname* is the name of the entity variable.

- legend label “*string*”

  This turns the legend on and explicitly sets the legend string. If this form is used, *legend on* is unnecessary.

Size Statement

The **size** statement describes how a field of data is mapped to the sizes of entities. The size statement consists of a series of clauses, separated by commas:

```
size clause1, clause2,...
```

Alternatively, the clauses can be given in separate size statements.

The Size Variable

The first clause normally contains the name of a field to be mapped to size (*total$*, in the **view** example on page 421). The field must be of a number type (**int**, **float**, or **double**), of which **float** is the most efficient. The field can be an array that is indexed by slider dimensions. If no size field is specified, all entities are the same size.

The Max Clause

Normally, the size variable is mapped to the size of the entities, so that the biggest entity has a size of 5. This size can be changed by specifying a different value. If there is no size variable, the default maximum size is 2.5. The **max** clause has the form

```
max float
```

The Scale Clause

Instead of using the **max** clause to affect size values, the **scale** clause can be used to scale these values; all values are multiplied by the scale. The scale clause’s syntax is

```
scale float
```

The Legend Clause

The **legend** clause defines the meaning of the size mappings. Any string can be placed in the size legend. The legend clause has the following forms:

- `legend off`

  This turns off the size legend.

- `legend on`

  This turns on the size legend (this is the default). The default legend is:

  `size:`*`varname`*

  where *varname* is the name of the variable that is mapped to size.

- `legend label` "*string*"

  This turns the legend on and explicitly sets the legend string. If this form is used, *legend on* is unnecessary.

Color Statement

The **color** statement describes how values are mapped to colors. The format is similar to the size statement, consisting of several clauses that can be separated by commas, or entered as multiple statements. The syntax is:

```
color clause1, clause2,...
```

Color Naming

Color names follow the conventions of the X window system, except that the names must be in quotes. Examples of valid colors are "green," "Hot Pink," and "#77ff42." The latter is in the form "*#rrggbb*", in which the red, green, and blue components of the color are specified in hexadecimal value. Pure saturation is represented by ff, a lack of color by 00. For example,"#000000" is black, "#ffffff" is white, "#ff0000" is red, and "#00ffff" is cyan.

The Color Variable

As with size, you also can specify a single field to be mapped to an entity color. The field can be an array that is indexed by slider dimensions. If the field is an array, it must be a number type. If the field is a number type, the scale and buckets clauses described below can be used to map a range of colors to the values of the field. If the field is not a number type, it is sorted, and each unique value is assigned a color.

The colors Clause

The **colors** clause specifies the colors to be used. The colors clause's syntax is:

```
colors "colorname" "colorname"...
```

The format for *colorname* is described in "Color Naming" on page 425. Note that there are no commas between the colors, because commas are used to separate clauses in the color statement. A sample colors clause is:

```
colors "red" "gray" "blue"
```

Colors in the list are subsequently referred to by their index, starting at zero. In the above example, red is color 0, gray is color 1, and blue is color 2.

If there is no colors statement, colors are chosen randomly. If there is a colors statement, at least as many colors must be specified as are to be mapped.

The scale Clause

The **scale** clause allows assignment of values to a continuous range of colors. For example, when displaying a percentage, red can be assigned to 0%, gray to 50%, and blue to 100%. Intermediate values are interpolated; for example 25% is pinkish, and 55% is a slightly bluish gray.

The syntax for the scale clause is

```
scale float float ...
```

The first value is mapped to color 0, the second to color 1, and so forth. The colors statement must contain at least as many colors as are to be mapped to the largest index.

Values in this statement must be in increasing order. Any value less than the first color is assigned the value of the first color. Any value greater than the last value is assigned the last color. Intermediate values are interpolated.

For example, assume the pctFemale field indicates what percentage of the group is female, and you want to map a group that is 100% female to red, 100% male to blue, and 50% each to gray. The colors statement for this is:

```
colors pctFemale, colors "blue" "gray" "red", scale 0 50 100;
```

Use the scale clause only in conjunction with a numeric color variable.

The buckets Clause

The **buckets** clause is similar to the scale clause without interpolation. All values are rounded down to the highest value in the clause, and that exact color is used. Values less than the first value use the first color.

The syntax for the buckets clause is

```
buckets float float ...
```

The syntax and assignment of colors is the same as for the scale clause.

If, in the above example, you used the buckets clause instead of the scale clause, the statement would be:

```
colors pctFemale, colors "blue" "gray" "red", buckets 0 50 100;
```

All values greater than or equal to 100 are colored red. Values greater than, or equal to, 50 but less than 100, are gray. All other values are then blue.

Use the buckets clause only with a numeric color variable.

The legend Clause

The **legend** clause creates a legend of the colors. The legend clause syntax can be any of the following:

```
legend off
legend on
legend "string" "string" ...
legend label "string"
```

The **legend off** clause turns the legend off. The **legend on** clause turns the legend on. It can be omitted if other legend statements are included. Specifying only **legend on** generates the default legend.

The default legend includes a single label to the left (with the name of the field that is mapped to color), and a list of colored labels on the right (with values obtained from the scale clause, the buckets clause, or from the field). To override the strings in the colored labels, specify the strings as:

```
legend "string" "string"
```

To override the label on the left, specify it following the word *label*. To eliminate this label, specify an empty string; that is:

```
legend label ""
```

Axis Statement

The **axis** statement causes a variable to be used as an axis in the 3D landscape. The variable’s values determine where the entities are positioned on the axis. There can be up to three axis statements. Like the size and color statements, the axis statement contains a series of comma-separated clauses, but all of them must be specified in a single statement.

```
axis clause1, clause2,...
```

The Axis Variable

As with size and color, you can specify a field to be used as an axis. The field can be an array that is indexed by slider dimensions. If the field is an array, it must be of type **number**. If the field is not of type number, it is sorted, and each unique value is assigned a position along the axis.

The Label Clause

The **label** clause has the form:

```
label "string"
```

The string is used to label the axis. It appears in the landscape, at the end of the axis line. The default label is the name of the axis variable.

The Max Clause

Normally, the axis variable is mapped directly to the position of the entities along the axis0. The **max** clause lets you normalize the values of the axis variable, so that the maximum value is mapped to the specified max. The max clause's syntax is:

```
max float
```

The Scale Clause

Instead of using the **max** clause to affect position values, the **scale** clause can be used to scale the values. All values are multiplied by the scale. The scale clause syntax is

```
scale float
```

The Color Clause

The **color** clause specifies the color used for the axis line and label. It has the form:

```
color "colorname"
```

The Extend Clause

The **extend** clause specifies whether the axis should be extended automatically to include the value zero. It has the form:

```
extend on
extend off
```

Summary Statement

The **summary** statement specifies a summation to be calculated over all the entities. The summary is used to color the drawing window in the animation control panel. Like the size and color statements, the summary statement has several clauses that can be specified in one statement, separated by commas, or in separate statements.

```
summary clause1, clause2,...
```

The Summary Variable

You can specify the variable to be used in the summary. This variable must be of number type. Typically, the summary variable is an array indexed by slider dimensions, so that the summary value varies across the slider dimensions.

The Color Clause

The **color** clause specifies the color used to display the summary values in the drawing window. It has the form

```
color “colorname”
```

Various shades of the color, from white to the specified color, are used to represent summary values. The minimum summary value is mapped to white, while the maximum summary value is mapped to the specified color. The default summary color is red.

The Legend Clause

The **legend** clause creates a legend of the summary colors. The legend clause syntax can be any of the following:

```
legend off
legend on
legend label "string"
```

The **legend off** clause turns the legend off. The **legend on** clause turns the legend on. It can be omitted if other legend statements are included. Specifying only **legend on** generates the default legend.

The legend includes a single label to the left (which defaults to the aggregation function and variable used in the summary), and two colored labels on the right (with the minimum and maximum summary values). To override the label on the left, specify it following the word *label*. To eliminate this label, specify an empty string; that is

```
legend label ""
```

Message Statement

The **message** statement specifies the message displayed when an entity is selected. The syntax is similar to that of the C **printf** statement. A sample message statement is

```
message "%s: $%f, %.0f%% of target, %.0f%% of last year",
         product, sales, pctTarget, pctLastYear;
```

This could produce the following message:

```
furniture: $2425.37, 23% of target, 87% of last year
```

The formats must match the type of data being used:

- Strings must use %.
- Ints must use integer formats (such as %d.
- Floats and doubles must use floating point formats (such as %f).

For details of the **printf** format, see the printf (1) reference (man) page (type **`man printf`** at the shell prompt).

A special format type has been added to **printf**. If the percent sign is followed by a comma (for example, “%,f”), commas are inserted in the number for clarity. Only the United States convention of d,ddd,ddd.dddd is supported, with the decimal point represented by a period, and commas separating every three places to the left of the decimal point. For example, if the above format were:

```
message "%s: $%,f, %,.0f%% of target, %,.0f%% of last year",
         product, sales, pctTarget, pctLastYear;
```

it would produce the message:

```
furniture: $2,425.37, 23% of target, 87% of last year
```

The $, *, h, l, ll, L, and n **printf** format options are not supported.

All values, including the format string, are expressions. Thus, if you had a pctFemale field, but wanted a more gender-neutral message, you could use:

```
message pctFemale>50?"%f%% females":"%f%% males",
         pctFemale>50?pctFemale:100-pctFemale;
```

If pctFemale is 70, the message “70% females” is displayed; if pctFemale is 30, the message “70% males” is displayed. In this case, you can also achieve the same result with a single format string:

```
message "%f%% %s", pctFemale>50?pctFemale:100-pctFemale,
         pctFemale>50?"females":"males";
```

If no message is specified, a default message containing the names and values of all the fields is used.

The Filter Statement

The **filter** statement specifies that only entities meeting certain filter criteria should be displayed initially (see “The Filter Menu” in Chapter 6). The filter criteria are in the form of expressions whose values must all be true or nonzero for an entity to be displayed (expressions are described in “Expressions” on page 412).

The syntax of the **filter** statement is

filter *expression*, *expression*, ...

For example, the statement

```
filter state == "CA" || state == "WA", sales > 9000, pctTarget >= 90;
```

specifies that only records from California or Washington state, with sales greater than 9000 and a *pctTarget* value greater than or equal to 90 should be displayed initially.

After the Scatter Visualizer is invoked, the filter criteria can be changed or removed interactively using the filter panel.

View Options

The **view** section of the configuration file has several options for controlling parameters of the display. These options can appear in a single options statement, separated by commas, or in separate options statements. The syntax of the options statement is

```
options option, option,...
```

The following options are available:

- `entity label size` *`float`*

  controls the size of the entity labels.

- `axis label size` *`float`*

  controls the size of the axis labels.

- `hide entity label distance` *`float`*

  controls the distance at which entity labels become invisible. Smaller distances might improve performance, but the labels disappear more quickly.

- `grid color` *`“colorname”`*

  controls the color of the grid.

- `grid size` *`float float float`*

  controls the spacing between grid lines. It applies the three values to grid lines along the x, y, and z axes, respectively.

- `entity shape` *`shapeName`*

  specifies the shape used to display entities. *shapeName* can be “cube,” “bar,” or “diamond.”

Appendix D

Creating Data and Configuration Files for the Rules Visualizer

This appendix describes

- data and configuration files
- command-line operation
- example files and commands

for each of the three components of the Rules tool (association data converter, association rules generator, and rule visualizer).

The Rules tool is completely operable via the Tool Manager (see Chapter 3). Alternatively, all components of the Rules tool can be invoked via the command-line interface and / or files created with a text editor, such as jot, vi, or Emacs. This second mode of operation lets you create configuration files needed for the association data converter and the association rules generator. It also lets you set up a process (using standard UNIX facilities) to run the association data converter and the association rules program nightly on new data using those configuration files.

The examples used in the following sections can be found in the */usr/lib/MineSet/assoccvt/examples/* and */usr/lib/MineSet/assocgen/examples/* directories. Descriptions and instructions for use can be found in the README file in these directories.

Note: Read Chapter 7, “Using the Rules Visualizer,” before using this appendix.

The Association Data Converter

The association data converter converts a raw data file (such as a user's ASCII data file) into a file of the format used by the association rule generator program. The association data converter requires as input a raw data file and a format file. Its output is a specially formatted data file for use by the association rules generator. Note that the process described below is for preparing data files for use by the associations program manually (that is, via the command line). When the associations program is run via the Tool Manager, this process is done automatically.

In the following description, %s denotes a string-valued input, %f denotes a floating-point number input, and %d denotes an integer number input.

Association Data Converter File Requirements

The association data converter requires:

- a *raw data file* (this is the user's data for running associations) in one of two accepted formats.
- a *format file*, which describes the raw data file's format.

The Raw Data File

The raw data file input from the association data converter can be in one of two formats:

- Single-item format
  - Each record has one item and an identifier.
  - All items with the same identifier are grouped on successive lines in the file (they need not be sorted, just grouped.)
  - Each record has the same length.
- Multiple-item format
  - Each record has multiple items and an identifier.
  - All items associated with the same identifier are in a single record.
  - Each record has the same length.

The Format File

The format file specifies the format of the raw data file to the association data converter.

The format file follows one of the forms listed in Table D-1 or Table D-2, depending on the raw data file's format (single or multiple item).

Table D-1 Single-Item Format

List of required items in the format file (Format 1)
Letter "S" to indicate single-item format file
Number of bytes in each record (excluding record separator, such as LF)
Number of fields that make up the identifier
Total number of bytes in the identifier
Offset and length in bytes for each field that makes up the identifier
Number of fields that make up the item
Total number of bytes in the item
Offset and length in bytes for each field that makes up the item
Description flag indicating if descriptions should be produced along with names (either a 0 [meaning No] or 1 [meaning Yes])
If the description flag is 1, the following are required too:
Number of fields that make up the description
Total number of bytes in the description
Offset and length in bytes for each field that makes up the description

Table D-2 Multiple-Item Format

List of required items in the format file (Format 2)
The letter “M” to indicate multiple-item format file
Number of bytes in each record (excluding record separator such as LF)
Number of items in each record
For each item:
Name of the item (column name / domain)
Number of fields of the record that make up the item
Total number of bytes in the item
Number of buckets (discrete bins or categories); 0 for categorical items
Offset and length in bytes for each field that makes up item

Files Generated by the Association Data Converter

The association data converter generates two files:

- The *output data file* contains the converted data from the raw data file in a format required by the association rules generator.
- The *output names file* contains auxiliary descriptor information used by the association rules generator.

The Association Data Converter Command-line Operation

Table D-3 lists the set of options for controlling the association data converter. A description of each option follows the table. An example of invoking the program is:

```
assoccvt -ifile sing.data -ofile sing.bin sing.format sing.names
```

Options for controlling data conversion from raw to internal format are listed below. A description of each option follows the table.

Table D-3 Options for the Association Data Converter

Option Format	Required	Default Value	Comments
-ifile %s	no	stdin	Name of raw data file (input)
-ofile %s	no	stdout	Name of output data file
-isize %d	no	4	Size of binary numbers in output file

-ifile %s
Specifies the name of the raw data file, which serves as input to the association data converter. This file contains the data to be converted.

-ofile %s
Specifies the name of the file that contains the data converted by the association data converter.

-isize %d
Specifies the binary integer size in the output data file.

There are two required arguments on the association data converter command line:

- The name of the format file to be used by the association data converter
- The name of the file containing the description of the integer codes in the output data file

Association Data Converter Examples

The following commands illustrate the use of the association data converter on the example files in */usr/lib/MineSet/assoccvt/examples*. The file *sing.data* is an example of data in the single item format and has some simple grocery store transactions. Each line has a transaction number and the name of an item bought in that transaction. The format of this file is described by *sing.format*. The file *mult.data* is an example about automobiles and has data about cars of different origin (American, Japanese, European) regarding attributes such as MPG, weight, etc. The values for these attributes are in discrete ranges rather than exact numbers. The format of this file is described by the file *mult.format*.

```
assoccvt -ifile sing.data -ofile sing.bin sing.format sing.names
assoccvt -ifile mult.data -ofile mult.bin mult.format mult.names
```

To test whether the files for data conversion are correctly installed, run any or all of the following commands from the shell command line. Then, using the UNIX *diff* command, compare the files created to those with the same name in */usr/lib/MineSet/assoccvt/examples*.

Enter:

```
assoccvt -ifile sing.data -ofile sing.bin sing.format sing.names
```

Then compare *sing.bin* with */usr/lib/MineSet/assoccvt/examples/sing.bin*, and compare *sing.names* with */usr/lib/MineSet/assoccvt/examples/sing.names*.

Enter:

```
assoccvt -ifile mult.data -ofile mult.bin mult.format mult.names
```

Then compare *mult.bin* with */usr/lib/MineSet/assoccvt/examples/mult.bin*; and compare *mult.names* with */usr/lib/MineSet/assoccvt/examples/mult.names*.

Association Rules Generator

The association rules generator generates association rules among items in a set of data. Its required inputs are described in the following subsections. Its output is a specially formatted rules file, which can be used by the rule visualization part of the Rules Visualizer (see Chapter 7).

Association Rules Generator Files Requirements

The association rules generator programs, *assocgen* and *mapassocgen*, require:

- a *data file* in the internally required format
- a *configuration file*, which specifies various program parameters
- (for *mapassocgen* only) a *mapping file*, which specifies the mapping between hierarchical levels
- (for *mapassocgen* only) a *description file*, which specifies a string description for each item at a specific hierarchical level

Association Rules Generator Command-line Operation

Rules are generated by applying one of two commands, along with one or more parameters. The command used depends on whether the data for which rules are to be generated are non-hierarchical or hierarchical (see “Starting the Association Rules Generator Part” in Chapter 7). The commands are:

- *assocgen*—which generates rules based on nonhierarchical data.
- *mapassocgen*—which generates rules based on hierarchical data.

Numerous options control the rule-generation process. Many of these are common to both the *assocgen* and *mapassocgen* commands. Options fall into one of the following categories:

- *Rule Generation Options*— control the process of rule generation.
- *Rule Restriction Options*—place restrictions on the set of generated rules.
- *Hierarchical Data Options*—define parameters used only when generating rules from hierarchical data (using *mapassocgen*.)

The **-ropts** string separates the first two sets of options. This string is required if there are any options from the second or third set.

The **-vopts** string separates the second and third sets of options. This string is required if there are any options from the third set.

An example rule generation command line (for which the parameters are explained in the following sections) might be:

```
assocgen -prev 20 -tran mult.bin -ropts -names mult.names
        -rout -mult.rules
```

Rule Generation Options Common to assocgen and mapassocgen

Table D-4 lists the set of options for controlling the rule-generation process. A description of each option follows the table.

Table D-4 Options for Controlling Rule Generation

Option Format	Default Value	Comments
-tran %s	(stdin)	Data file path
-prev %f	(1.0)	Prevalence threshold (as a percentage)
-uniq %d		Number of items in dataset
-dir %s	(/usr/tmp)	Directory for temporary files
-tprefix %s	(A_)	Prefix for temporary files
-msg %s	(assocgen.msg)	Message file

-tran %s
Specifies the path for the file. By default, the file is read from *stdin*.

-prev %f
Specifies the minimum prevalence threshold as a percentage of the total number of records. The default is 1.0%. If the prevalence threshold results in a minimum count less than 3, an error message is displayed, and no rules are generated.

-uniq %d
Specifies the number of unique or distinct items across all records (if known). Specifying this (or an upper bound) speeds processing.

-dir %s
Specifies the directory to store temporary files, including the message file (see **-msg**, below). The default is *./* .

-tprefix %s
Specifies the prefix to be used for temporary files, except the message file (see **-msg**, below). The default prefix is *A_*.

-msg %s
Specifies the message file. The default is *assocgen.msg*.

Rule Restriction Options Common to assocgen and mapassocgen

Table D-5 lists the set of options for restricting generated rules. Options in this set are used after those listed in Table D-4 and separated on the command line from the former options by **-ropts**. A description of each option follows the table.

Table D-5 Options for Restricting Generated Rules

Option Format	Default Value	Comments
-pred %f	(50.0)	Minimum predictability (as a percentage)
-rnum	(FALSE)	Print only the number of rules generated
-rsort %d [%s]+	(4 RHS PRED PREV LHS)	Field sorting order. Fields can be all or any subset. PRED and PREV are sorted in descending order.
-names %s		Name of file containing item descriptions
-rout %s	(stdout)	Name of file in which to output rules

-pred %f
Specifies the minimum predictability threshold for rules. Rules with a predictability below this value are not generated. The default is 50%.

-rnum
Output only the number of rules generated, not the rules themselves.

-rsort %d [%s]+
Specifies the sort order for rules. The first number denotes the number of sorting fields specified; the second field specifies the fields. The four keys for sorting rules are (in order):

- RHS—items on right-hand side of rule
- PRED—predictability of rule
- PREV—prevalence of rule
- LHS—items on left-hand side of rule

-names %s
Specifies the name of the file which contains the descriptions of the items. This is typically the names file created during the *assoccvt* step.

-rout %s
Specifies the name of the file in which rules are to be written. If this is not specified, rules are written to *stdout*.

Hierarchical Data Options Common to assocgen and mapassocgen

There are no hierarchical data options common to both assocgen and mapassocgen. See “Hierarchical Data Options for mapassocgen Only” on page 447.

Rule-Generation Options for mapassocgen Only

In addition to the options provided by the *assocgen* program, the *mapassocgen* program provides two additional options (Table D-6). These options are rule-generation options and thus must be specified in the first set of options (the set before the **-ropts** string). A description of each option follows the table.

Table D-6 Options for the mapassocgen Command

Option Format	Comments
-agg %d	Hierarchical level for which rules are to be obtained.
-map %d [%d]+ %s	File for mapping lowest hierarchical level to level for which rules are to be obtained.

-agg %d
Specifies the level of the hierarchy at which the rules are desired. Level 0 is the lowest level of the hierarchy, level 1 is the next level up in the hierarchy, and so on.

-map %d [%d]+ %s
Specifies a file that allows the *mapassocgen* to map the lowest level in the hierarchy (present in the data) to the level at which the rules are desired. The first number specifies the total number of levels of aggregation. Next, for each level, a number denotes the size in bytes for the mapping value for that level. The lowest level is the implicit sequence 0, 1, 2, 3, and so on, and is not present in the map file. Finally, the path is given for the map file.

The values at the lowest level are the integers 0, 1, 2, 3, and so on. The map file lists the values at the new level (or levels) in the same order. First list the value(s) corresponding to level 1, then the value(s) corresponding to value 2, and so on. The values at the lowest level (level 0) are omitted because the list of values is in the implicit order 0, 1, 2, 3, and so forth.

Table D-7 provides an example from the dataset used in the “Hierarchical Data Example” section. This example has two hierarchical levels:

Table D-7 Example Hierarchy

Level 0	Level 1
Milk	Dairy
Chips	Snack
Coffee	Beverage
Eggs	Dairy
Tea	Beverage
Soda	Snack
Cheese	Dairy
Butter	Dairy

The first line in the mapping file (*/usr/lib/MineSet/assocgen/examples/synth.map*) indicates that the value “0” at the lowest level is mapped to value “1” at the next level; the second line indicates that the value “1” at the lowest level is mapped to value “0” at the next level; and so forth.

Rule Restriction Options for mapassocgen Only

There are no rule restriction options specific to *mapassocgen* only. See "Rule Restriction Options Common to assocgen and mapassocgen" on page 443.

Hierarchical Data Options for mapassocgen Only

The option used for hierarchical data is listed in Table D-8. A description follows the table.

Table D-8 Options Set 3

Option	Comments
-lvldesc %d %s	Hierarchical level, string description file

-lvldesc %d %s
Specifies the hierarchical level and the corresponding string description file. Each string description file must be on a separate line. The strings are mapped, in order, to items 0, 1, 2, 3, and so forth, at the hierarchy level.

This option succeeds all other options and must be separated from them by **-vopts**.

Association Rule Examples

Assume you have the data listed in Table D-9. This example is based on the data in the file */usr/lib/MineSet/assoccvt/examples/synth.data*. In this example, each row represents one transaction, and the transaction id is implicit.

Table D-9 Data Example 2

Item	Item	Item
Milk	Chips	Coffee
Milk	Chips	Eggs
Milk	Chips	Coffee
Chips	Coffee	Eggs
Milk	Coffee	Cheese
Milk	Eggs	Cheese
Milk	Tea	Butter
Eggs	Tea	Soda
Eggs	Tea	Soda
Eggs	Tea	Soda

After using *assocgen* on the file produced by running the sample data in Table D-9 through *assoccvt*, the rule file that is output has the following format:

```
1    1    30.0000  75.00   60.00 Item=Chips      Item=Milk
```

The first pair of numbers denote the number of items on the LHS and RHS of the rule, respectively. These are always 1's, since only a single item is supported for the LHS and the RHS. The next three numbers denote (in percentages) the prevalence, predictability, and expected predictability. Then the LHS item is listed, followed by the RHS item. In the example above, the LHS is item "Item=Chips", the RHS is item "Item=Milk."

The expected predictability is the frequency of occurrence of the RHS items. The difference between expected predictability and observed predictability is a measure of the increase in predictive power due to the presence of the LHS rule. Expected predictability gives an indication of what the predictability would be if there were no relationship between the items.

Nonhierarchical Data Example

Assume the minimum prevalence threshold is 30% (3 records out of 10 in the example below). With a default minimum predictability threshold of 50%, and given the input file described above, the *assocgen* program generates the set of rules shown in Table D-10.

Table D-10 Rule Generation Example 1

1	1	30.0000	75.00	60.00	Item=Chips	Item=Milk
1	1	30.0000	75.00	60.00	Item=Coffee	Item=Milk
1	1	30.0000	75.00	40.00	Item=Coffee	Item=Chips
1	1	30.0000	50.00	40.00	Item=Milk	Item=Chips
1	1	30.0000	75.00	40.00	Item=Chips	Item=Coffee
1	1	30.0000	50.00	40.00	Item=Milk	Item=Coffee
1	1	30.0000	100.00	60.00	Item=Soda	Item=Eggs
1	1	30.0000	75.00	60.00	Item=Tea	Item=Eggs
1	1	30.0000	100.00	40.00	Item=Soda	Item=Tea
1	1	30.0000	50.00	40.00	Item=Eggs	Item=Tea
1	1	30.0000	75.00	30.00	Item=Tea	Item=Soda
1	1	30.0000	50.00	30.00	Item=Eggs	Item=Soda

The fields in each line correspond to

- the number of items on the LHS of the rule (always 1)
- the number of items on the RHS of the rule (always 1)
- the prevalence

- the predictability
- the expected predictability (explained in the following section)
- the name (or code) of the item on the LHS
- the name (or code) of item on the RHS

Hierarchical Data Example

Using the example dataset in “Nonhierarchical Data Example” on page 449, Table D-11 shows the mapping of the values at the lowest level to the highest level. The first column represents data at the lowest hierarchical level, while the values Snack, Dairy, and Beverage in the second column represent a higher hierarchical level.

Table D-11 Example Hierarchy

Level 0	Level 1
Milk	Dairy
Chips	Snack
Coffee	Beverage
Eggs	Dairy
Tea	Beverage
Soda	Snack
Cheese	Dairy
Butter	Dairy

In this example, value “Milk” is mapped to “Dairy”, value “Chips” is mapped to “Snack”, and so on. “Snack”, “Dairy”, and “Beverage” can be represented as integers 0, 1, and 2 in the mapping file. Then, the **-map** option can be specified as

```
-map 2 4 synth.map
```

where 2 indicates two levels in the hierarchy, 4 indicates a 4-byte integer for each value in the map file, and *synth.map* is the name of the map file. The binary file *synth.map* for our running example can be found in */usr/lib/MineSet/assocgen/examples* and contains the following values (for purposes of illustration, numbers are provided in decimal rather than binary form):

```
1 0 2 1 2 0 1 1
```

To obtain rules at the lowest hierarchical level, specify **-agg 0**. The program output for this is shown in Table D-12.

Table D-12 Example of Rules at the Lowest Hierarchical Level

```
1    1    30.0000    75.00     60.00    0 | 1    1 | 0
1    1    30.0000    75.00     60.00    2 | 2    1 | 0
1    1    30.0000    75.00     40.00    2 | 2    0 | 1
1    1    30.0000    50.00     40.00    1 | 0    0 | 1
1    1    30.0000    75.00     40.00    0 | 1    2 | 2
1    1    30.0000    50.00     40.00    1 | 0    2 | 2
1    1    30.0000    100.00    60.00    1 | 7    1 | 3
1    1    30.0000    75.00     60.00    0 | 5    1 | 3
1    1    30.0000    100.00    40.00    1 | 7    0 | 5
1    1    30.0000    50.00     40.00    1 | 3    0 | 5
1    1    30.0000    75.00     30.00    0 | 5    1 | 7
1    1    30.0000    50.00     30.00    1 | 3    1 | 7
```

When listing each item, the program shows the complete hierarchical description. For example, 0|1 indicates that item 1 (“Chips”) is mapped to value 0 (“Snack”) at the next higher level in the hierarchy. If you specify **-agg 1**, you get the rules at the next higher level of the hierarchy (which, for this example, is the top level):

```
1       1       80.0000   80.00    80.00  1     0
1       1       80.0000  100.00   100.00  0     1
```

To see the strings “Snack,” “Beverage,” and “Dairy” instead of values 0, 1, and 2 at the top-level hierarchy, specify a third set of options (described in the “Example of Applying Description Files” on page 452).

Example of Applying Description Files

Using the example in “Hierarchical Data Example” on page 450, you can specify a description file for the top-level hierarchy (level 1) as follows:

```
mapassocgen -tran <dataFileName> -agg 1 -map 2 4
        <hierarchyMappingFileName> -ropts -vopts
        -lvldesc 1 <level1descriptionFileName>
        -rout <rulesFileName>
```

The description file *level1descriptionFileName* now looks like this:

```
Snack
Dairy
Beverage
```

The rules at level 1 of the hierarchy then appear like this:

```
1       1       80.0000   80.00    80.00  Dairy    Snack
1       1       80.0000  100.00   100.00  Snack    Dairy
```

Similarly, you can specify a description file for the items at the lowest level. If the description file, *synth0.names*, looks like this:

```
Milk
Chips
Coffee
Eggs
Tea
Soda
Cheese
Butter
```

then item 0 is mapped to “Milk,” item 1 is mapped to “Chips,” and so on. Then if you specify the options string

```
-lvldesc 1 synth1.names -lvldesc 0 synth0.names
```

after **-vopts**, the rules generated at the lowest hierarchical level are shown in Table D-13.

Table D-13 Second Example of Rules Generated at Lowest Hierarchical Level

1	1	30.0000	75.00	60.00	Snack | Chips	Dairy | Milk
1	1	30.0000	75.00	60.00	Beverage | Coffee	Dairy | Milk
1	1	30.0000	75.00	40.00	Beverage | Coffee	Snack | Chips
1	1	30.0000	50.00	40.00	Dairy | Milk	Snack | Chips
1	1	30.0000	75.00	40.00	Snack | Chips	Beverage | Coffee
1	1	30.0000	50.00	40.00	Dairy | Milk	Beverage | Coffee
1	1	30.0000	100.00	60.00	Dairy | Butter	Dairy | Eggs
1	1	30.0000	75.00	60.00	Snack | Soda	Dairy | Eggs
1	1	30.0000	100.00	40.00	Dairy | Butter	Snack | Soda
1	1	30.0000	50.00	40.00	Dairy | Eggs	Snack | Soda
1	1	30.0000	75.00	30.00	Snack | Soda	Dairy | Butter
1	1	30.0000	50.00	30.00	Dairy | Eggs	Dairy | Butter

Rules Visualization

The rules visualization part of the Rules Visualizer graphically displays rules resulting from the association rules generator.

Rules Visualization File Requirements

The rules visualization requires:

- a **rules file** in the internally required format.
- a **configuration file**, which specifies various display parameters.

The Rules File

The rules file is generated by the association rules generator (See "Association Rules Generator" on page 441).

The Configuration File

The configuration file describes how the data from the rules file is to be displayed. This file consists of three sections:

- The **input** section—specifies the rules file to be used.
- The **expressions** section (optional)—creates new viewing parameters.
- The **view** section—specifies how the data is presented.

An example configuration file, *group.ruleviz*, is

```
input {
   file “group.rules”;
}

expressions {
   float ratio = expected / predictability;
}

view {
   height predictability, max 10, legend on;
   disk height expected, legend on;
   color prevalence, scale 0 10, colors “white” “purple”,
   legend on;
   options grid size 6;
   message “LHS: %s\nRHS: %s\npredictability: %.2f
   expected: %.2f  prevalence: %.2f”,
   LHS, RHS, predictability, expected, prevalence;
}
```

Input Section

The input section has the form:

```
input { file “rulesFilename”; }
```

The file statement specifies the rules file. It is the only statement in the input section.

Expressions

The expressions section of the configuration file defines field names to be used subsequently in both this section itself and in the view section. This section specifies new fields in terms of the fields in the rules file. For example, the following line specifies the ratio between predictability and expected predictability:

```
expressions { float ratio = predictability / expected; }
```

The expression section uses field names defined in the rules file. These field names and their types are listed in Table D-14.

Table D-14 Field Names and Types for Rules File

Rules File Field Name	Type	Notes
numLHS	int	Always 1.
numRHS	int	Always 1.
prevalence	float	
predictability	float	
expected	float	
LHS	string	
RHS	string	

Expressions are defined with a combination of field names and operators. The operations and their symbols are listed in Table D-15.

Table D-15 Operators Used With Expressions

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
==	Equals
!=	Not equals
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
&&	AND
||	OR
!	NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
A?B:C	If (A), then B else C

Also, the following functions are available:

- **divide**(x, y, z) divides *x* by *y*, unless *y* is zero. If *y* is zero, the result is *z*; this is equivalent to y==0 ? z : x / y.
- **modulus**(x, y, z) is similar to **divide**, but for modulus.

The following sample code illustrates some of the possible expressions:

```
float variable0 = expected / predictability;
float variable1 = prevalence +1;
float variable2 = variable1 - 1;
float variable3 = variable1 * 3;
float variable5 = 10 % 4;
float variable6 = variable1;
float variable7 = variable1 || 87;
float variable8 = variable1 && 34;
float variable9 = (7 < 5 ? 4 : 3);
float variable10 = divide(15,8,9);
float variable11 = modulus(15,0,9);
float variable12 = (“abc” < “def” ? 1 : 2);
int variable 13 = (int) variable 12;
```

Expressions using **int** and **float** promote both sides to float. Expressions using **int** and **double**, or **float** and **double**, promote both sides to double. The result of a relational expression (for example, ==, <) is always an **int**. Type casting is also supported.

Strings can be compared using relational expressions; the strings are compared lexicographically.

View Section

The view section describes how data is presented. A rule is displayed at the junction of its left-hand-side and right-hand-side items. The view section lets you specify what is shown at the junctions.

The view section has the form:

```
view { viewStatement; ... }
```

You can view bars, disks, and labels at the junctions. The bars and disks have heights and colors.

Height Statement

The height statement describes how the rules are mapped to the heights of bars and disks. The height statement consists of a series of clauses, separated by commas. Alternatively, it can be specified as multiple height statements.

```
height sales, max 2.0;
```

or

```
height sales;
max 2.0;
```

The first clause normally contains the name of a column that is to be mapped to bar height (sales, in the example). The column must be of a number type (int, float, or double); float is the most efficient. If no height column is specified, all bars are flat, and the remaining height clauses have no effect.

The **max clause** specifies the height of the tallest bars. If no max clause is specified, the height is 1.0 in arbitrary units. If, after looking at the view, you see that the heights are too low or too high, use the max clause to adjust them. The syntax of the max clause is

max *float*

where *float* is a floating point number (the decimal point is optional). For example, to specify the maximum height as 2, enter:

```
max 2
```

The **scale clause** scales the arbitrary height values; all values are multiplied by the scale. The syntax of the scale clause is

```
scale float
```

Do not use the scale clause with the max clause.

The **legend** clause specifies whether mapping information is displayed in the lower window pane. This information is about mapping between display entities and data values (for example, bar height corresponds to predictability values). The legend clause has the following syntaxes:

- legend off

  This turns off the height legend (this is the default).

- legend on

  This turns on the height legend. The legend can be changed by using the legend label form, in which case legend on is unnecessary. By default, the legend has the following syntax:

  ```
  height:varname
  ```

  where *varname* is the name of the variable that is mapped to height.

- The legend can be changed by using the legend label form:

  ```
  legend label "string"
  ```

  If legend label is used, legend on is unnecessary.

By default, the height statement affects bars. To specify disks, begin the statement with **disk height**:

```
disk height sales, max 2.0;
```

If no max or scale clause is specified for disk heights, the disks inherit the clause specified for bars.

Color Statement

The color statement describes how values are mapped to colors. The format is similar to the height statement, consisting of several clauses that can be separated by commas, or entered as multiple statements.

Color names must be in quotation marks. Examples of valid colors are “green,” “Hot Pink,” and “#77ff42.” The last one is in the form “*#rrggbb*”, in which the red, green, and blue components of the color are specified as hexadecimal values. Pure saturation is represented by ff, a lack of color by 00. For example, “#000000” is black, “#ffffff” is white, “#ff0000” is red, and “#00ffff” is cyan. You can use the colorview program to determine the names of the colors available on your workstation.

The **color** variable lets you map a column to a color. The column must be a number type. There is no normalization of colors.

The **colors** clause specifies the colors to be used. The colors clause syntax is:

```
colors "colorname" "colorname"...
```

The format for *colorname* has been described above. Note that there are no commas between the colors. This is because commas are used to separate clauses in the color statement. A sample colors clause is

```
colors "red" "gray" "blue"
```

Colors in the list are subsequently referred to by their index, starting at zero. In the above example, red is color 0, gray is color 1, and blue is color 2.

If there is no colors statement, all bars have the same color.

The **scale** clause allows assignment of values to a continuous range of colors. For example, when displaying a percentage, red can be assigned to 0%, gray to 50%, and blue to 100%. Intermediate values are interpolated; for example, 25% is pinkish, and 55% is a slightly bluish gray.

The syntax for the scale clause is

```
scale float float ...
```

The first value is mapped to color 0, the second to color 1, and so forth. The colors statement must contain at least as many colors as are to be mapped to the largest index.

Values in the scale clause must be in increasing order. Any value less than the first color is assigned the value of the first color. Any value greater than the last value is assigned the last color. Intermediate values are interpolated.

For example, assume the pctFemale column indicates what percentage of the group is female, and you want to map a group that is 100% female to red, 100% male to blue, and 50% each to gray. The colors statement for this is:

```
colors pctFemale, colors "blue" "gray" "red", scale 0 50 100;
```

The **buckets** clause is similar to the scale clause without interpolation. All values are rounded down to the highest value in the clause, and that exact color is used. Values less than the first value use the first color.

The syntax for the buckets clause is

```
buckets float float ...
```

The syntax and assignment of colors is the same as for the scale clause.

If, in the above example, you use the buckets clause instead of the scale clause, the statement is:

```
colors pctFemale, colors "blue" "gray" "red", buckets 0 50 100;
```

All values greater or equal to 100 are colored red. Values greater than or equal to 50, but less than 100, are gray. All remaining values is blue.

If a color variable is specified, but neither a scale clause nor a buckets clause is given, a default scale clause is used. The values are generated automatically, ranging from the minimum value to the maximum value in the data.

The Legend Clause

The **legend** clause creates a legend of the colors. The legend clause syntax can be any of the following:

```
legend off
legend on
legend "string" "string" ...
legend label "string"
```

The **legend off** clause turns the legend off. The **legend on** clause turns the legend on. It can be omitted if other legend statements are included. Specifying only **legend on** generates the default legend.

The default legend includes a single label to the left (with the name of the field that is mapped to color), and a list of colored labels on the right (with values obtained from the scale clause or the buckets clause). To override the strings in the colored labels, specify the strings as shown:

```
legend "string" "string"
```

To override the label on the left, specify it following the word *label*. To eliminate this label, specify an empty string; that is:

```
legend label ""
```

By default, the color statement affects bars. To affect disks, begin the statement with **disk color**:

```
disk color pctFemale;
```

If no colors, scale, or buckets clause is given for disk colors, the disks inherit the clauses given for bars.

Label Statement

You can specify a variable name and a color for labels to appear in front of the bars, at the base. By default, no labels appear. The color is a single color (unlike the bars and disks). For example, the following line displays the numeric predictability value in red at the base of each bar:

```
label predictability, color “red”;
```

Message Statement

The message statement specifies the message displayed when the pointer is moved over an object or when an object is selected. The syntax is similar to that of the C **printf** statement. A sample message statement is

```
message "LHS: %s\NRHS: %s\npredictability/expected: %.2f",
             LHS, RHS, predictability/expected;
```

This could produce the following message:

```
LHS: milk
RHS: bread
predictability/expected: 2.00
```

The formats must match the type of data being used:

- Strings must use %s.
- Ints must use integer formats (such as %d).
- Floats and doubles must use floating point formats (such as %f).

For details of the **printf** format, see the printf (1) reference (man) page (type **`man printf`** at the shell prompt).

A special format type has been added to **printf**. If the percent sign is followed by a comma (for example, `%,f`), commas are inserted in the number for clarity. Currently, only the United States convention of d,ddd,ddd.dddd is supported, with the decimal point represented by a period, and commas separating every three places to the left of the decimal point. For example, if the above format were:

```
message "LHS: %s\nRHS: %s\npredictability/expected: %.2f",
      LHS, RHS, predictability/expected;
```

it would produce the message:

```
LHS: milk
RHS: bread
predictability/expected: 1,000.00
```

The $, *, h, l, ll, L, and n **printf** format options are not supported.

All values, including the format string, are expressions. Thus, if want to distinguish rules with predictability greater than twice the expected, you can use

```
message predictability > expected*2 ? "LHS: %s\nRHS:
      %s\npredictability/expected: HIGH" : "LHS: %s\nRHS:
      %s\npredictability/expected: LOW" , LHS, RHS;
```

This could produce the message

```
LHS: milk
RHS: bread
predictability/expected: LOW
```

or:

```
LHS: milk
RHS: cake
predictability/expected: HIGH
```

You could also achieve the same result with a single format string:

```
message "LHS: %s\nRHS: %s\npredictability/expected: %s",
   LHS, RHS, predictability > expected*2 ? "HIGH" : "LOW";
```

If no message is specified, a default message is used.

Item Statement

An item statement describes the item names displayed along the LHS and RHS axes. An item statement has the form

```
item colors <colorLeft> <colorRight>;
```

The colors are absolute colors, like the label color. There is also a statement to turn on / off the item names:

```
item <off|on>;
```

Grid Statement

The grid statement describes the grid on which the rules are displayed. A grid statement has the form

```
grid color <color>;
```

or

```
grid <off|on>;
```

The color is a single color.

Options

The options statement lets you fine-tune certain parameters of the display. When the view section is first entered, options are loaded from the defaults file */usr/lib/MineSet/ruleviz/view.ruleviz.options*. This file is searched for in the following directories, in the order listed:

- */usr/lib/MineSet/ruleviz*
- *~/.MineSet* (where ~ is your home directory)
- the current directory

The file is not required to be present.

Options specified directly in the configuration file override those in the defaults files. The syntax of the options statement is

```
options option, option, ...
```

The following options are available:

- bar label size
- hide bar label distance
- hide disk distance
- grid size
- item size
- hide item distance
- font

The following is a description of each option.

- `options bar label size` *float*

  Specifies the size of the labels in front of the bars. Larger values result in larger labels.

- `options hide bar label distance` *float*

  Specifies the distance at which the bar labels are not drawn. Smaller distances improve performance, but the labels might not be visible.

- `options hide disk distance` *float*

  Specifies the distance at which disks are not drawn. Smaller distances improve performance, but the disks might not be visible.

- `options grid size` *float*

  Specifies the width and depth of grid cells.

- `options item size` *float*

  Specifies the size of the items.

- `options hide item distance` *float*

  Specifies the distance at which the items are not drawn. Smaller distances improve performance, but the items might not be visible.

- `options font` *”fontName”*

  Specifies the font used for items and bar labels.

Appendix E

Command-Line Interface to MIndUtil: Classifiers, Discretization, Column Importance, and File Conversions

The first part of this appendix describes the MIndUtil program and its options. The second part lists and describes the general options for MIndUtil. The final part describes induction modes. The MIndUtil program comes with the server side of the MineSet images and is invoked automatically on the server when working through Tool Manager.

MIndUtil Invocation and Options

MIndUtil provides the MineSet inducers and mining utilities, such as discretization (binning). It also provides features for file conversions. In the following description, all examples assume the UNIX shell is *csh* or *tcsh*. Users of *sh* and *ksh* can transform `setenv ENV val` into `env=val; export env.`

The syntax for invoking MIndUtil is:

```
MIndUtil [-s] [-o <optionfile>] [-O <option>=<value>]
```

where the **-s** option suppresses environment options (described below). The **-o** option allows reading options from an ASCII option specification file containing one `<option>=<value>` per line. By convention, MineSet uses the suffix *.classify-opt* for such option files. The **-O** option (uppercase) allows setting a specific option by following it with the option name, an equal sign, and the value. The **-o** and **-O** can be repeated multiple times. If an option is set more than once, the last time it is set determines its value. For example, if it is set through an option file and then set again through using the **-O** flag, the latter one determines its value.

Each option has a unique name; all option names are written in uppercase letters. If you want to set up the *.datamove* file to keep data and classifier option files on the server, the following lines must be in the *.datamove* file:

```
keep_data_files=yes
keep_classifier_options_files=yes
```

This ensures that the option specification files ending with the *.classify-opt* extension (consisting of the options passed to MIndUtil via the Tool Manager) are not erased from the server after you invoke inducers through Tool Manager.

Example With MIndUtil Options

A typical file (*iris.classify-opt*) might contain the following lines:

```
MODE=train-and-test
LABEL=iris_type
INDUCER=decision-tree
DT_MAX_LEVEL=0
DT_SPLIT_BY=normalized-mutual-info
DT_PRUNING_FACTOR=0.85
DT_LBOUND_MIN_SPLIT=5
CLASSIFIER_NAME=iris-dt.class
VIZ_NAME=iris-dt.treeviz
TRAIN_FRACTION=0.666667
ACC_EST_SEED=7258789
```

Given a schema file (*iris.schema*, which references *iris.data*), you can run MIndUtil from the command line to induce a decision tree by using the options file as follows:

```
MIndUtil -o iris.classify-opt -O SCHEMA=iris.schema
```

This is exactly the way MIndUtil is invoked by DataMover.

Options in MIndUtil can be set through a hierarchy of levels. An option set at a higher level (see below) overrides any setting from a lower level. The levels are:

- *Hard-coded default*—Many options have a hard-coded default value. If the value is not overridden in any of the higher levels, the hard-coded default is used.
- *Environment option*—An environment variable can contain the option's value. You can set the environment variable with the same name as the option itself. For example,

  ```
  setenv SCHEMA iris.schema
  ```

  sets the SCHEMA option to *iris.schema*.

  An environment variable takes precedence over hard-coded defaults. The command-line option **-s** suppresses environment variables.
- *Command-line options*—You can set specific options with `-O <option>=<value>`

  For example, to generate a decision tree from *iris.classify-opt*, set the pruning factor to 0, and set the minimum records in a split to 1, use:

  ```
  MIndUtil -o iris.classify-opt -O SCHEMA=iris.schema \
          -O DT_PRUNING_FACTOR=0 -O DT_LBOUND_MIN_SPLIT=1
  ```

  This induces a larger tree for the iris dataset. Command-line options take precedence over environment variables and hard-coded defaults.

  The order of command line arguments is important: Options to the right override earlier options to their left. Thus, the **-O** options override the values set in the *iris.classify-opt* file.
- *User input*—If an option is required but the option was not set using any of the above levels in the hierarchy, you are prompted for the option, and you can type a value.

  If you type **?**, a help string appears to explain the meaning of the option. User input has the highest precedence and given values override command line options, environment variables, and hard-coded defaults.

A special environment variable, called PROMPTLEVEL, determines when to prompt the user for an option. The variable has three possible values:

- *Required-only* prompts you for required options only. There are no prompts for options with a hard-coded default value. This is the lowest level prompting mode and the default.
- *Basic* prompts you for basic options (each option is hard-coded as basic or not), whether or not they have a default value. Some options are defined as "nuisance" (non-basic) options and are not prompted for by this mode. The purpose of this mode is to prompt for the most commonly used options.

  If the option has a default, you can change it to be a nuisance option by setting the option value to an exclamation mark ("!"). A nuisance option can be changed to a non-nuisance option by setting its value to be a question mark ("?"). For example, you can type:

  ```
  setenv PROMPTLEVEL basic
  MIndUtil -o iris.classify-opt -O DT_MAX_LEVEL="?"
  ```

  You now are prompted for most options (non-nuisance) with defaults taken from *iris.classify-opt*. To accept the default, press Enter. Since SCHEMA is not defined in *iris.classify-opt* and is a required option, you are prompted for SCHEMA without a default. DT_MAX_LEVEL is a nuisance option, but setting it to a question mark specifies to prompt for it.
- *All* prompts you for all options, regardless of their nuisance setting.

When MIndUtil is executed from Tool Manager, all options except the schema filename are passed from the client through an options file *<file>.classify-opt*. On the server, the DataMover invokes MIndUtil with the options file and the appropriate schema file.

Tool Manager prepends any options it finds in *.mineset-classopt* on the client workstation. The file is searched first in the current directory, then in the home directory. The first one found is used.

When an option requires one of a given set of values (for instance, an enumerated option), a prefix of the desired option value can be used, and comparison is case-insensitive. If there are multiple values with the given prefix, the first one in the list is chosen. For example, the first option in MIndUtil is MODE, which takes on one of the following values: *induce*, *train-and-test*, *estimate-accuracy*, *discretize*, *auto-select*, *compute-importance*, *mineset-to-mlc*, *mlc-to-mineset*, or *visualize*. Setting the option to "i" selects induce. In scripts, use the full option name for future compatibility.

To facilitate repeat runs of a program under the same options, these can be put in a file. The file must be in a format that can be sourced by *csh* or *tcsh*. The name of the file can be set through the environment variable *OPTION_DUMP*. For example, if your *.login* contains the statement

```
setenv OPTION_DUMP ~/.mindoptions
```

the options from the last run are stored in *~/.mindoptions*.

To repeat a previous run, simply source the dump file; for example,

```
source ~/.mindoptions
```

Note that the file is generated as you input options. This means you can source the file from *csh* or *tcsh* to get the options you have already filled in manually, even if you aborted the run.

General Options

MIndUtil is written using MLC++, the machine learning library in C++ (see *http://www.sgi.com/Technology/mlc*). More options can be used for those familiar with MLC++. Here are the important ones shared by many modes.

All filename specifications require the file suffix, except where detailed below.

- MODE is an enumerated option containing one of the following: *induce*, *train-and-test*, *estimate-accuracy*, *discretize*, *auto-select*, *compute-importance*, *mineset-to-mlc*, *mlc-to-mineset*, or *visualize*.
  - Induce builds a classifier using all the data.
  - Train-and-test splits the data into a training set and a test set; a classifier is built from the training set and evaluated on the test set.
  - Estimate-accuracy performs cross-validation to estimate the accuracy of a classifier built using the induce option.
  - Discretize allows discretizing continuous attributes.
  - Auto-select allows finding a set of important attributes together with prespecified attributes.
  - Compute-importance computes the importance of each attribute as if it were used individually with a prespecified set of attributes.
  - MineSet-to-MLC and MLC-to-MineSet allow converting files from MLC++ format to MineSet format and vice versa.
  - Visualize allows converting a classifier to a visualization.

  The modes are detailed in the next section.
- SCHEMA is a string defining the MineSet schema file to use. The file specification must be complete (that is, with the *.schema* extension). (See “Using MineSet With Existing Data Files” in Chapter 2 for a description of schema files.)
- LABEL is the name of the column or attribute that is to be used as the label whenever it is needed. The label name must be one of the columns in the schema file.

- DISC_TYPE is an enumerated option taking on the value *binning* or *entropy*. This determines the discretization mode. *binning* invokes uniform binning, while *entropy* invokes “automatic” binning that is nonuniform (see “The Bin Column Button” in Chapter 3). The default is *entropy*.
- MIN_SPLIT is an integer specifying the minimum number of instances that must be in each bucket when discretization is being done.
- LOGLEVEL is an integer >= 0 defining the amount of logging information to print during the run. The default is zero. This option is hidden; you are not prompted for it.
- DRIBBLE is a Boolean operation defining whether to dribble output during processing in order to show progress. The default is TRUE. This option is hidden; you are not prompted for it.
- LINE_WIDTH is an integer > 1 defining the line width for the output. Automatic wrapping occurs to break words before this width. Wrapped lines begin with the WRAP_PREFIX string. The default line width is 79. This option is hidden; you are not prompted for it.

Induction Modes

This section describes the options for induction modes, induce and train-and-test. The induce mode induces a classifier using the whole dataset. The train-and-test mode induces a classifier on a portion of the dataset and tests it on a holdout set. The modes require specifying an INDUCER option, which is either decision-tree or evidence. Options shared by both inducers in train-and-test mode are:

- VIZ_NAME is a string defining the visualization name whenever appropriate. For the decision tree inducer, this fully specified filename is the name of the configuration file (recommended suffix is *.treeviz*) and a suffix of *.data* is automatically added to the data file needed. For the evidence inducer, only one filename is needed (recommended suffix is *.eviviz*).
- CLASSIFIER_NAME is a string defining the classifier name whenever appropriate.

- DISP_CONFUSION_MAT is a Boolean defining whether to display a confusion matrix, showing the distribution of mistakes for one class against another. This is a rudimentary ASCII display.
- TRAIN_FRACTION is a floating point number between 0 and 1. It determines what ratio of the records to use as a training set. The rest are used as a test set. The default is two-thirds.

The option shared by both inducers under both induce and train-and-test modes is:

- ACC_EST_SEED is an integer serving as the seed for the random-number generator used to split the records into training and test sets.

Decision Tree Inducer Options

The following options are available for decision trees (INDUCER = decision-tree):

- DT_MAX_LEVEL, an integer >=0, limits the number of levels to grow the decision tree. The default of zero implies no limit.
- DT_PRUNING_FACTOR, a floating point number >=0, determines the pruning factor. The default of zero implies no pruning (Tool Manager defaults to a pruning factor of 0.85).
- DT_LBOUND_MIN_SPLIT, an integer, provides a lower bound on the number of records required to trickle down to at least two branches in a given node. No split will be made otherwise. The default is 1 (Tool Manager defaults to 5).
- DT_MIN_SPLIT_WEIGHT, a floating point number >=0, is the minimum ratio of training records divided by the number of classes that are required to trickle down to at least two branches in a given node. The default is 0 (Tool Manager defaults to 0.1).

- DT_SPLIT_BY is an enumerated option taking one of the following values: mutual-info, normalized-mutual-info, gain-ratio. It specifies the evaluation criterion for choosing the attribute to split on at every node (see Chapter 9 for details). The default is normalized-mutual-info.
- DT_ADJUST_THRESHOLDS is a Boolean operator determining whether splits on continuous attributes have thresholds that are midpoints between two data points or whether the thresholds should be actual data values. The default is FALSE (that is, not to adjust the thresholds to data values).

  This option is useful when you want to avoid splits on fractional values if attributes take on only integer values.

Evidence Inducer Options

The following options are available for the evidence inducer (INDUCER = evidence):

- EVI_LAPLACE_CORRECTION, a Boolean determining whether to apply the Laplace correction (see Chapter 10). The default is false.
- EVI_FSS, a Boolean determining whether to apply feature subset selection (see Chapter 10). The default is false.

Estimate Accuracy

If the MODE is estimate-accuracy, cross-validation will be performed. The following options are available:

- CV_FOLDS is an integer determining the number of cross-validation folds. The default is 10. See Chapter 8 for details.
- CV_TIMES is an integer determining the number of times to repeat cross-validation. The default is 1. See Chapter 8 for details.

All other options are the same as for the induction modes.

Discretization

If MODE is discretize, discretization of attributes is performed and thresholds are determined. The following options are available:

- OUTPUT_NAME is the name of the file to contain the results.
- ATTR_X, where X starts at 0 and increases. This defines the names of the attributes that you would like discretized.
- BINS_X, where X starts at 0 and increases. This specifies the number of bins to discretize ATTR_X into. Note that this is an upper bound and entropy binning may choose a lower number of bins. If the number of bins is zero, automatic heuristics are used.

Example: A discretization of two attributes, petal width and petal length, according to label iris_type, such that the number of bins is automatically determined can be done using the following options:

```
MODE=discretize
LABEL=iris_type
ATTR_0=petal_length
BINS_0=0
ATTR_1=petal_width
BINS_1=0
DISC_TYPE=entropy
MIN_SPLIT=5
OUTPUT_NAME=iris.disc
```

Column Importance and Auto Selection

If MODE is auto-select or compute-importance, corresponding to the Find Importance and Compute Importance modes in the Tool Manager's Column Importance, the following options are available:

- OUTPUT_NAME is the name of the file to contain the results.
- ATTR_X, where X starts at 0 and increases. This defines the names of preselected attributes. All other attributes are candidates for auto-selection or are ranked if column importance is chosen.
- SELECT_N, an integer that determines the number of attributes to automatically select in auto-select mode, which corresponds to the non-Advanced mode and the advanced "find..." mode in column importance in the Tool Manager.

Example: To choose three attributes that can be used together with petal_width to classify iris_type, the following options can be used:

```
MODE=auto-select
LABEL=iris_type
SELECT_N=3
ATTR_0=petal_length
DISC_TYPE=entropy
MIN_SPLIT=5
OUTPUT_NAME=feature.fss
```

In this example and the one in the "Discretization" section, the discretization mode was entropy and the minimum number of instances in a bin was set to 5.

MineSet-to-MLC, MLC-to-MineSet

These modes provide facilities to convert from MLC++ format to MineSet format and vice versa. They can be used to convert UC Irvine (*http://www.ics.uci.edu/~mlearn/MLRepository.html*) formatted files or C4.5 formatted files, which are common in the machine learning community.

MineSet-to-MLC provides the following options:

- SPLIT_TRAIN_TEST, whether to split the data into two files: a training set and a test set.
- MLCFILE, the filename to export to. Suffixes of *.names*, *.data*, and *.test* are appended to this stem if SPLIT_TRAIN_TEST is true; otherwise, the suffixes are *.names* and *.all*. You can then run MLC++ inducers on these files, independent of MineSet.

MLC-to-MineSet provides the following options:

- DATAFILE, the file to import. If no suffix is given, it is assumed to be *.data*. It is recommended that you concatenate the training and test sets into a *.all* file and use that for importing.
- NAMESFILE, the names file describing the DATAFILE. A reasonable default is automatically suggested based on the DATAFILE option.
- OUTPUT_DATA, a MineSet output file. This should have a *.data* suffix.
- OUTPUT_SCHEMA, a MineSet schema file. This should have a *.schema* suffix.
- OUTPUT_LABEL, a string indicating what name to use for the label attribute.
- REMOVE_UNKNOWN_INST, a Boolean option indicating whether to remove records that have attributes with unknown values. The default is FALSE.

Visualize

This mode lets you generate visualization files from classifiers. The following options are available:

- CLASSIFIER_NAME is the name of the classifier, including the file suffix.
- VIZ_NAME is a string defining the visualization name. For the decision tree inducer, this fully specified file name will be the name of the configuration file (recommended suffix is *.treeviz*) and a suffix of *.data* will be automatically added to the data file needed. For the evidence inducer, only one file name is needed (recommended suffix is *.eviviz*).

Appendix F

Format of the Evidence Visualizer’s Data File

The purpose of this appendix is to describe the Evidence Visualizer’s input data file. This file is a textual representation of the evidence classifier. The data file is generated automatically through the toolmanager. In some instances one may wish to edit this file in order to alter label or attribute names, or to rearrange values.

The Evidence Visualizer requires a data file containing the label and attributes, along with counts and probabilities. These are used to create the graphics. It is output as a result of running the Evidence Inducer through the Tool Manager. The format of the data file is:

```
"<Label>" <L>
"<label1>" <count1> <probability1>
"<label2>" <count2> <probability2>
:
"<labelL>" <countL> <probabilityL>

<M>

"<attrib1>" <N1> <importance1>
"<value1_1>" <cnt1_1_1> <prob1_1_1> ... <cnt1_1_L> <prob1_1_L>
"<value1_2>" <cnt1_2_1> <prob1_2_1> ... <cnt1_2_L> <prob1_2_L>
:
"<value1_N1>" <cnt1_N1_1> <prob1_N1_1> ... <cnt1_N1_L> <prob1_N1_L>

"<attrib2>" <N2> <importance2>
"<value2_1>" <cnt2_1_1> <prob2_1_1> ... <cnt2_1_L> <prob2_1_L>
"<value2_2>" <cnt2_2_1> <prob2_2_1> ... <cnt2_2_L> <prob2_2_L>
:
"<value2_N2>" <cnt2_N2_1> <prob2_N2_1> ... <cnt2_N2_L> <prob2_N2_L>

:
:
:
```

```
"<attribM>" <NM> <importanceM>
"<valueM_1>" <cntM_1_1> <probM_1> ... <cntM_1_L> <probM_1_L>
"<valueM_1>" <cntM_2_1> <probM_2_2> ... <cntM_2_L> <probM_2_L>
:
"<valueM_NM>" <cntN1_NM_1> <probM_NM_1> ... <cntM_NM_L> <probM_NM_L>
```

Where *L* is the number of label values, *M* is the number of attributes, and *N* is the number of values or bins for attribute *i*. The <>’s indicate variables. The actual file has numbers or strings. NULL is considered a unique value if it is present in an attribute. If NULLs exist for an attribute they always appear as the first value (i.e., the first line following the attribute header) and are represented by "?".

The counts are the number of records in the table that have that particular attribute value (or range of values). Hence the sum of the counts for each attribute equals the total number of records in the table. The probability is the number of counts for that attribute value divided by the total number of counts. If the data file was generated with Laplace correction turned on, the probability is only approximately the number of counts for that attribute value divided by the total number of counts (see “Refining the Inducer With Further Options” in Chapter 10). Hence the probability value indicates the proportion of labelX values that have this attribute value instead of another value.

Data files must have a *.eviviz* extension. When starting the Evidence Visualizer or when opening a file, you must specify the data file.

Here’s a concrete example of an eviviz data file, */usr/lib/MineSet/eviviz/examples/cars.eviviz*:

```
"origin" 3
"US" 254 0.625616
"Europe" 73 0.179803
"Japan" 79 0.194581

6

"mpg" 5 25.448
"?" 5 0.019685 3 0.0410959 0 0
"- 16.1" 87 0.34252 0 0 0 0
"16.1-21.05" 77 0.30315 10 0.136986 5 0.0632911
"21.05-30.95" 67 0.26378 43 0.589041 28 0.35443
"30.95+" 18 0.0708661 17 0.232877 46 0.582278

"cylinders" 5 29.1759
"8" 108 0.425197 0 0 0 0
"4" 72 0.283465 66 0.90411 69 0.873418
"6" 74 0.291339 4 0.0547945 6 0.0759494
"3" 0 0 0 0 4 0.0506329
"5" 0 0 3 0.0410959 0 0

"horsepower" 4 21.1419
"?" 4 0.015748 2 0.0273973 0 0
"- 78.5" 25 0.0984252 39 0.534247 46 0.582278
"78.5-134" 131 0.515748 31 0.424658 33 0.417722
"134+" 94 0.370079 1 0.0136986 0 0

"weightlbs" 4 28.5157
"- 2379.5" 30 0.11811 43 0.589041 57 0.721519
"2379.5-2959.5" 57 0.224409 18 0.246575 22 0.278481
"2959.5-3274" 29 0.114173 9 0.123288 0 0
"3274+" 138 0.543307 3 0.0410959 0 0

"time_to_60" 3 10.0055
"- 13.45" 78 0.307087 3 0.0410959 3 0.0379747
"13.45-19.45" 162 0.637795 52 0.712329 75 0.949367
"19.45+" 14 0.0551181 18 0.246575 1 0.0126582

"year" 1 2.84217e-14
"ignore" 254 1 73 1 79 1
```

Note that the sum of the probabilities corresponding to a particular label value for a given attribute always equals one. Consider attribute "weightlbs", for label value "US" (the first one), we have .11811+.224409+.114173+.543307=1.0 . Also note that attributes mpg and horsepower have NULL values.

Appendix G

Nulls in MineSet

Nulls represent unknown data. MineSet supports nulls in the data access tools, the mining tools, as well as the visualization tools. The purpose of this appendix is to give you a better understanding of the way MineSet handles nulls.

Semantics of Nulls

Unknown data values are often represented as nulls in data sources. While it is possible to associate different semantics with nulls, the most commonly used semantic is that of nulls representing missing or unknown values. For example, if a data record is made up of fields representing FIRSTNAME, MIDDLENAME, LASTNAME, and if a person’s MIDDLENAME is not known, it can be represented by the null value.

Some databases, such as Oracle RDBMS, do not distinguish between null and empty strings. In such a case, it is not possible to distinguish between an unknown middle name and a person who does not have a middle name. On the other hand, Sybase RDBMS distinguishes between null and empty strings.

Like most relational databases, MineSet associates the semantics of unknown values with nulls. The ability to distinguish between null values and nonexistent values depends on the source of data. Thus, when accessing data from Sybase, MineSet can differentiate between nulls and nonexistent values.

Nulls can occur in data for a variety of reasons: They can occur naturally in data as a means to represent unknown data, or they can come about as the result of doing certain kinds of aggregations. For example, if there are no flights between San Francisco and MineSet City, a query such as "find the average flight time from San Francisco to MineSet City" yields a null value.

Representation of Nulls

In data files, as well as in the visual tools, nulls are represented by the string "?" (question mark). Thus, if Joe Miner's middle name is unknown, his name is represented in our example data file (having schema FIRSTNAME, MIDDLENAME, LASTNAME) as:

```
Joe     ?     Miner
```

The graphical representation of nulls varies from tool to tool. See the chapters on the individual tools for a discussion of how they represent them graphically.

Operations on Nulls

Given the semantic that nulls represent unknown values, it becomes straightforward to give meaning to expressions involving nulls.

Arithmetic Expressions

Arithmetic operations involving nulls always give a null result. For example:

(5 + ?) evaluates to ? (adding 5 to an unknown yields yet another unknown);

(6 / ?) evaluates to ?

Boolean Expressions

In addition to taking on the values of TRUE and FALSE, Boolean variables can also be null. If a Boolean valued variable has a null (unknown) value, the result of combining it with another Boolean variable in an expression is unknown, unless it is possible to determine just from the known value what the result is. In particular:

? and FALSE is FALSE, because FALSE ANDed with anything is always FALSE

? and TRUE is ?

? or FALSE is ?

? or TRUE is TRUE, because TRUE ORed with anything is always TRUE

not ? is ?

Relational operations

Relational operations (==, !=, <, >, <=, and >=) involving nulls always evaluate to null. Some particular cases worth emphasizing are:

? == ? evaluates to ?, not TRUE

? != ? evaluates to ?, not FALSE

? != x evaluates to ?, not FALSE

Given two unknown values, it is unknown whether the two are equal or unequal. This behavior can be confusing when using a search panel. For example, when searching for all values not equal to 0, nulls do not show up, yet neither do they show up when searching for values equal to 0. Because of this, search panels provide the ability to search explicitly for nulls. (Some search panels provide the option of treating nulls as zeros; see the individual tool discussions for more information.)

Testing for nulls

The function **isNull()** can determine whether or not a variable has the value null. For example:

isNull(*X*) evaluates to TRUE if variable *X* has the null value

isNull(*X*) evaluates to FALSE if variable *X* has a non-null value

Aggregations in the Presence of Nulls

MineSet stays close to the semantics of SQL and relational databases when aggregating columns that might have null values. Thus, null values are ignored when computing SUM, AVG, MIN, MAX, and COUNT. This is best illustrated by an example. Consider a data file having records representing the number of pets a person has. The schema of this record is NAME, NUM_PETS, and null (unknown) values are represented by "?".

Table G-1

Name	NUM_PETS
Tesler	3
Rathmann	?
Haber	1
Bhargava	0
Sangudi	?

Then,

SUM(NUM_PETS) = 4

COUNT(NUM_PETS) = 3 (and not 5, even though there are 5 rows of data)

AVG(NUM_PETS) = 1.33

MAX(NUM_PETS) = 3

MIN(NUM_PETS) = 0

In these aggregations, null values are basically ignored (note that the value 0 is different from ?, and is not ignored).

A special case of this is an aggregation where all the values being aggregated are themselves null. An even more specialized case is when there are no values being aggregated: for instance, when summing an empty column. In both these cases, the sum, average, min, and max are ?, while the count is 0.

Sort Order for Nulls

In an ascending sorted sequence, null values always appear before non-null values. In a descending sorted sequence, null values always appear after non-null values.

Bins and Arrays With Nulls

MineSet lets you bin numeric data into bins or discrete intervals. It also lets you (via the aggregation panel in the Tool Manager) create arrays on these bins. When a column of values is binned, all null values are put in a bin labeled "?". Such a bin label is always created, whether or not the data being binned has nulls in it. You have control over whether to use this bin for nulls in your application. You can do so by allowing arrays to ignore or keep bins for nulls by setting the desired option in the Tool Manager’s Preferences dialog. For example, if you know that the column being binned has no nulls, or you intend to study the data corresponding to non-null values only, you can choose to ignore the bin for nulls.

Appendix H

Examples of Tool Usage

The purpose of this appendix is to give you a better understanding of the kinds of information that can be distilled from data with the MineSet visual tools. Stepping through the following examples can help you apply the visual tools to your data.

This appendix consists of two parts.

- "Starting Up MineSet" describes what you must do before trying the examples.
- The second part consists of one demonstration script for each MineSet visual tool.

Starting Up MineSet

All examples begin with Parts A and B (which are performed only once). You must perform parts C, D, and E each time MineSet is invoked.

Part A: Install MineSet

Set up MineSet as described in Chapter 2.

Installed on the server in */usr/lib/MineSet/DBexamples* are

- all the sample data, along with a brief description of what it contains.
- directions on how to load the data using the provided scripts. (This is in the *README.server* file as well as in step B.)

Part B: Load the Sample Datasets

Load the sample datasets into an Oracle database that has been set up on your server. The data and these directions (*README.server*) are installed in */usr/lib/MineSet/DBexamples* on the server.

The */usr/MineSet/DBexamples* directory contains scripts for loading the complete set of data files into one of the supported databases. To load the complete set of data, run one of the following loader scripts, depending on which database you have. (This assumes your database and environment are already set up.)

```
sh load_all_Oracle.sh <userid> <passwd>
```

```
sh load_all_Sybase.sh <userid> <passwd>
```

If you are going to work with an INFORMIX database, use the `dbaccess` interface to select

```
create_all_Informix.sql
```

followed by

```
load_all_Informix.sql
```

Loading Individual Datasets

Alternatively, you can load, or reload, the sample data separately. Each data directory in */usr/lib/MineSet/DBexamples* on the server contains files necessary to load the data into any of the supported databases. These files are:

README - explains the data

**.sql* - sets up an Oracle table
**.ctl* - control file for loading into Oracle

**_syb.sql* - sets up a Sybase table
**.bcf.fmt* - Sybase format file

**_inf.sql* - sets up an INFORMIX table
**_load.sql* - loads the data into the INFORMIX table

In the **.ctl* file, the separator is declared in the line

```
" fields terminated by X'20'  "
```

The separator is specified in ASCII hexadecimal; thus:

`X'20'` is used for ‘ ’
`X'2c'` is used for ‘,’
`X'09'` is used for ‘\t’

Loading Into Oracle

Perform the following steps on the server with an Oracle database:

1. Ensure the following environment variables are set correctly:

   ```
   ORACLE_HOME
   ```

   ```
   ORACLE_SID
   ```

2. Type

   ```
   sqlplus <userid>/<passwd>
   SQL> @<dataset>.sql
   ```

   Where *dataset* is the name of the dataset being loaded, and *userid*/*passwd* are your assigned username and password for the Oracle database.

   To delete an already existing table, type

   ```
   SQL>  drop table dataset;
   ```

3. Type

   ```
   sqlload control = <dataset>.ctl userid = <userid>/<passwd>
   log = /tmp/<dataset>.log  direct = true
   ```

4. Check the resulting *dataset.log* to ensure the data was loaded correctly.

Loading Into Sybase

Perform the following steps on the server with a Sybase database:

1. Ensure that the following environment variables are set:

   ```
   SYBASE
   DSQUERY
   ```

2. To create the table, type

   ```
   isql -U<userid> -P<passwd> -i <dataset>_syb.sql
   ```

   Where *dataset* is the name of the dataset being loaded, and *userid/passwd* are your assigned username and password for the Sybase database.

   To delete an already existing table, type

   ```
   isql -U<userid> -P<passwd>
   drop table <dataset>
   go
   ```

3. To load the data, type

   ```
   bcp <dataset> in <dataset>.data -U<userid> -P<passwd> -f
   <dataset>.bcp.fmt
   ```

where *dataset* is the table name (created via <`dataset`>`_syb.sql`), `in` means "load into the dbms," <*dataset*>`.data` refers to the name of the ASCII data file, and `-f` points to the already-created format file. (When reading in from a file, the data types are character.)

Loading Into INFORMIX

Perform the following steps on the server with an INFORMIX database:

1. Ensure the following environment variables are set:

   ```
   ONCONFIG
   INFORMIXSERVER
   INFORMIXTERM
   ```

2. To create the table, type

   `dbaccess`

3. If necessary, log into the appropriate database.
4. Choose *Query-language*, then choose the appropriate database from those listed.
5. Choose *<dataset>_inf.sql*, and run it.
6. Choose *<dataset>_load.sql*, and run it (where *<dataset>* is the name of the dataset being loaded).

Part C: Run MineSet

At the UNIX prompt on your workstation, start the MineSet Tool Manager by typing the command **`mineset`**. The MineSet Tool Manager screen comprises four panels:

- Server Name panel (upper left)
- Data Source panel (lower left)
- Data Transformations panel (middle)
- Data Destination panel (right)

Part D: Log on

To log on to the server and database:

1. Type the server name in the upper left panel.
2. Click *Log in to Server.*
3. Enter your server user name and password.
4. Click *OK.*
5. Use the DBMS dropdown menu to choose the appropriate database that is installed on the server.
6. Enter your database username and password.
7. Click *OK.*
8. If necessary, choose a database using the Database dropdown menu.

Part E: Choose the Database Table

You are now ready to choose tables from the database.

1. Use the Table dropdown menu to choose the desired table.
2. Alternatively, you can select SQL query from the TableType dropdown menu if you want to use SQL to retrieve data.

Note: While making table manipulations, you can back up to a previous state at any time by clicking the left mouse button in the Table History section of the Data Transformation panel.

Map Visualizer Example Using the Netherlands' Birth Data

This example uses the Map Visualizer to view a data file with three interrelated components: birth rates, population sizes, and geographical region.

Getting the Data

To retrieve the data file, choose the *nlb* table from the Table dropdown menu.

The Current Columns list (middle) shows columns for the *nlb* table, which contains the following information about birth rates in the Netherlands from 1989 to 1993:

- regions—three letter acronym for state in Netherlands
- sqkm—the area of the state
- year—the year
- population—the population of the state that year
- age—age of the mother
- birthrate—the number of births for that age of mother

Note: To use sliders in tools, arrays of data are required.

Creating Arrays

To make arrays, you need binned columns to act as indexes. To bin the year and age columns:

1. Click *Bin Columns...* The Bin Columns dialog box appears.
2. Choose "year" from the list of columns at the top of the dialog box.
3. Choose the Automatic Thresholds tab.
4. Check *Group into:*.
5. Enter 5 so it reads *Group into:* 5 *bins*.

6. Choose Uniform from the Use Approach dropdown menu
7. Click *Apply*. This creates one-year bins for each year 1989, 1990, 1991, 1992, and 1993. A new column, year_bin, appears when you close this dialog.

To bin the age column:

1. Click *age* in the Columns to Bin list.
2. Check *Group into:*.
3. Enter 6. This field should read *Group into:* 6 *bins*.
4. Click *Apply*. This creates the age bins. The boundaries of the bins are: 22, 27, 32, 37, and 42.
5. Click *Close*.

The original year and agebin columns are replaced with year_bin and age_bin.

Specify the three arrays of data to be created.

1. Click *Aggregate*...
2. Of the list in the dialog box, select sqkm, population, and birthrate. This can be done by holding down the *Ctrl* key while you select all three.
3. Click the left arrow (on the left of the "Group-By columns" panel) to shift these columns to the "Columns to Aggregate panel."
4. Set the first Index drop down menu to age_bin.
5. Set the Index2 drop down menu to year_bin.
6. Accept the default “sum” aggregation by clicking *OK*.
7. In the current columns list, note that the binned columns have now been removed because they were used as indices.
8. Note also that the sqkm, birthrate, and population columns have been replaced by columns with the same names suffixed by “[ ]” and prefixed by the type of aggregation performed.

Mapping the Data

1. In the Data Destination panel (right), under the Viz Tools tab, choose Map Visualizer from the option menu for which the default is Tree Visualizer. Note that the Visual Elements list changes to show only two items, Height **-** Bars and *Color **-** Bars.
2. Map the birthrate to the bar height.
   - In the current columns list, click sum_birthrate[ ].
   - In the requirements list, click the Height**-**Bars item.
   - Note that the height item in the Visual Elements list now indicates that birthrate data values are mapped to bar height in the Map Visualizer display.

     ```
     sum_birthrate[] -> Height-Bars
     ```
3. Similarly, map sum_population[ ] to bar color:

   ```
   sum_population[] -> *Color - Bars
   ```

Setting the Tool Options

To set the tool options in this example for the Map Visualizer:

1. In the Data Destination panel, click *Tool Options*. The Map Visualizer configuration options dialog box appears.
2. A geography file must be specified. Click *Find File* next to the Geography File text field. Choose the file */usr/lib/MineSet/mapviz/gfx_files/netherlands.regions.hierarchy.*
3. Choose Bar Legend On.
4. Specify a mapping from population values to colors:
   - In the "Color list to use" text field, green and gray already appear as defaults. To add a third color, blue, click the "+" to the right. When the color wheel appears, chose blue.
   - From the Mapping option menu, select Continuous (the default).
   - In the text field next to the Mapping option menu, type **`"0 1000000 2000000"`** (that's zero, one million, and two million).
5. Choose Color Legend On.

6. Enter a message:

   ```
   "%,.2f births per 1000 %,.0f total population",
        ‘sum_birthrate[]‘, ‘sum_population[]‘
   ```

   These two lines must be entered in the Message field as a single line. The variable names *sum_birthrate[ ]* and *sum_population[ ]* must be enclosed by single back-quotes because the bracket characters are special characters in the Map Visualizer grammar. In this example, the brackets are an integral part of the variable name, and the single back-quotes tell the Map Visualizer not to treat the brackets as special characters.

7. Click *OK* to accept the tool options.

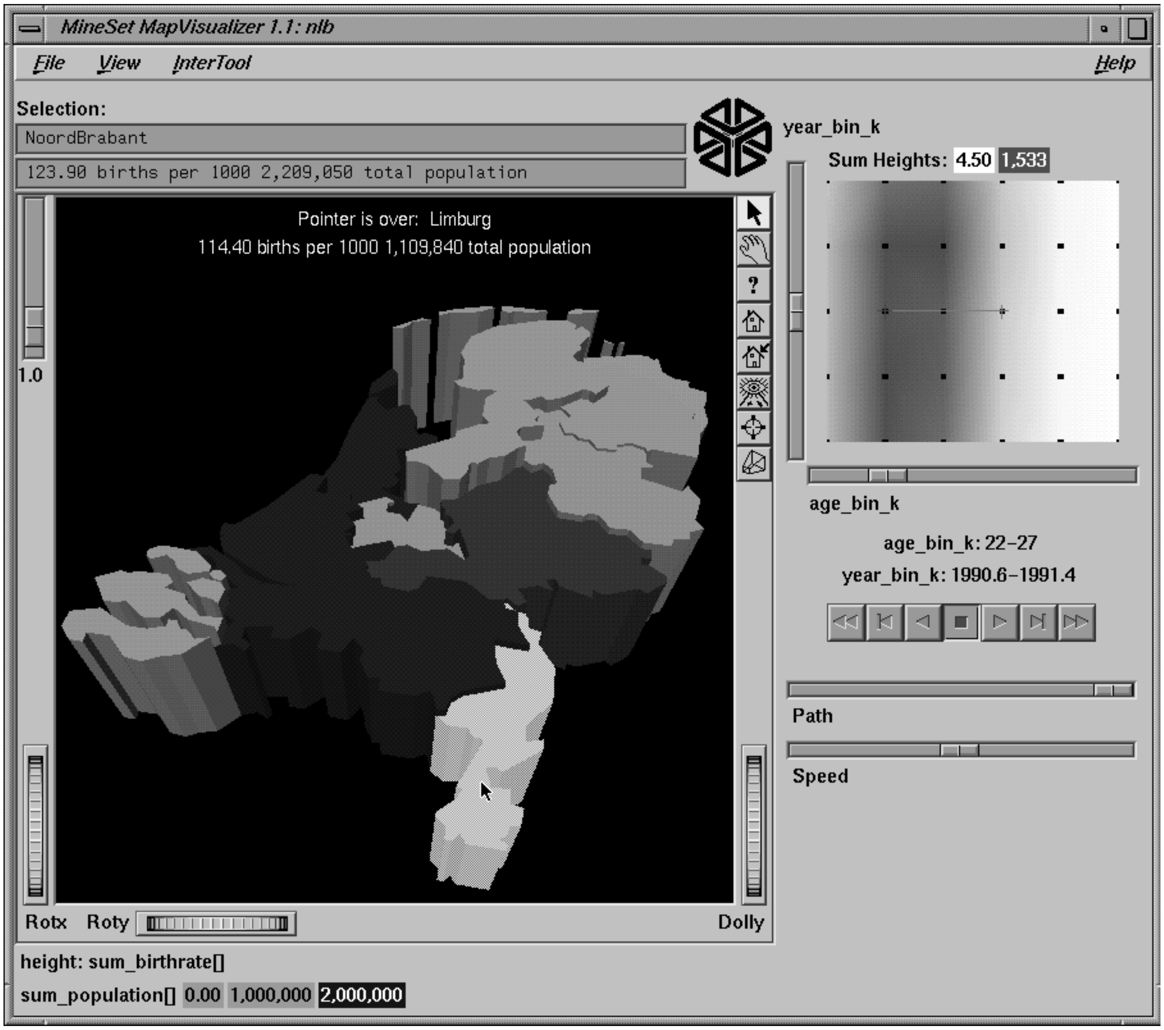
Invoking the Map Visualizer

To invoke the Map Visualizer, click *Invoke Tool*. The tool appears on the screen.

A dialog box briefly appears while the data is retrieved. About 420 records are initially retrieved from the database. The array aggregations specified above cause those records to be aggregated, creating a data file with 12 records (or lines), each containing a long array.

Things to Note

The following notes apply to the visualization shown in Figure H-1.



Selection:

NoordBrabant

123.90 births per 1000 2,209,050 total population

Pointer is over: Limburg

114.40 births per 1000 1,109,840 total population

Sum Heights: 4.50 1,533

age_bin_k: 22–27

year_bin_k: 1990.6–1991.4

height: sum_birthrate[]

sum_population[] 0.00 1,000,000 2,000,000

Figure H-1 Map Visualizer Applied to Netherlands Birth Data

- The red vertical strip in the summary slider shows that the birthrate for women in the Netherlands is greatest around age 27, and that birthrates by age do not change much over time.
- In the scroll area on the right side of the Map Visualizer window, move the X slider (age group values) left and right while the Y slider (year values) stays at “-1990.” Observe that the green region in the middle (Flevoland) shows high birthrates for young women. Green indicates a region with low population, gray indicates medium population, and blue indicates high population
- If you move the Y slider (year) up, there is an anomaly in the data for the state of Drenthe in the last two years. The birthrate stays low for all ages. The state is very high in the second to last year, and NULL (indicated by dark gray) in the final year. This is because the data for the last year was mistakenly dated as the second to last year. This is a case where the visualization helps point out an error in the data.
- If you click an individual region, the relevant information about the object appears at the top.

Scatter Visualizer Example Using Working Adults Data

This example uses the Scatter Visualizer to understand relationships among education level, age, race, income, and hours worked in a sample of working adults in the United States in 1994. This data was extracted from the census database at *http://www.census.gov/ftp/pub/DES/www/welcome.html*.

Transforming the Data

1. Choose the adult94 table from the Table dropdown menu.
2. Bin the age column.
   - Click *Bin Columns*. The binning dialog box appears.
   - Select age from the Select columns to bin at the top.
   - Under the User Specified Thresholds tab, check *Use Evenly spaced bins*.
   - Fill in range start: 20.
   - Fill in range end: 60.
   - Fill in bin size: 5. This specifies five-year increments.

     Note: This places different categories of people into different age bins; it does not track the same people over time.
   - Click *Apply*, then *Close*. A new column, age_bin, appears at the bottom of the list of Current Columns.
3. Click *Remove Column* for workclass, education, marital_status, relationship, race, sex, capital_gain, capital_loss, and native_country. You may select all of them while holding down the Ctrl key and then clicking *Remove Column* once.

4. Specify the three arrays of data to be created.
    - Click *Aggregate...*
    - In the dialog that appears, choose: final_weight, gross_income, education_num, and hours_per_week. Do this by holding down the Ctrl key and clicking each column.
    - Move these columns to the “Columns to aggregate” panel by clicking the left arrow to the left of the center panel.
    - Change the first Index drop down menu from “none” to “age_bin”. The age_bin column disappears because that will be the column used to index the aggregated columns on the left.
    - The four columns under the “Columns to aggregate heading” use the “sum” aggregation by default. Change this from “sum” to “average” for gross_income, education_num, and hours_per_week. Do this by selecting these three columns then checking the *Average* check box at the bottom, and unchecking the *Sum* check box.
    - Similarly change the aggregation of “final_weight” from *Sum* to *Count*.
    - Leave occupation as Group By.
    - Click *OK*.

Mapping the Data

1. Under the Viz Tools tab of the Data Destination panel, choose the Scatter Visualizer from the top menu.
2. Set the following mappings:

```
Axis 1 <- avg_grossincome[]
Axis 2 <- avg_education_num[]
Axis 3 <- avg_hours_per_week[]
Entity <- occupation (this allows filtering on this variable)
Entity-size <- count_final_weight[]
Entity-color <- occupation
Entity-label <- occupation
Summary <- count_final_weight[]
```

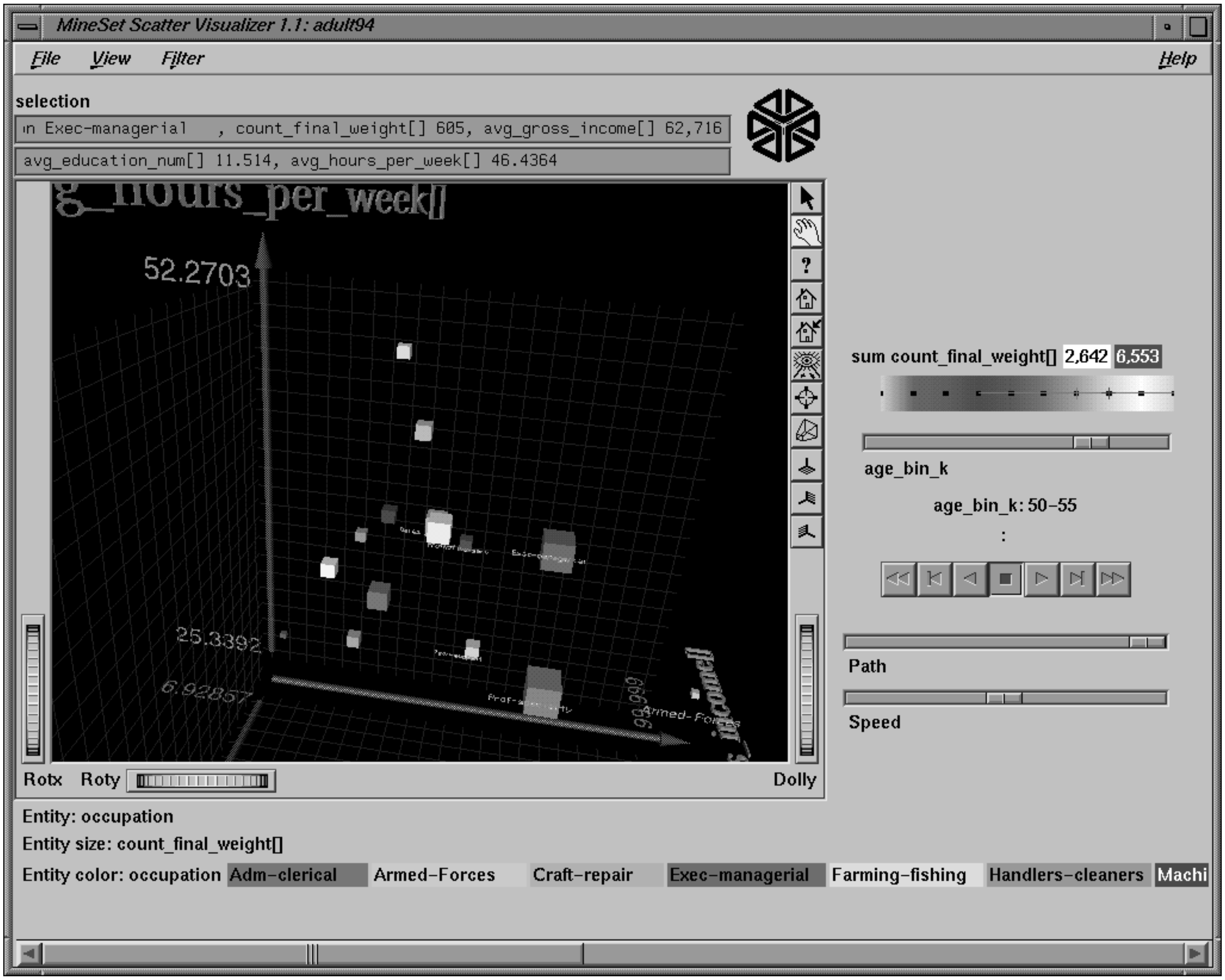
Invoking the Scatter Visualizer

To invoke the Scatter Visualizer, click *Invoke Tool*. A dialog box briefly appears while the data is retrieved (about 48,000 records, reduced to 15 records, each having columns that are arrays indexed by age_bin).

The Scatter Visualizer appears.

Things to Note

The following notes apply to the visualization shown in Figure H-2.



```
ın Exec-managerial   , count_final_weight[] 605, avg_gross_income[] 62,716
avg_education_num[] 11.514, avg_hours_per_week[] 46.4364
```

age_bin_k: 50–55

Entity: occupation

Entity size: count_final_weight[]

Entity color: occupation Adm–clerical Armed–Forces Craft–repair Exec–managerial Farming–fishing Handlers–cleaners Machi

Figure H-2 Scatter Visualizer Applied to Working Adults Data

The three axes show income, education level, and hours worked per week. The entity size is proportional to the number of records in that group. The entity size is the volume of the cube used to represent an aggregate. There is a distinct color for each occupation aggregate (entity). The colors at the bottom correspond to all the unique occupations represented in the data. The more volatile data points represent far smaller sample sizes. If the size of an entity goes to zero, it means there is no data in that particular age bin.

- The summary slider on the right shows count. Redder regions indicate ages where there is more data. Clearly there are fewer people in the work force that have ages less than 20 or greater than 50.
- Move the summary slider all the way to the left. The entities now represent ages below 20. In grasp mode, examine the entities. Note that although everyone's income is uniformly low, there are several occupations where the amount of education achieved is relatively high. The Machine operators/inspectors work the longest hours in this age group.
- Rotate so that the avg_education_num[ ] and avg_gross_income[ ] axes are parallel to the screen. Compare the cubes (entities) representing the occupations of Professional-Specialty and Executive-Managerial as you move the slider on the right. There are roughly the same number of each over time judging from the size. The ranks of both increase slightly as the population ages. Interestingly, those of Professional-Specialty have considerably more education than Executive-Managerial, but their income is equal.
- For any age group look at the relationship between education and salary. There is a roughly linear correspondence, but there are some outliers. For example, older people in the transport-moving business get paid a lot relative to their education level.
- Go back, using the Table History, and reaggregate. This time do a “group by” on sex and occupation. This yields twice as many entities. You can now compare salaries of men versus women in the same occupation. Or you might try "group by" on race and occupation to see if some ethnicities prefer certain occupations.

Tree Visualizer Example Using Working Adults Data

This example uses the Tree Visualizer to look at the same data just explored with the Scatter Visualizer. The resulting visualization looks much different and can be used to show different things.

Transforming the Data

1. Choose the adult94 table.
2. Delete the columns working_class, final_weight, education, occupation, relationship, capital_gain, capital_loss, and native_country using the *Remove Column* button.

Mapping the Data

1. Under the Viz Tools tab of the Data Destination panel, choose Tree Visualizer.
2. Set the following mappings:

```
Key - Bars <- race
Height - Bar <- gross_income
Height - Disk <- education_num
            (this is a numeric value indicating level of education)
Height - Base <- gross_income
Color - Bar <- hours_per_week
Hierarchy Root Level <- sex
Hierarchy level2 <- marital_status
```

Setting the Tool Options

1. Click *Tool Options*.
2. In the Bars column, set the following:
   - Check "Normalize heights across all levels" box.
   - Max / Scale heights: 4
   - Height Aggregation - Average
   - Check *Use Legend* for height.
   - Color list to use: Select 3 colors by repeatedly pressing the "+" and selecting desired colors (for example, blue, gray, red).
   - Color mapping: 30 40 50 (hours worked per week). This means the color blue indicates that the average number of hours worked per week for the people represented by an object of this color is 30, gray indicates 40 hours per week, and red 50.
   - Color Aggregation - Average
   - Check *Use Legend* for color.

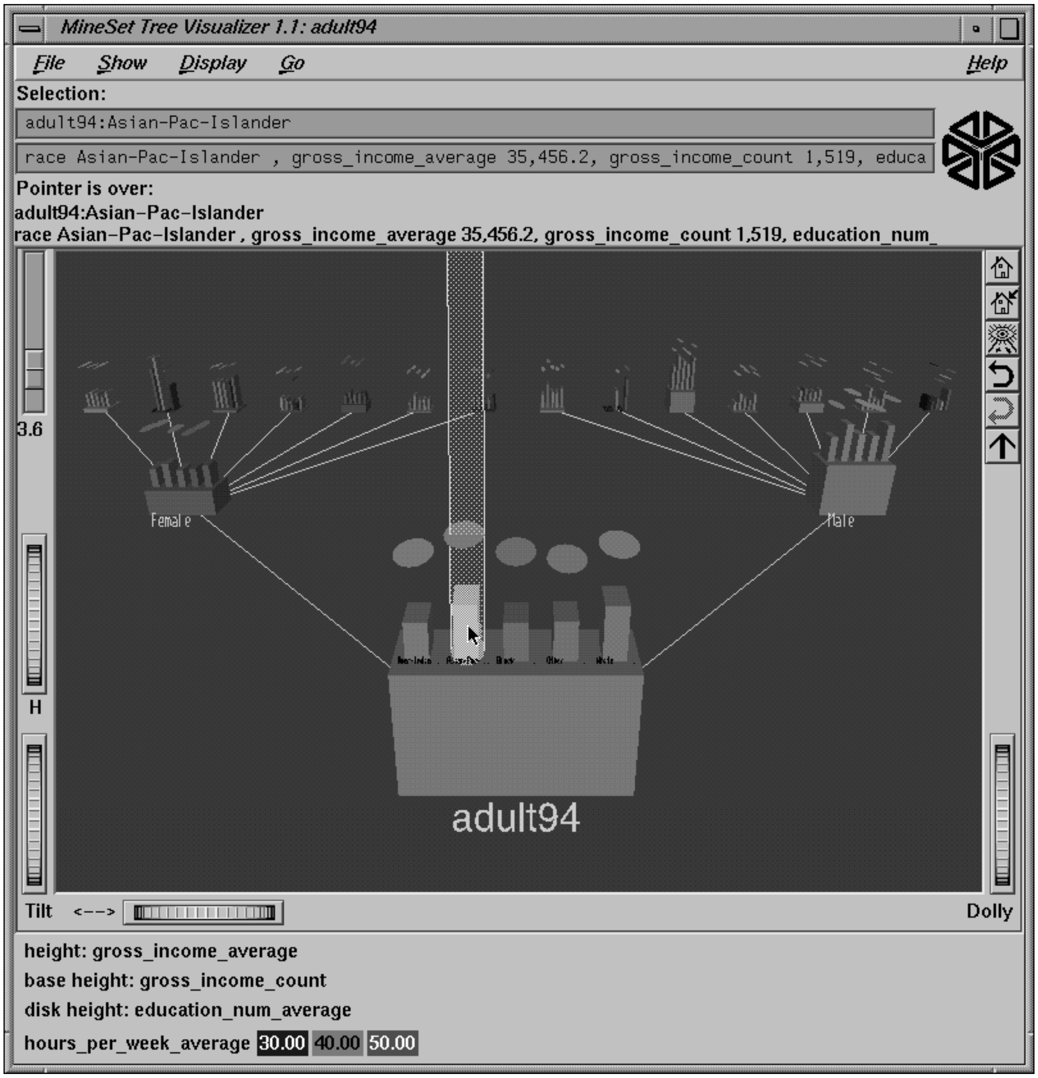
3. In the Node Bases column, set the following:
   - Check the “Normalize heights Across all levels” box.
   - Choose Height Aggregation - Count
   - Leave the color mappings empty. If no color mappings are specified, the node bases are colored the same as the bars.
   - Check *Use Legend* for the height.
4. In the Disks column, set the following:
   - Check the "Normalize heights across all levels" box.
   - Height Aggregation - Average.
   - Leave the color mappings empty. The same color mappings specified for the bars are used for the disks by default.
5. Accept the remaining defaults by clicking *OK*.

Invoking the Tree Visualizer

To invoke the Tree Visualizer, click *Invoke Tool*. The Tree Visualizer appears after the data is retrieved from the server.

Things to Note

The following notes apply to the visualization shown in Figure H-3.



Selection:

adult94:Asian-Pac-Islander

race Asian-Pac-Islander , gross_income_average 35,456.2, gross_income_count 1,519, educa

Pointer is over:

adult94:Asian-Pac-Islander

race Asian-Pac-Islander , gross_income_average 35,456.2, gross_income_count 1,519, education_num_

height: gross_income_average

base height: gross_income_count

disk height: education_num_average

hours_per_week_average 30.00 40.00 50.00

Figure H-3 Tree Visualizer Applied to Working Adults Data

There is a bar and a disk for each race at each node. The bars show average income and the disks show average education levels. The color indicates number of hours per week worked. Tendency toward blue means fewer hours worked and tendency toward red means more hours worked.

- Click the root node to see that White and Asian / Pacific-Islanders have the highest incomes. Although they have roughly equivalent incomes, Asians have higher education in general.
- At the next level, compare the male and female nodes. The female node is far more blue, indicating fewer hours worked. To see the numbers for the average work week of each sex, move the cursor over the base. The text output window at the top shows 42.4 hours / week for males and 36.4 hours for female. Similarly, you can also see numbers for individual bars by placing the cursor over one. Another thing to note is that the education level for females (shown by the disks) is much higher than their income (shown by the bars). (This is probably because they work fewer hours.) The same is not true for males. Another thing to note from the node bases is that there are far more males in the dataset than females (32,650 males compared with 16,192 females).
- Men who are Married-civilian-spouses earn considerably more than men who are divorced or separated.
- The bright red bar at Female:Married-AF-spouse:Asian / Pacific-Islander is caused by a single female “married Armed Forces spouse” of that ethnic type, who happens to work long hours and is highly paid. Similarly the red bar at Male:Married-AF-spouse:White represents 10 people of this type whose average hours worked per week is 54.
- There are 8 times more married men than divorced men, but only 2.6 times more married women than divorced women in the data.

Rules Visualizer Example Using Cars Data

This example uses a data mining tool, the Associations Rules Generator, to derive information (in the form of rules) from a table. It then uses the Rules Visualizer to display the results. The dataset contains statistics on cars made in the U.S., Europe, and Japan from 1970 to 1982.

Preparing the Input Data for the Association Rules Generator

1. Choose the cars table
2. Use Change Types to convert cylinders to **int**.
3. Click *Bin Columns*.
4. Hold down the Ctrl key and select every column but cylinders.
5. Under the Automatic Thresholds tab, check Group into: and enter 4 as the number of bins.
6. Accept the remaining default options (automatic discretization with respect to the label) by clicking *Apply*, then *Close*.
7. Under the Mining Tools tab in the Data Destination panel, choose the Associations tab. There are 3 steps to producing the files for the Rules Visualizer. Each corresponds to a section in this panel.

Creating a Binary File

To create a binary file:

1. Under Creating / Selecting a Binary File, click *Assoc Convert Options*.
2. Click *Map All* (the rightmost button), then click *OK*.
3. Click *Run Convert*.
4. The message `Binary File Created` appears. Click *OK* to dismiss it.

Creating a Rules File

To create a rules file:

1. Under Creating / Selecting a Rules File, click *Assoc Options*.
2. In the Association Rule Generation Options dialog, reduce the Prevalence to 0.5. Leave Predictability as 50. This means that rules that do not occur with at least a prevalence of 0.5 and predictability of 50 are not listed in the resulting rules file.
3. Click *OK*.
4. Click the *Run Assoc* button. A rules file is created.
5. The message `Rules file received from server` appears. Click *OK* to dismiss it.

Mapping Data to the Rules Visualizer

Map the following columns to requirements:

```
Height-Bars <- prevalence
Color-Bars <- pred_div_expected
```

The *pred_div_expected* variable means this ratio:

$$\frac{\text{P (the RHS occurs given that the LHS is present)}}{\text{P (the RHS occurs)}}$$

Thus, *pred_div_expected* is a measure of the increase of predictive power due to the presence of the left-hand side (LHS) rule.

Setting the Tool Options

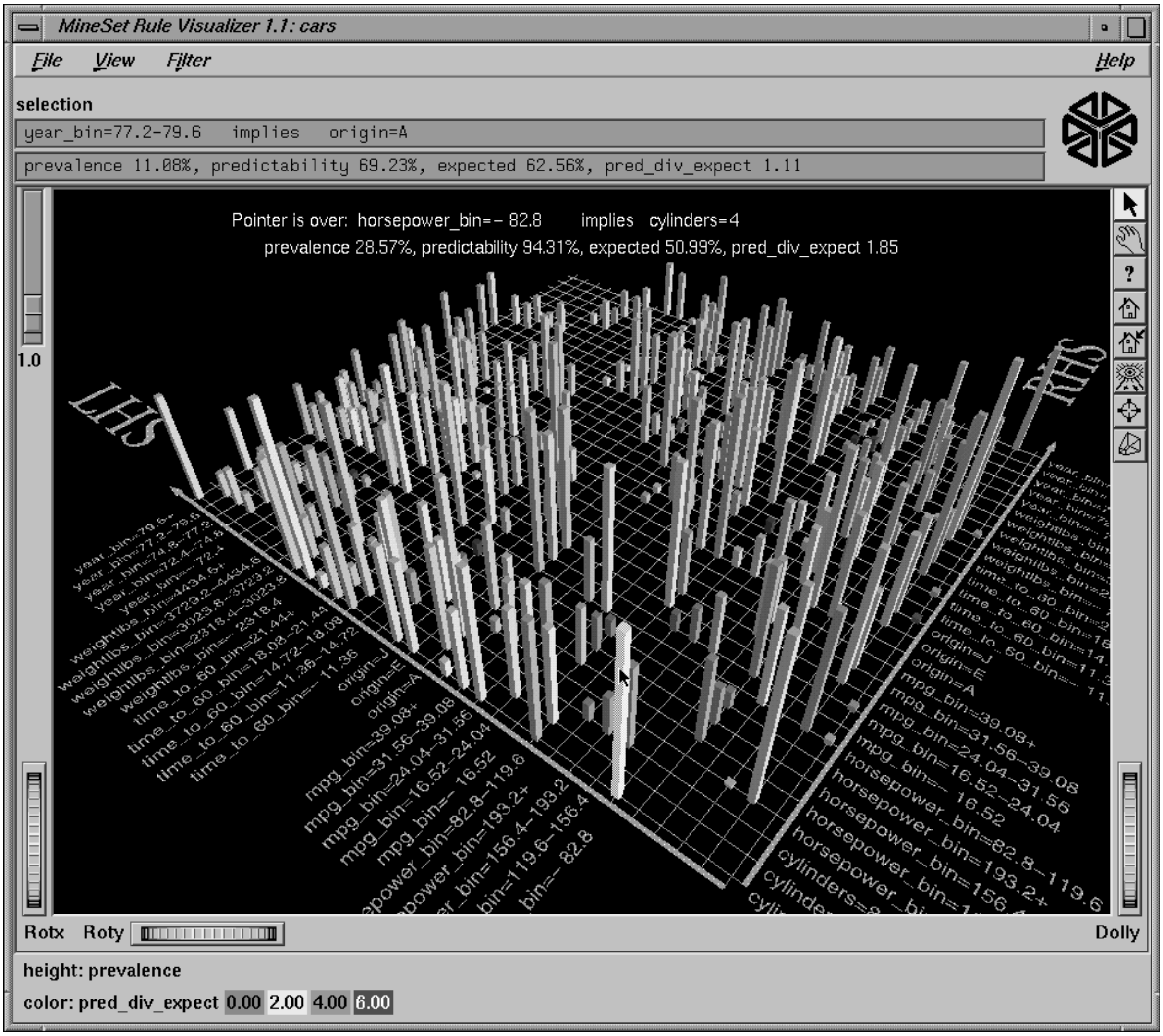
1. Click *RuleViz Options* at the bottom of the right panel.
2. For the bars, set the following color mapping:
   - color list: green, yellow, orange, red. If necessary, you may delete some of the default colors by using the middle mouse button to select a color then clicking the **-** (minus sign) button to the right of the color list.
   - Mapping, continuous: 0 2 4 6.
3. Make Item size 3, and the Hide distance 260. This increases the visibility of the text items.
4. Click *OK*.

Invoking the Rules Visualizer

To invoke the Rules Visualizer, click *Run RuleViz*. The Rules Visualizer appears.

Things to Note

The following notes apply to the visualization shown in Figure H-4.



year_bin=77.2-79.6   implies   origin=A

prevalence 11.08%, predictability 69.23%, expected 62.56%, pred_div_expect 1.11

Pointer is over: horsepower_bin=– 82.8   implies   cylinders=4
prevalence 28.57%, predictability 94.31%, expected 50.99%, pred_div_expect 1.85

height: prevalence

color: pred_div_expect 0.00 2.00 4.00 6.00

Figure H-4 Rules Visualizer Applied to Cars Data

- There are two very short red bars in the back. These show that it is very rare for cars with 3 or 5 cylinders to be manufactured. However, when it happens, those with 3 cylinders are made in Japan, and those with 5 cylinders are made in Europe.
- To find out what a particular left-hand side implies use filtering:
  - From the Filter pulldown menu, choose the rule(s) you are interested in from the LHS.
  - Click *Filter* at the bottom.

  For example, choose MPG < 16 (not fuel efficient). This implies:

  - Eight cylinders 95% of the time.
  - Weight is greater than 3,300 lbs (heavy) 96% of the time
  - For 93% of the time, the horsepower is >124.
  - Acceleration is faster than 14 seconds with 70% likelihood. Consider that out of the whole population of cars, acceleration is faster than 14 seconds only 25% of the time (the expected predictability).
  - All these very low gas mileage cars are made by the U.S.
- Cars that have 3 cylinders always have fast acceleration (<14 seconds) whereas cars that have 8 cylinders have fast acceleration only 70% of the time.
- High horsepower implies quick acceleration.
- Europe, in addition to being the only producer of 5-cylinder cars, also makes many 4-cylinder cars. 90% of all the cars produced in Europe are 4-cylinder. The total occurrence of 4-cylinder cars in Europe, the U.S., and Japan, is 51%. Europe also makes more light cars than the U.S. or Japan (<2400 lbs 60% of the time, compared with the expected 33% of all cars in the dataset).
- To find out what implies good gas milage, filter the RHS to MPG>31 and choose Select All on the LHS. One of the rules in the result shows plainly that cars made after 1979 were much more fuel efficient.

Decision Tree Visualizer Example Using Mushroom Data

This example uses a data mining tool, the Decision Tree Classifier, to classify records according to a label. It then uses the Tree Visualizer to display the Classifier. The dataset contains information on 5882 sample mushrooms.

Generating the Decision Tree

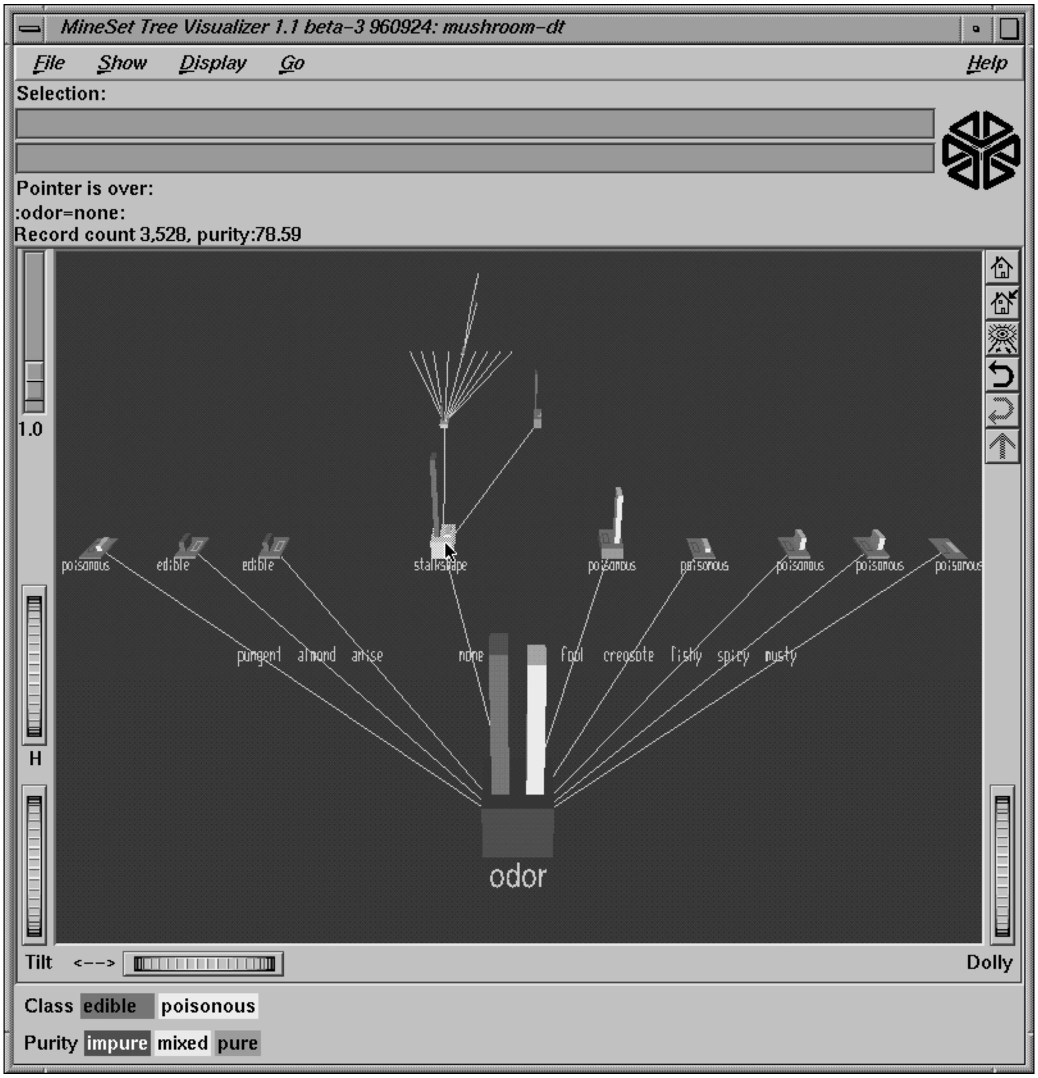
1. Choose the mushroom table.
2. Under the Mining Tools tab, choose the Classifiers subtab.
3. Using the Algorithm dropdown menu, choose Decision Tree (it is the default).
4. Using the Discrete Label dropdown menu, choose the column edibility (it is the default). The last column in the table always appears as the default label. If no discrete label is present, you must create one.
5. Click *GO!*.

   A dialog showing statistics appears, and on top of this, the Decision Tree Visualizer appears.

   The Tree Visualizer appears after the data is retrieved from the server.

Things to Note

The following notes apply to the visualization shown in Figure H-5.



MineSet Tree Visualizer 1.1 beta-3 960924: mushroom-dt

Selection:

Pointer is over:
:odor=none:
Record count 3,528, purity:78.59

Figure H-5 Decision Tree Classifier Applied to Mushroom Data

At each level in the tree a split is made on various attributes in order to separate the edible mushrooms from the poisonous ones. At the leaves the nodes should be 100% pure (green). This means they contain either all edible or all poisonous mushrooms. Impure nodes, where there is a mix, are colored red.

- Odor is the best attribute to use when trying to predict whether or not a mushroom is edible.
- The only time you cannot determine edibility just by smell is when it has none. In that case, stalk shape is the next best attribute to use. Even though the stalk shape node is showing green (almost all the mushrooms here are edible), there are still some poisonous mushrooms present, hence you should look at other attributes before feeling safe about eating one that has no odor.
- Try running the classifier again after removing the odor column. Note that the tree looks much different. Now gill size is the best attribute to use for predicting edibility. Note that the next best attribute is different based on the value at the root. If the mushroom has broad gill size, then spore print color is used; if the mushroom has narrow gill size, then gill spacing is the next attribute to consider.
- Try using feature selection to determine which combination of attributes gives the best results. This can be done by selecting the Feature selection tab under mining tools in the Data Destination panel. Specify the number of attributes you want to use. This tries all possible combinations of attributes to optimize for accuracy of classification.

  For example, if you enter 6, it attempts to find which combination of six attributes provide the best accuracy for predicting the label. Fewer attributes are returned if 100% accuracy is reached with fewer than the specified number of attributes. For this example, assuming odor has been removed, the most important attributes are sporeprintcolor, gillsize, stalkroot, and bruises. Note that the accuracy is 100%; this is better than what would have been achieved had all attributes been used. Try building a decision tree based on these results.

Evidence Visualizer Example Using Mushroom Data

This example uses a data mining tool, the Evidence Classifier, to classify records according to a label. It then uses the Evidence Visualizer to display the Classifier. The dataset contains information on 5882 sample mushrooms.

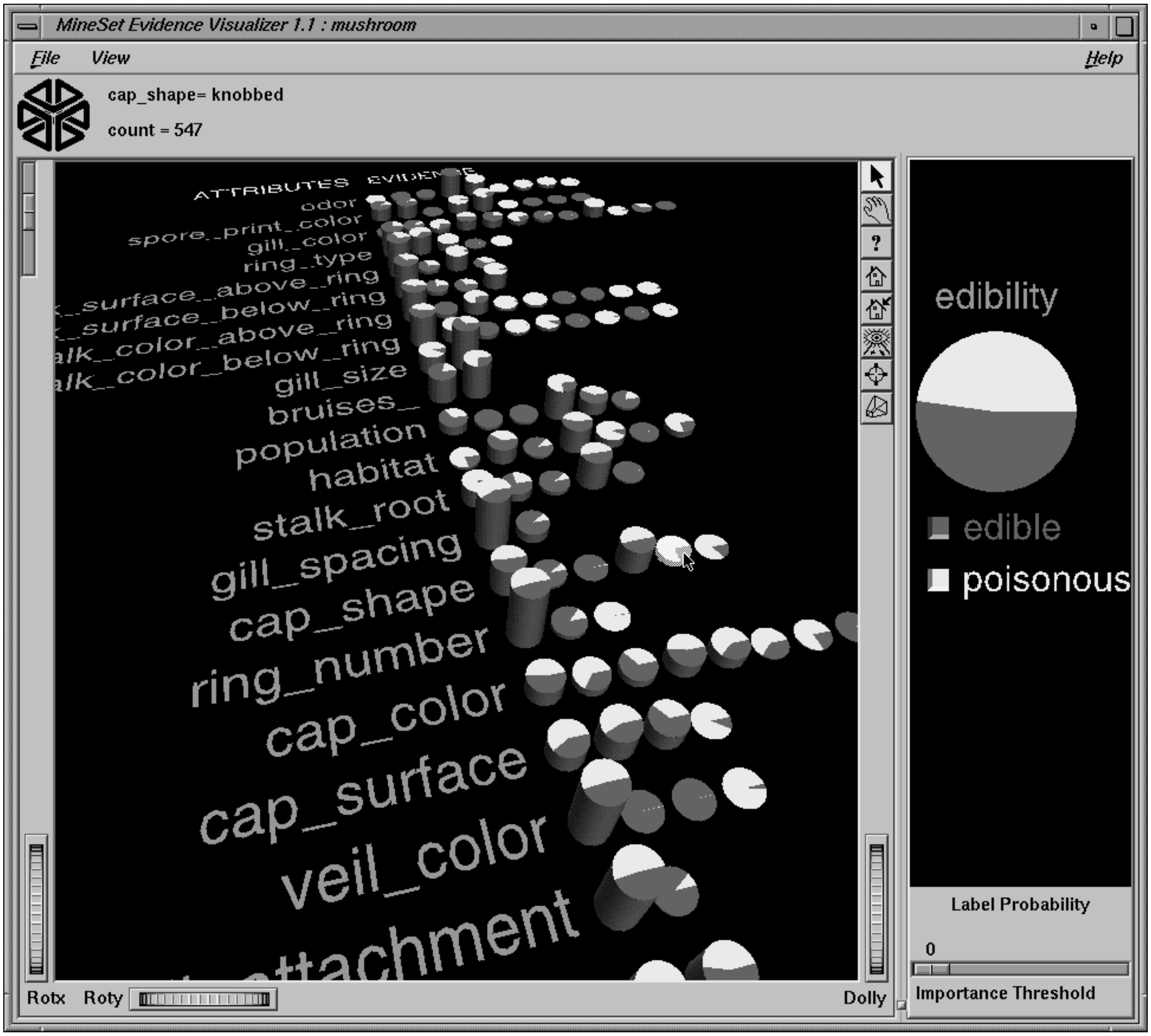
Generating the Evidence Classifier

1. Choose the mushroom table
2. Under the Mining Tools tab, choose the Classifiers subtab.
3. Use the Algorithm dropdown menu to select Evidence.
4. Using the Discrete Label dropdown menu, choose the column edibility (it is the default). The last column in the table always appears as the default label. If no discrete label is present, you must create one by changing types or binning.
5. Click *GO!*.

   A dialog showing statistics appears, and on top of this the Decision Tree Visualizer appears. You can see in the statistics window that the classifier is created using 5416 training instances. The remaining 2708 instances are “held out,” and used for checking accuracy. When the classifier is used to predict which class these 2708 instances fall into, only 92 are classified incorrectly. This implies the accuracy to be [95.85-97.22] with 95% confidence.

Things to Note

The following notes apply to the visualization shown in Figure H-6.



cap_shape= knobbed

count = 547

Figure H-6 Evidence Classifier Applied to Mushroom Data

The viewer on the left shows the effect that each value of each attribute has on the likelihood of edibility. The display on the right shows the proportion of each label value in the data (in this case the proportion of edible mushrooms is roughly equal to the amount that are poisonous.)

- The list of attributes (columns) has been sorted by importance (usefulness in predicting the label). Odor appears at the top of the list. As we saw with the decision tree, odor is the best attribute to use when trying to predict whether or not a mushroom is edible.
- The height of each pie corresponds to the number of records that have that attribute value. The sum of the heights in each row is constant. The heights of the pies in each attribute row form a histogram (that is, they show the way the records are distributed over the values for each attribute). Moving the mouse over a pie shows the value and (count) at the top of the screen.
- Since the Evidence Classifier assumes all attributes to be independent, it considers sporeprintcolor to be the next most important attribute for determining edibility. Note that the decision tree showed that if the mushroom has no odor, then stalk shape should be used. The reason the decision tree made that decision was because it was considering only those records with no odor.
- Click the two values of gillsize to see how they affect the probability distribution in the big pie, on the right. If the gill size is narrow, the mushroom is very likely to be poisonous. If the gill size is broad, then the evidence tilts significantly toward edible.
- When pies on the right-hand side are selected, their effect is accumulated by multiplication. The selected pies are multiplied. The result then is multiplied by the prior probability distribution; the outcome of this is the expected distribution (shown on the right). If a pie on the left has all equal-sized slices, selecting it has no effect on the pie on the right.

- Since veiltype (at the bottom of the list) has only one binned value, it is not used by the classifier. If you move the Importance slider slightly to the right, the attributes at the bottom start to disappear because they have low importance values.
- Try using feature selection (under the mining tools tab in the Data Destination window) to determine which combination of attributes give the best results. It is possible to actually improve accuracy by eliminating some of the attributes.
- To see a characterization of poisonous mushrooms, click the button next to the label value poisonous on the right. All the values with high bars indicate strong evidence for poisonous. Moving the mouse over the values shows the probability that a mushroom has that value, given that it is poisonous.

Appendix I

Further Reading and Acknowledgments

Some datasets were taken from the UCI repository (Merz, C. J., and Murphy, P. M. (1996). UCI Repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science) found at *http://www.ics.uci.edu/~mlearn/MLRepository.html*.

Further reading

A general and easy-to-read introduction to machine learning is Weiss, S. M., and C. A. Kulikowski. *Computer Systems that Learn*. San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1991.

An easy-to-read introduction to decision tree induction is Quinlan, J. R. *C4.5: Programs for Machine Learning*. Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1993.

An excellent book on decision trees from a statistical perspective is Breiman, L., J. H. Friedman, R. A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

A good edited volume of machine learning techniques is Dietterich, T. G. and J. W. Shavlik (Eds). *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc., 1990.

A summary of accuracy estimation techniques is given in Kohavi, R. “A study of cross-validation and bootstrap for accuracy estimation and model selection.” In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, edited by C. S. Mellish. Morgan Kaufmann Publishers, Inc., 1995.

An excellent introduction to the Evidence Classifier (Naive-Bayes) is Kononenko, I. (1993). Inductive and bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, pp. 7:317-337.

A good reference to a paper explaining that no classifier can be "best" is Schaffer, C. A conservation law for generalization performance. In *Machine Learning: Proceedings of the Eleventh International Conference*, 259-265. Morgan Kaufmann Publishers, Inc., 1994.

A general comparison of algorithms and descriptions is provided in Taylor, C., D. Michie, and D. Spiegalhalter. *Machine Learning, Neural and Statistical Classification*. Paramount Publishing International, 1994.

Further Readings About the Evidence Inducer

The following paper describes the wrapper method used to select the features for the Evidence Classifier:

- Kohavi, R., Sommerfield, D. (1995). Feature Subset Selection Using the Wrapper Model: Overfitting and Dynamic Search Space Topology. The First International Conference on Knowledge Discovery and Data Mining, pp. 192-197.

The following paper describes the Laplace correction option:

- Cestnik, B. (1990). Estimating Probabilities: A crucial Task in Machine Learning. Proceedings of the Ninth European Conference on Artificial Intelligence, pp. 147-149.

The following paper describes the Evidence Classifier (Naive-Bayes):

- Langley, P., Iba, W., Thompson, K. (1992). An Analysis of Bayesian Classifiers. Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 223-228.

The following books describe the Evidence Classifier:

- Good, I. J. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, 1965.
- Duda, R., Hart, P. *Pattern Classification and Scene Analysis*, Wiley, 1973.

The following paper shows that while the conditional independence assumption can be violated, the classification accuracy of the evidence classifier (called Simple Bayes in this paper) can be good:

- Domingos P., Pazzani M (1996). Beyond independence: conditions for the optimality of the simple Bayesian classifier. *Machine learning, Proceedings of the 13th International Conference* (ICML '96), pp. 105-112.

Acknowledgments

The iris database (described in Chapter 8, "MineSet Inducers and Classifiers") was originally used in Fisher, R. A. 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7(1):179-188. It is a classical problem in many statistical texts.

The breast cancer database was obtained from Dr. William H. Wolberg, L. Mangasarian, and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News* 23(5):1 & 18. University of Wisconsin Hospitals, Madison, September 1990.

The data for the mushroom sample file comes from: *Audubon Society Field Guide to North American Mushrooms*. New York: Alfred A. Knopf, 1981.

The data on congressional voting was taken from the *Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL,* Congressional Quarterly Inc.: Washington, D.C., 1985.

Index

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-3214-002.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 415-965-0964
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389