Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

FERRAMENTAS DE VISUALIZAÇÃO DE DADOS DO MineSet™

ROBSON B. T. DE OLIVEIRA
SOLANGE O. REZENDE

Nº 71

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Mar./1998

Ferramentas de Visualização de Dados do **MineSet™**[1]

Robson Butaca Taborelli de Oliveira

Solange Oliveira Rezende[2]

Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística

São Carlos
Março de 1998

[1] MineSet™ é uma marca registrada da Silicon Graphics, Inc.
[2] Trabalho realizado com apoio do CNPq e FAPESP.

Conteúdo

Índice de Figuras

1. Introdução

A visualização de dados, contidos em uma base de dados, é um poderoso mecanismo para a exploração e compreensão dos dados. A visualização de dados permite ao usuário encontrar características e tendências existentes dentro da base de dados, as quais seriam difíceis, ou mesmo, impossíveis de descobrir caso a análise dos dados fosse feita em cima de tabelas com milhares de linhas e colunas [2].

Para se conseguir uma boa visualização de dados, às vezes, é necessário fazer algumas operações sobre os dados, preparando-os de tal forma, que possam ser melhor visualizados. Tais operações podem ser realizadas utilizando técnicas de **Data Mining** ou **KDD** (*Knowledge Discovery in Data Base*) [3]. E, neste relatório técnico, será apresentado um aplicativo, **MineSet™**, que incorpora técnicas de **Data Mining** e possui algumas ferramentas de visualizações integradas.

Este trabalho fornece as informações necessárias para que o usuário consiga realizar visualizações de dados sem a necessidade de usar diretamente o **MineSet™**, mas sim, utilizando apenas as ferramentas de visualização de dados **Scatter Visualizer**, **Tree Visualizer** e **Map Visualizer**. Contudo, uma visão geral sobre o software e suas propriedades também serão apresentados, já que estas ferramentas de visualização estão integradas ao **MineSet™**. E, ainda, faz-se necessário que o usuário tenha uma idéia das capacidades existentes no aplicativo, de forma a capacitá-lo à extrair todos os benefícios que podem ser conseguidos desta ferramenta na exploração e entendimento de dados.

O relatório não se apresenta como um manual completo de referência da ferramenta **MineSet™**, mas sim, conduz o usuário a um primeiro contato com a ferramenta, e principalmente das ferramentas de visualização integradas. Caso o leitor deseje obter uma referência mais completa da ferramenta **MineSet™** da Silicon Graphics deverá então recorrer as páginas dos manuais fornecidos junto com o software, ou mesmo, na ajuda on-line embutida na ferramenta.

Para uma melhor compreensão de como pode ser utilizado o aplicativo, este relatório foi dividido em duas partes. A primeira parte, seção 2, trata da ferramenta completa, ou seja, dá uma visão geral do **MineSet™** e das suas ferramentas integradas, conseguindo desta forma situar o usuário das propriedades existentes na ferramenta. E na segunda e maior parte deste relatório, seções 3, 4, 5, 6, 7, 8 e 9, será mostrado como realizar visualizações independentemente do **MineSet™**, onde serão detalhados os procedimentos que deverão ser seguidos e os arquivos que deverão ser criados para a realização da visualização.

2. MineSet

O software **MineSet™** é formado por um conjunto de ferramentas integradas, que permitem a realização de mineração e visualização de dados contidos em banco de dados, ou mesmo em um simples arquivo com formato específico. Estas ferramentas permitem "garimpar" e mostrar o resultado graficamente, de tal forma que auxilie o usuário a obter uma melhor exploração, compreensão e visualização dos dados. As ferramentas de mineração descobrem padrões construindo modelos que podem ser visualizados pelas ferramentas de visualização. Ainda, as ferramentas de visualização podem ser aplicadas diretamente aos dados e, por isso, uma melhor observação dos dados pode ser conseguida. Estas ferramentas permitem ao usuário adquirir um entendimento mais profundo e intuitivo dos dados, fazendo com que o usuário descubra padrões escondidos e tendências importantes [2].

Todas estas ferramentas de visualização possuem interfaces visuais tridimensionais e interativas, que permitem ao usuário a manipulação de objetos na tela, assim como animações. Tal habilidade para visualizar e pesquisar padrões complexos nos dados é um excelente auxílio de apoio à tomada de decisão.

Três componentes básicos constituem a ferramenta completa **MineSet™**:

- Módulo de controle centralizado, consistindo de uma ferramenta com interface gráfica chamada **Tool Manager**, e um processo chamado de **DataMover**, que roda no servidor;

- Mineração de banco de dados, constituída de quatro ferramentas:
  - Gerador de Regras de Associação
  - Classificador e Indutor de Árvore de Decisão
  - Classificador e Indutor de Evidência
  - Importância da Coluna

- Ferramentas de visualização, que permitem a visualização dos dados utilizando diferentes metáforas visuais:
  - Tree Visualizer
  - Scatter Visualizer
  - Map Visualizer
  - Rule Visualizer
  - Evidence Visualizer

As seções seguintes descrevem brevemente cada componente do **MineSet™** .

2.1 Tool Manager

Cada uma das ferramentas para mineração e visualização de dados podem ser configuradas e iniciadas através de uma interface gráfica ao usuário, a **Tool Manager**, que inicia individualmente as ferramentas e pode especificar:

- o local de onde os dados serão analisados:
  - de um banco de dados
  - de um arquivo
- o conjunto de transformações feitas através de:
  - ferramentas de mineração — que encontram padrões nos dados
  - variáveis de "binning" — que agregam valores em grupos
  - arrays — tomando os valores de uma coluna e colocando-os em um array indexado em outra coluna
  - colunas distribuídas — fazer duas ou mais colunas a partir de uma única coluna de valores, distribuídas por valores discretos de uma outra coluna
  - removendo colunas
  - adicionando novas colunas — em função de colunas existentes
- o tipo de visualização pretendida:
  - hierárquica (Tree Visualizer)
  - mapa (Map Visualizer)
  - relação entre variáveis independentes (Scatter Visualizer)
  - regras associadas (Rule Visualizer)
  - evidências (Evidence Visualizer)
- mapeamento específico dos valores de dados em elementos visuais como cores, barras, alturas, etc.
- opções não relacionadas aos dados como cores de fundo, espaçamento entre as malhas e tamanho dos rótulos

Um exemplo da janela da **Tool Manager** é mostrado na figura 1.

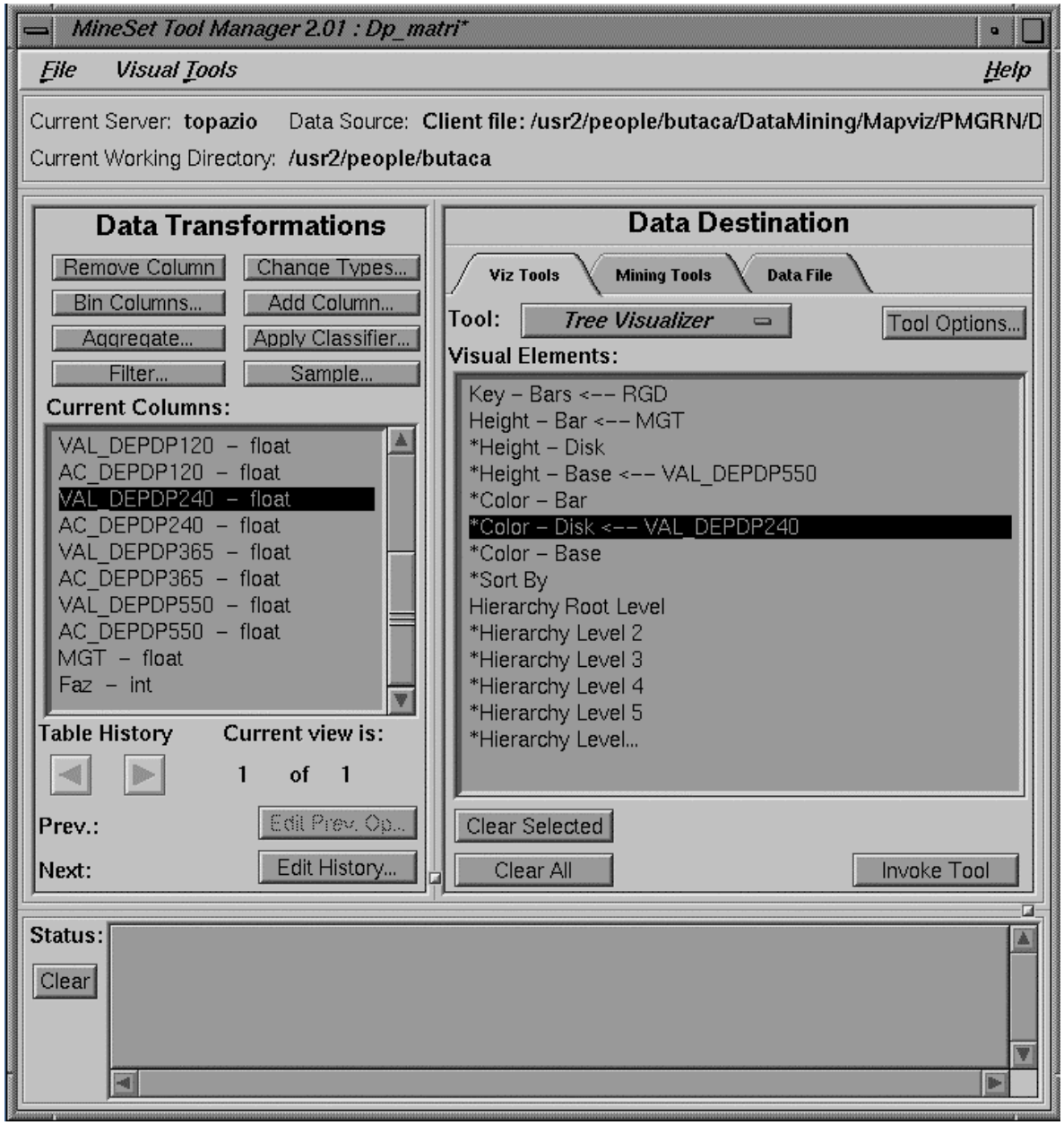

Figura 1. Janela inicial do Tool Manager

2.2 DataMover

O DataMover é um processo que é rodado assim que o aplicativo **MineSet™** é iniciado. Este processo permite:

- fazer a conexão com base de dados ou arquivos, e recuperar dados;
- invocar as ferramentas de mineração de dados;
- realizar manipulação de dados;

- retornar os dados ao **Tool Manager** para distribuição para as ferramentas de visualizações;
- armazenar os dados em arquivos no servidor ou no cliente para operações futuras.

2.3 Gerador de Regras de Associação

O Gerador de Regras de Associação processa um arquivo de entrada gerando um arquivo de saída, que consiste de regras. Estas regras indicam a freqüência com que um item ocorre em um registro em companhia de outro item. A associação é quantificada por três elementos:

- a previsibilidade da regra, que quantifica quão freqüentemente X e Y ocorrem juntos como uma fração do número de registros em que X ocorre. Por exemplo, dado que alguém comprou leite, com que freqüência ele também compra pão;
- a prevalência da regra, que quantifica quão freqüentemente X e Y ocorrem juntos em um arquivo como uma fração do número total de registros. Por exemplo, quão freqüentemente leite e pão são comprados juntos;
- a previsibilidade esperada, que dá uma indicação de qual deveria ser a previsibilidade se não existir nenhuma relação entre os itens no registro. Por exemplo, quão freqüente é comprado pão, indiferentemente se foi comprado leite também.

2.4 Classificador e Indutor de Árvores de Decisão

O Classificador de Árvore de Decisão classifica dados de acordo com um conjunto de atributos, fazendo uma série de decisões baseadas nestes atributos. Aplicando este classificador para determinar o perfil de alguém com credibilidade no mercado, por exemplo, uma árvore de decisão pode determinar se alguém que possui uma casa, um carro que custa entre $10,000 e $20,000, e tem dois filhos, pode ser considerado um bom risco de crédito.

O Indutor de Árvore de Decisão gera um Classificador de Árvore de Decisão a partir de um conjunto de treinamento, ou seja, um conjunto de dados classificados pelo usuário previamente. A estrutura da árvore de decisão do classificador é mostrada através do Tree Visualizer, com cada decisão representando um nó da árvore mostrada. A representação gráfica pode ajudar o usuário a entender o algoritmo de classificação, bem como prover uma valiosa compreensão dos dados. Finalmente, o classificador pode ser utilizado para classificar dados não classificados.

2.5 Classificador e Indutor de Evidências

O Classificador de Evidência classifica dados examinando as probabilidades de um resultado específico ocorrer baseado em um atributo. Por exemplo, pode ser determinado se alguém que possui um carro que custe entre $15,000 e $25,000 possua 80% de chances de ser um bom risco de crédito, e 20% de chances de não ser. O classificador prediz a classe com a maior probabilidade baseada em um simples modelo de probabilidade.

O classificador é primeiramente gerado através de um conjunto de treinamento, similar ao Classificador de Árvore de Decisão. A análise dos dados é mostrada utilizando-se o Evidence Visualizer, que mostra os dados em forma de partes de gráficos (pizza) ilustrando as diferentes probabilidades. Esta representação gráfica pode ajudar o usuário a entender o algoritmo de classificação, bem como prover uma valiosa compreensão dos dados. E também, o classificador pode ser utilizado para classificar dados não classificados.

2.6 Importância de Coluna

A Importância de Coluna determina quão importante os vários atributos são para determinar o valor de um determinado atributo. Por exemplo, o usuário pode desejar que sejam selecionados, automaticamente, os três melhores atributos que ajudem a determinar se alguém é um bom risco de crédito. O sistema pode selecionar renda, propriedades, e custo do carro. Estes atributos podem então ser mapeados para os eixos do **Scatter Visualizer**, ou usados na hierarquia do **Tree Visualizer** podendo ser visualizados.

A Importância de Coluna possui um modo avançado que fornece capacidades adicionais. Primeiro, permite que o usuário determine quão importante cada atributo é. Por exemplo, o usuário poderia determinar se tanto renda quanto salário têm importâncias iguais em determinar risco de crédito. Apesar de renda parecer ser um pouco melhor em determinadas importâncias, talvez o usuário prefira utilizar o salário porque é mais fácil de obter-se. Segundo, uma vez que o usuário explicitamente escolheu um atributo, pode ser determinado quais outros atributos são importantes em conjunção com o escolhido. Por exemplo, se o usuário tiver escolhido salário ao invés de renda, o custo da casa pode se tornar mais importante que possuir uma casa, e renda teria uma importância muito menor.

2.7 Tree Visualizer

O **Tree Visualizer** ajuda o usuário a analisar os dados que possuam um relacionamento hierárquico. Esta ferramenta de visualização provê uma capacidade interativa tal que permite examinar as relações entre dados em

diferentes níveis hierárquicos. Por exemplo, o **Tree Visualizer** pode ser usado para examinar a linha de produtos de uma companhia, mostrando graficamente a contribuição de cada produto para o rendimento total da companhia. Cada ramo da hierarquia mostra informação em um nível de detalhes crescente, quebrando os rendimentos por linhas de produtos e, eventualmente, em produtos individuais. Um outro exemplo da utilização do **Tree Visualizer** seria para mostrar os rendimentos das vendas da companhia, mostrando um total de vendas da companhia nacionalmente, bem como os subtotais nas regiões e demais níveis. As capacidades de filtragem e buscas do **Tree Visualizer** permitem que o usuário focalize em um elemento específico dos dados e realize consultas específicas.

O **Tree Visualizer** também é usado para verificar os resultados do Classificador de Árvores de Decisão, com cada decisão sendo representada por um nó separado na árvore. Cada nó mostra também barras que ilustram como o classificador classifica os dados baseados nas decisões acima daquele ponto.

2.8 Map Visualizer

O **Map Visualizer** permite ao usuário visualizar os relacionamentos existentes dos dados em áreas correlatas de forma geográfica. Por exemplo, o usuário pode visualizar diferentes áreas de um país, mostrando o impacto relativo de um programa de marketing. Ainda, existem capacidades que permitem ao usuário focalizar em certas regiões e realizar uma análise mais detalhada em elementos geográficos menores. Por exemplo, pode ser analisado como um ou mais produtos estão sendo vendidos em áreas geográficas diferentes. Uma propriedade poderosa de animação, unida com a capacidade de conectar diferentes visões do mesmo dado ou dados relacionados, permite uma comparação rápida e análises diferenciadas. Esta ferramenta permite que o usuário examine visualmente padrões nos dados que são difíceis de detectar quando os dados são mostrados em uma forma tabular bidimensional.

2.9 Scatter Visualizer

O **Scatter Visualizer** permite que o usuário examine o comportamento dos dados através de diferentes dimensões. Os dados são mostrados em uma malha representando até três dimensões. Dimensões extras podem ser mapeadas para os tamanhos, cores e rótulo de cada entidade mostrada, e mais duas dimensões independentes podem ser associadas como dimensões dinâmicas. Um *slider* pode ser utilizado para selecionar valores específicos ao longo dessas dimensões, ou um caminho pode ser traçado através dessas dimensões para fazer animações do gráfico. Durante a travessia do caminho ocorrem mudanças na tela que refletem as variações das variáveis independentes.

2.10 Rule Visualizer

O **Rule Visualizer** representa visualmente os resultados do Gerador de Regras de Associação. Tal ferramenta permite análises detalhadas de dados auxiliando o usuário a examinar relacionamentos entre os elementos através de novas formas. E deste modo, o usuário pode descobrir os relacionamentos que diferem significativamente do que o usuário possa ter esperado; isto, em troca, pode levar a importantes descobertas sobre os dados ou os processos por trás dos dados. Esta capacidade de visualização da ferramenta permite ao usuário descobrir padrões adicionais de ocorrência entre os elementos dos dados. Por exemplo, o usuário pode usar a analise dos produtos vendidos nas últimas promoções de vendas para guiar a campanha de propaganda para o próximo período de vendas.

2.11 Evidence Visualizer

O **Evidence Visualizer** representa visualmente os resultados do Classificador de Evidências. Esta ferramenta mostra os dados em formas de partes de gráficos (pizzas) representando como os vários atributos contribuem para a decisão. Por exemplo, ele pode mostrar que possuir uma casa contribui para ser um bom risco de crédito. Selecionando os pedaços de gráficos, pode ser mostrado o efeito gerado da combinação de vários atributos sobre o resultado.

2.12 Cenário Básico de Execução do MineSet™.

Cada ferramenta do **MineSet™** é iniciada, configurada e rodada de uma maneira consistente. Será mostrado em seguida a seqüência de passos que o usuário deve seguir para uma execução típica do **MineSet™**. Vale ressaltar que alguns dos passos poderão ser omitidos de acordo com a necessidade do usuário. Os passos seguem abaixo:

Passo 1.
Começar o **Tool Manager**, que é a interface gráfica para gerar e especificar os arquivos de configuração, arquivos de dados, e as ferramentas a serem usadas.

Passo 2.
O **Tool Manager** abre uma conexão com o **DataMover**, que roda no servidor.

Passo 3.
Utilizar o **Tool Manager** para especificar:

- a base de dados e tabelas, ou os arquivos contendo os dados tanto no cliente como no servidor;
- qual ferramenta de mineração será utilizada;
- qual ferramenta de visualização será utilizada;
- quanto dos dados devem ser mostrados;

- um arquivo opcional no cliente ou no servidor no qual será salvo os resultados para processamento futuros.

A informação recuperada pelo **DataMover** é utilizada para guiar esta interação. Como resultado, o **Tool Manager** gera um arquivo de configuração. Este arquivo contém os parâmetros que determinam a execução dos passos seguintes.

Passo 4.
O **Tool Manager** transmite uma cópia do arquivo de configuração gerado no passo 3 para o **DataMover**, que processa o arquivo:

- comunicando com um banco de dados ou um arquivo;
- realizando as transformações dos dados especificadas;
- rodando as ferramentas de mineração;
- gerando o arquivo de dados.

Este arquivo de dados consiste dos dados do usuário em um formato específico, que pode ser lido pela ferramenta **MineSet™**. Então uma cópia do arquivo de dados é colocada na estação de trabalho (cliente) do usuário.

Passo 5.
O **Tool Manager** invoca a ferramenta de visualização especificada no passo 3.

Passo 6.
A ferramenta acessa o arquivo de dados e, baseado nos parâmetros definidos pelo usuário no passo 3, mostra graficamente os dados.

Passo 7.
Se for gerado um classificador, tal classificador pode ser aplicado a dados adicionais.

3. Visualização de Dados Sem a Utilização do Tool Manager

A partir daqui será mostrado como realizar visualizações de dados em algumas das ferramentas integrantes do **MineSet™** sem utilizar os recursos oferecidos pelo **Tool Manager**. Os procedimentos que devem ser seguidos e os arquivos necessários para a realização das visualizações serão vistos em detalhes.

Ocorre que muitas vezes o usuário já possui os dados necessários para realizar a visualização, e não precisa passar por nenhuma etapa do **Tool Manager**, ou mesmo que o usuário deseja realizar visualizações de pouca quantidade de dados para fazer, por exemplo, demonstrações. Por isso, o objetivo desta parte do relatório é o de incentivar o usuário a começar a utilizar o **MineSet™** e desta forma, fazer com que o usuário sinta como pode ser útil visualizar dados de uma maneira mais clara. E ainda, dar uma melhor noção do funcionamento das

ferramentas de visualizações e mostrar como os arquivos necessários devem ser criados.

Os procedimentos e arquivos necessários para fazer visualizações nas ferramentas **Tree Visualizer**, **Scatter Visualizer** e **Map Visualizer** serão apresentados nas seções seguintes.

4. Guia de Utilização do Scatter Visualizer

Este guia de utilização, mostra como realizar visualizações de dados utilizando o **Scatter Visualizer** sem utilizar o **Tool Manager**.

A ferramenta de visualização, **Scatter Visualizer**, mostra em uma perspectiva tridimensional os dados organizados no **Arquivo de Dados** e que serão interpretados pelo **Arquivo de Configuração**. Os dados serão mostrados em um espaço delimitado por no máximo três eixos de coordenadas. A forma como devem ser criados estes arquivos será detalhada mais adiante. Em seguida, uma visão geral dos tipos de dados e expressões permitidas no aplicativo **Scatter Visualizer** será apresentada para situar melhor o usuário com as características da ferramenta.

Todos os exemplos mostrados para ilustrar a utilização dos arquivos de dados e de configuração do **Scatter Visualizer** são hipotéticos e não foram extraídos de nenhum banco de dados real.

4.1 Visão Geral

4.1.1 Tipos de Dados

O **Scatter Visualizer** suporta os seguintes tipos de dados: int, float, double, dataString, string e date. A ferramenta também aceita arrays de tamanho fixo.

4.1.2 Expressões

As expressões seguem o padrão da sintaxe da linguagem C. As seguintes operações podem ser feitas:

+ - * / % == != > < >= <= && || ! & | ^ ?:

As seguintes funções também são aceitas:

divide(x,y,z) é equivalente ao comando em C y == 0 ? z : x/y;

modulus(x,y,z) é igual a **divide**, porém para módulo.

Para maiores detalhes veja o *Apendix D: Creating Data and Configuration Files for the Scatter Visualizer* em [1].

4.2 Arquivo de Dados

O arquivo de dados é um arquivo no formato ASCII organizado em linhas e colunas, sendo que cada linha representa um registro dos dados e cada coluna representa um campo deste registro. A separação dos campos (colunas) deve ser feita com um simples *tab,* porém outras formas de separação podem ser escolhidas. Todas as linhas do arquivo devem conter os mesmos campos. Um exemplo simples de um arquivo de dados é mostrado abaixo:

FSP	2200	5000	1950	1000	1300
ESP	2080	4500	1720	1200	1100
JT	400	1290	160	100	73
NP	200	250	42	32	45
DP	120	320	67	100	31
JM	50	150	40	20	40
JB	20	50	10	21	34
CSP	60	70	40	40	10
GM	400	520	450	150	200

Neste exemplo, as colunas estão separadas por *tab*, e representam os campos (do arquivo de configuração) *jornal*, *Sul*, *Centro*, *Norte*, *Oeste* e *Leste*. Onde *jornal* representa o nome de um jornal qualquer e, *Sul*, *Centro*, *Norte*, *Oeste* e *Leste* representam as vendas deste jornal nas respectivas regiões da cidade de *São Carlos* (os dados do exemplo são hipotéticos).

O arquivo de dados não deve possuir linhas em branco ou comentários. Dados esquecidos ou extras numa linha causam erros durante a interpretação. Se existir dois ou mais *tabs* (ou qualquer outro separador) antes de um campo, os campos anteriores serão considerados como nulos ou seja desconhecidos, mas serão interpretados. O arquivo de dados do **Scatter Visualizer** possui extensão *.data*.

4.3 Arquivo de Configuração

O arquivo de configuração do **Scatter Visualizer** consiste de uma série de seções: **Input**, **Expressions** e **View**. O arquivo de configuração possui a extensão *.scatterviz* . Os arquivos padrões e as seções do arquivo de configuração serão detalhadas a seguir.

4.3.1 Arquivos Padrões

Assim que cada seção é encontrada, um arquivo de configuração especial é lido. Este arquivo é chamado de arquivo padrão. Este arquivo é procurado da seguinte forma:

1. diretório */usr/lib/MineSet/scatterviz*. Neste diretório estão os padrões do sistema.
2. diretório *~/.MineSet* (o usuário deve criar este diretório) que está na área do usuário. Neste diretório estão os padrões definidos pelo usuário.
3. diretório atual de trabalho.

Pode existir um arquivo de configuração em cada um dos diretórios citados acima, mesmo com os nomes iguais, porém, assim que vão sendo encontrados, as definições de um arquivo sobrepõem às definições lidas anteriormente. E por fim, as opções dos arquivos de configuração sobrepõem a dos arquivos padrões.

4.3.2 Seção Input

A seção **Input** contém o nome e o formato dos arquivos de dados. É nesta seção que é atribuído o significado de cada coluna do arquivo de dados. Um exemplo seria o seguinte:

```
input                   # palavra chave input que representa o início da seção
{
        file "jornais.data";
        string Jornal;
        float Sul;
        float Centro;
        float Norte;
        float Oeste;
        float Leste;
}
```

Este exemplo define que os dados serão obtidos do arquivo *jornais.data*, e que existem seis campos que são *Jornal*, *Sul*, *Centro*, *Norte*, *Oeste* e *Leste*. O *Jornal* é uma string que representa o nome do jornal, enquanto que os demais campos são do tipo float e representam as vendas do jornal nas respectivas regiões da cidade.

A sintaxe da sentença **file** é da forma **file "***nomearquivo.data***"**. Os dados são procurados primeiro (caso não seja dado o caminho absoluto) no diretório onde se encontra o arquivo de configuração, caso não seja encontrado então o diretório atual é procurado.

Na seção **Input** pode-se declarar sentenças de enumerações, **enum** que servem como os índices dos campos de arrays. A sentença **enum** tem as seguintes formas:

enum tipo nome **from** valor1 **to** valor2 **by** incremento;
enum tipo nome **from** valor1 **to** valor2 **across** incremento;
enum tipo nome { valor1, valor2, ..., valorN};

Por exemplo, as seguintes sentenças são equivalentes

enum int ano from 1900 to 1990 by 10;
enum int ano from 1900 to 1990 across 10;
enum int ano { 1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990 };

e representam um tipo **enum** ano que contém 10 números que vai de 1900 à 1990.

A sentença **enum** possui um suporte especial para o tipo de dado **date**. Para maiores informações veja o *Apendix D: Creating Data and Configuration Files for the Scatter Visualizer* em [1].

Arrays podem ser declarados das seguintes formas:
tipo nome [numero];
tipo nome [nome_enum];
tipo nome [null nome_enum];

Alguns exemplos:
double vendas [10];
ou,
enum int ano from 1900 to 1990 by 10;
double vendas [ano];

Os separadores dos números de um array podem sobrepor o separador padrão *tab*, pela opção **separator**. Por exemplo,
double vendas [10] separator ':';
onde **separator** muda o separador dos itens do array para ":" ao invés do *tab* (padrão).

4.3.3 Seção Expressions

Nesta seção pode-se definir campos adicionais que são expressões de campos existentes. Um exemplo seria criar um campo como a soma de outros dois campos existentes.

Dando continuidade ao exemplo dado na seção **Input**, segue uma declaração de seção **Expressions**:

```
expressions
{
        float VendasTotais = Centro+Sul+Norte+Leste+Oeste;
}
```

Neste exemplo, um novo campo do tipo float, *VendasTotais*, é definido como sendo a soma das vendas das regiões *Centro*, *Sul*, *Norte*, *Leste* e *Oeste* da cidade.

A seção **Expressions** não contém opções, logo nenhum arquivo de configuração é lido enquanto esta seção está sendo interpretada.

4.3.4 Seção View

Esta seção descreve como os dados são mostrados, incluindo o mapeamento dos tamanhos, cores, eixos e outros. Os valores padrões para estas opções estão no arquivo */usr/lib/MineSet/scatterviz/view.scatterviz.options*. Um exemplo segue abaixo:

```
view
{
        entity Jornal;
        size VendasTotais;
        color Jornal, legend label "Jornais Vendidos em Sao Carlos";
        axis Centro , label "Regiao Central", color "red";
        axis Sul, label "Regiao Sul", color "green";
        axis Norte, label "Regiao Norte", color "steelblue";
        options grid color "#202000";
}
```

A sentença **entity** permite especificar uma variável que identifica uma entidade unicamente no display. Esta sentença consiste de uma série de cláusulas, que são separadas por vírgulas:

```
        entity cláusula1, cláusula2, ...
```

A sentença **size** descreve como um campo de dados é mapeado para o tamanho das entidades. Seu formato é

```
        size cláusula1, cláusula2, ...
```

A sentença **color** descreve como os valores são mapeados às cores. O formato é parecido com a sentença descrita anteriormente.

A sentença **axis** faz com que uma variável seja utilizada como um eixo em 3D. O valor das variáveis determinam onde as entidades são posicionadas nos eixos. Seu formato possui várias cláusulas separadas por vírgulas.

Para maiores informações sobre a seção **View** veja o *Apendix D: Creating Data and Configuration Files for the Scatter Visualizer* em [1].

4.4 Exemplos dos Arquivos de Dados e Configuração do Scatter Visualizer

Exemplos dos arquivos de dados e configuração do **Scatter Visualizer** podem ser analisados e rodados no diretório */usr/lib/MineSet/examples/scatterviz_examples* - este diretório é criado na instalação do **MineSet™**. Para rodar os exemplos, o usuário deve entrar com o comando **scatterviz** (arquivo executável gerado na instalação) diretamente do prompt da shell do sistema operacional. O exemplo a seguir mostra como é iniciado o **Scatter Visualizer,** considerando que o diretório atual seja o citado acima e o arquivo *company.scatterviz* resida neste diretório:

```
% scatterviz company.scatterviz
```

O usuário pode rodar a aplicação através do **Tool Manager**. Um exemplo da tela da ferramenta **Scatter Visualizer** é mostrado na figura 2.

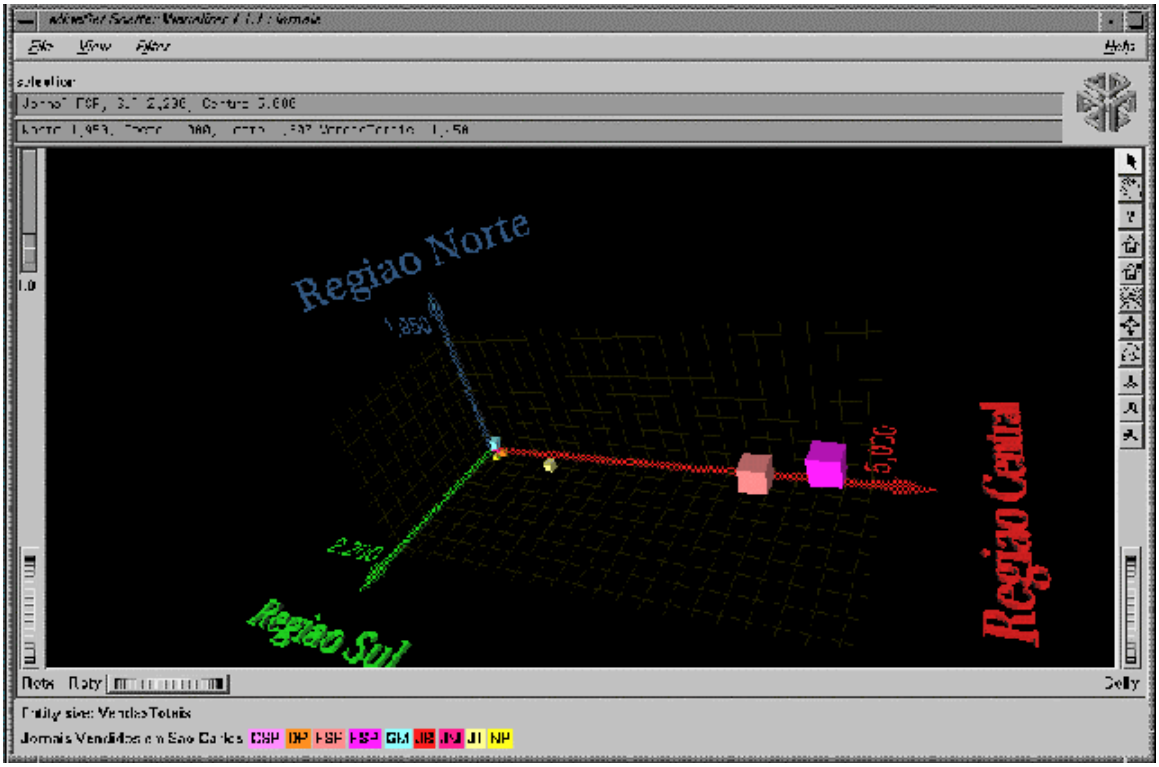

Figura 2. Visualização na ferramenta **Scatter Visualizer**

5. Guia de Utilização do Tree Visualizer

Este guia mostra como realizar visualizações de dados utilizando o **Tree Visualizer** sem utilizar o **Tool Manager**.

O **Tree Visualizer** possibilita que o usuário visualize os dados em uma forma hierárquica, ou melhor, em uma abstração em árvore com raiz, nós filhos e nós folhas. A visualização é feita criando-se os arquivos de Dados e de Configuração. A hierarquia dos dados contidos no arquivo de Dados é definida através do arquivo de Configuração. Os detalhes da criação e implementação destes dois arquivos serão visto no decorrer deste guia. Esta forma de abstração é muito útil para visualizar os dados que possuem natureza hierárquica. Por exemplo, os dados das vendas de uma rede de lojas em uma região, com lojas matrizes e filiais espalhadas nessa região.

Todos os exemplos mostrados para ilustrar a utilização dos Arquivos de Dados e de configuração do **Tree Visualizer** são hipotéticos e não foram extraídos de nenhum banco de dados real.

5.1 Visão Geral

5.1.1 Tipos de Dados

O **Tree Visualizer** suporta os seguintes tipos de dados: : int, float, double, dataString, string e date.

A ferramenta também aceita os tipos de dados array de tamanhos fixos e variáveis e array enumerado.

5.1.2 Expressões

As expressões seguem o padrão da sintaxe da linguagem C. As seguintes operações podem ser feitas:

+ - * / % == != > < >= <= && || ! & | ^ ?:

As seguintes funções também são aceitas:

divide(x,y,z) é equivalente ao comando em C, y == 0 ? z : x/y;
modulus(x,y,z) é similar a **divide**, porém para módulo;
hierarchy(string) é valida apenas dentro da seção hierarquia. Esta função cria uma string que representa os componentes de uma hierarquia;
isSummary() verifica se expressão está sendo aplicada à informação da base.

Para maiores detalhes o leitor deve ler o *Apendix B: Creating Data and Configuration Files for the Tree Visualizer* em [1].

5.2 Arquivo de Dados

O arquivo de dados é um arquivo no formato ASCII organizado em linhas e colunas, sendo que cada linha representa um registro dos dados e cada coluna representa um campo deste registro. A separação dos campos (colunas) pode ser feita com um simples *tab* ou outras formas de separação podem ser escolhidas. Todas as linhas do arquivo devem conter os mesmos campos. Um exemplo simples de um arquivo de dados é mostrado abaixo:

03/09/97	Refrigerante	A	10,6,30,50,40,20	12,2,23,30,20,30
03/09/97	Refrigerante	B	12,10,40,60,30,10	8,9,30,40,20,9
03/09/97	Refrigerante	C	7,7,20,40,30,2	9,2,22,41,32,1
03/09/97	Cerveja	X	8,20,32,42,30,12	11,2,10,22,40,1
03/09/97	Cerveja	Y	4,2,26,60,20,10	5,12,22,40,22,7
03/09/97	Cerveja	Z	12,10,30,20,50,10	9,2,22,10,30,2

Neste exemplo as colunas estão separadas por de *tab*, e representam os campos (do arquivo de configuração) *data*, *bebida*, *marca*, *vendasM* e *vendasF*. Onde o campo *data* representa a data em que foi efetuada venda, *bebida* representa o tipo de bebida vendida, *marca* representa o nome da bebida e, os campos *vendasM* e *vendasF* representam a quantidade de bebida consumida por homens e mulheres, respectivamente, segundo uma faixa etária. Estes dois últimos campos são separados por vírgulas, e cada posição nestes campos é representada (no arquivo de configuração) pelas seguintes faixas etárias: “18-26”, “26-34”, “34-42”, ”42-50”, ”50-58” e “58+”.

O arquivo de dados não deve possuir linhas em branco ou comentários. Dados esquecidos ou extras numa linha causam erros durante a interpretação. Se existirem dois ou mais *tabs* (ou qualquer outro separador) antes de um campo, os campos anteriores serão considerados como nulos ou seja desconhecidos, mas serão interpretados.

O arquivo de dados do **Tree Visualizer** possui extensão *.data*.

5.3 Arquivo de Configuração

O arquivo de configuração do **Tree Visualizer** consiste de quatro seções e duas subseções, as seções são **Input**, **Expressions**, **Hierarchy** e **View**, e as subseções são **Aggregate** e **Expressions**, ambas contidas na seção **Hierarchy**.

O arquivo de configuração possui a extensão *.treeviz* . Os arquivos padrões, as seções e subseções do arquivo de configuração serão detalhadas a seguir.

5.3.1 Arquivos de Padrões

Assim que cada seção é encontrada, um arquivo de configuração especial é lido. Este arquivo é chamado de arquivo padrão. Este arquivo é procurado da seguinte forma:

1. O diretório */usr/lib/MineSet/treeviz*. Neste diretório estão os padrões do sistema.
2. O diretório *~/.MineSet* (o usuário deve criar este diretório) que está na área do usuário. Neste diretório estão os padrões definidos pelo usuário.
3. O diretório atual de trabalho.

Pode existir um arquivo de configuração em cada um dos diretórios citados acima, mesmo com os nomes iguais, porém, assim que vão sendo encontrados, as definições de um arquivo sobrepõem às definições lidas anteriormente. E por fim, as opções dos arquivos de configuração sobrepõem a dos arquivos padrões.

5.3.2 Seção Input

A seção **Input** contém o nome e o formato dos arquivos de dados. É nesta seção que é atribuído o significado de cada coluna do arquivo de dados. Um exemplo seria o seguinte:

```
input          # palavra chave input que representa o início da seção
{
        file "bebidas.data";
        enum string faixa_etaria{
                "18-26", "26-34", "34-42",  "42-50",  "50-58",  "58+"
        };
        string data;
        string bebida;
        string marca;
        float vendasM[ enum faixa_etaria ];
        float vendasF[ enum faixa_etaria ];
}
```

Como é mostrado, os dados são obtidos do arquivo *bebidas.data* e existem cinco campos que são *data*, *bebida*, *marca*, *vendasM* e *vendasF*. Onde *data* representa a data da venda, *bebida* representa o tipo de bebida vendida, *marca* representa o nome da bebida, *vendasM* e *vendasF* representam a quantidade (da bebida) vendida para pessoas do sexo masculino e feminino, respectivamente. A

quantidade vendida é discriminada pela idade e pelo sexo da pessoa através de um array enumerado, cujo os índices são representados pelo tipo enumerado, *faixa_etaria*.

A sintaxe das sentenças **file** e **separator** são idênticas a do visualizador mostrado anteriormente. (Veja Guia de Utilização do Scatter Visualizer - Criação do Arquivo de Dados e de Configuração)

Na seção **Input**, pode-se definir tipos de dados **enum** e arrays enumerados, existindo ainda, várias opções que podem ser ajustadas. Para maiores detalhes o leitor deve ler o *Apendix B: Creating Data and Configuration Files for the Tree Visualizer* em [1].

5.3.3 Seção Expressions

Esta seção permite que o usuário defina novas colunas, que são expressões resultantes de colunas existentes. Por exemplo:

```
expressions
{
        float VendasTotais =  vendasM + vendasF;
        float pct = divide (vendasM*100, VendasTotais, 50.0);
}
```

Neste exemplo, o campo *VendasTotais* é criado a partir da soma entre *vendasM* e *vendasF*, que representa a quantidade total de bebida consumida por homens e mulheres. E o campo pctMasc representa a porcentagem de bebida consumida pelos homens em relação à *VendasTotais*.

A seção **Expressions** não possui opções.

5.3.4 Seção Hierarchy

Esta seção descreve como os dados do arquivo de dados são convertidos em uma hierarquia. Por exemplo:

```
hierarchy
{
        levels bebidas, marca;
        key vendasM;

        aggregate
        {
                sum vendasM;
```

```
        sum vendasF;
    }

    expressions
    {
        float VendasTotais = vendasM + vendasF;
        float pct = divide (vendasM*100, VendasTotais, 50.0);
    }
}
```

A sentença **levels** define a hierarquia dos dados. A sua sintaxe é

```
levels campo1, campo2, ...
```

os campo1, campo2, ... representam as colunas dos dados. Onde campo1 está em uma hierarquia superior a do campo2 e assim por diante.

A sentença **key** especifica a chave que será utilizada para selecionar as barras de cada nó na hierarquia. Apenas uma única chave pode ser utilizada, e esta chave não pode ter sido usada na sentença **levels**. A sintaxe desta sentença é a seguinte:

```
key nome [sort [ascending | descending ] ];
```

Onde *nome* representa uma coluna de dados (campo) e **sort**, **ascending**, **descending** são opções para ordenação das barras.

5.3.4.1 Subseção Aggregate

Esta subseção descreve como os dados são agregados nos níveis superiores da hierarquia. No exemplo mostrado anteriormente temos:

```
aggregate
{
    any data;
    sum vendasM;
    sum vendasF;
}
```

que indica que as **vendasM** e **vendasF** serão somadas na medida que sobem na hierarquia, ou seja, cada nível faz a soma dos valores dos níveis inferiores. Existem, ainda, além da agregação **sum**, as agregações **average**, **min**, **max**, **count** e **any**.

5.3.4.2 Subseção Expressions

Esta subseção é parecida com a seção **Expressions**, exceto que a subseção é aplicada somente após a criação e agregação da hierarquia. No exemplo dado anteriormente, tem-se:

```
expressions
{
        float VendasTotais = vendasM + vendasF;
        float pct = divide (vendasM*100, VendasTotais, 50.0);
}
```

Se a seção **Expressions** for declarada, então a subseção **Expressions** não poderá ser mais declarada, e vice-versa.

Para maiores detalhes e explicações, o leitor deve ler o *Apendix B: Creating Data and Configuration Files for the Tree Visualizer* em [1].

5.3.5 Seção View

A seção **View** descreve como a hierarquia que foi definida será visualizada, incluindo o mapeamento de alturas, cores, rótulos e outros. Um exemplo segue abaixo:

```
view hierarchy landscape
{
        height vendasM, normalize levels, max 4.0, filter > 5%;
        base height max 2.0;
        disk height vendasM;
        color pct, scale 0 25 50 75 100;
        color colors "red" "Indigo" "turquoise1" "blue" "azure";
        options rows 1;
        message "Data: %s , Quantidade de bebida vendida:%d , %.2f%% (%s)",
        data, VendasTotais, pct>50?pct:100-pct, pct>50?"masculino":"feminino";
        color legend label "Porcentagem Consumida (sexo masculino) " "  0%"
        " 25%" " 50%" " 75%" "100%";
        height legend label "altura: numero de bebidas consumidas (sexo
masculino)";
        disk height legend label "disco: quantidade consumida (sexo masculino)";
}
```

view hierarchy landscape indica o tipo de visualização suportado - este é o único tipo suportado. Assim que o interpretador entra na seção **View**, o arquivo padrão *viewHierarchyLandscape.treeviz.options* é lido.

A sentença **height** descreve como as colunas são mapeadas nas alturas dos objetos visuais. Esta sentença possui várias cláusulas:

- **normalize**, que determina o maior valor para a altura de um objeto e normaliza os outros em função desta altura;
- **max**, é utilizada com a cláusula **normalize** para definir a maior altura permitida;
- **scale**, se as cláusulas **normalize** e **max** não forem usadas, a cláusula **scale** pode escalonar os valores das alturas dos objetos;
- **filter**, que filtra os objetos visuais baseado na altura dos mesmos;
- **legend**, que define significados para os mapeamentos feitos.

A sentença **base height** especifica como a altura da base é calculada.

A sentença **color** descreve como os valores são mapeados para as cores. Esta sentença possui as seguintes cláusulas:

- **color naming**, que segue as convenções do **X Window System** para nomes de cores;
- variável **color**, que é especificada para ser mapeada em uma cor;
- **key**, ao invés da variável anterior pode-se utilizar uma chave;
- **colors**, que especifica as cores que serão usadas;
- **scale**, permite associação de valores a uma variedade contínua de cores;
- **legend**, que cria uma legenda para as cores.

Para maiores detalhes sobre as sentenças, cláusulas e as muitas opções existentes nesta seção o leitor deve ler o *Apendix B: Creating Data and Configuration Files for the Tree Visualizer* em [1].

5.4 Exemplos dos Arquivos de Dados e Configuração do Tree Visualizer

Exemplos dos arquivos de dados e configuração do **Tree Visualizer** podem ser analisados e rodados no diretório */usr/lib/MineSet/examples/treeviz_examples* - este diretório é criado na instalação do **MineSet™**. Para rodar os exemplos, o usuário deve entrar com o comando **treeviz** (arquivo executável gerado na instalação do **MineSet™**) diretamente no prompt da shell do sistema operacional. O exemplo a seguir mostra como é iniciado o **Tree Visualizer,** considerando que o diretório atual seja o citado acima e o arquivo *brand.treeviz* resida neste diretório:

```
% mapviz brand.treeviz
```

O usuário pode rodar a aplicação através do **Tool Manager**.

Um exemplo da tela da ferramenta **Tree Visualizer** é mostrado na figura 3.

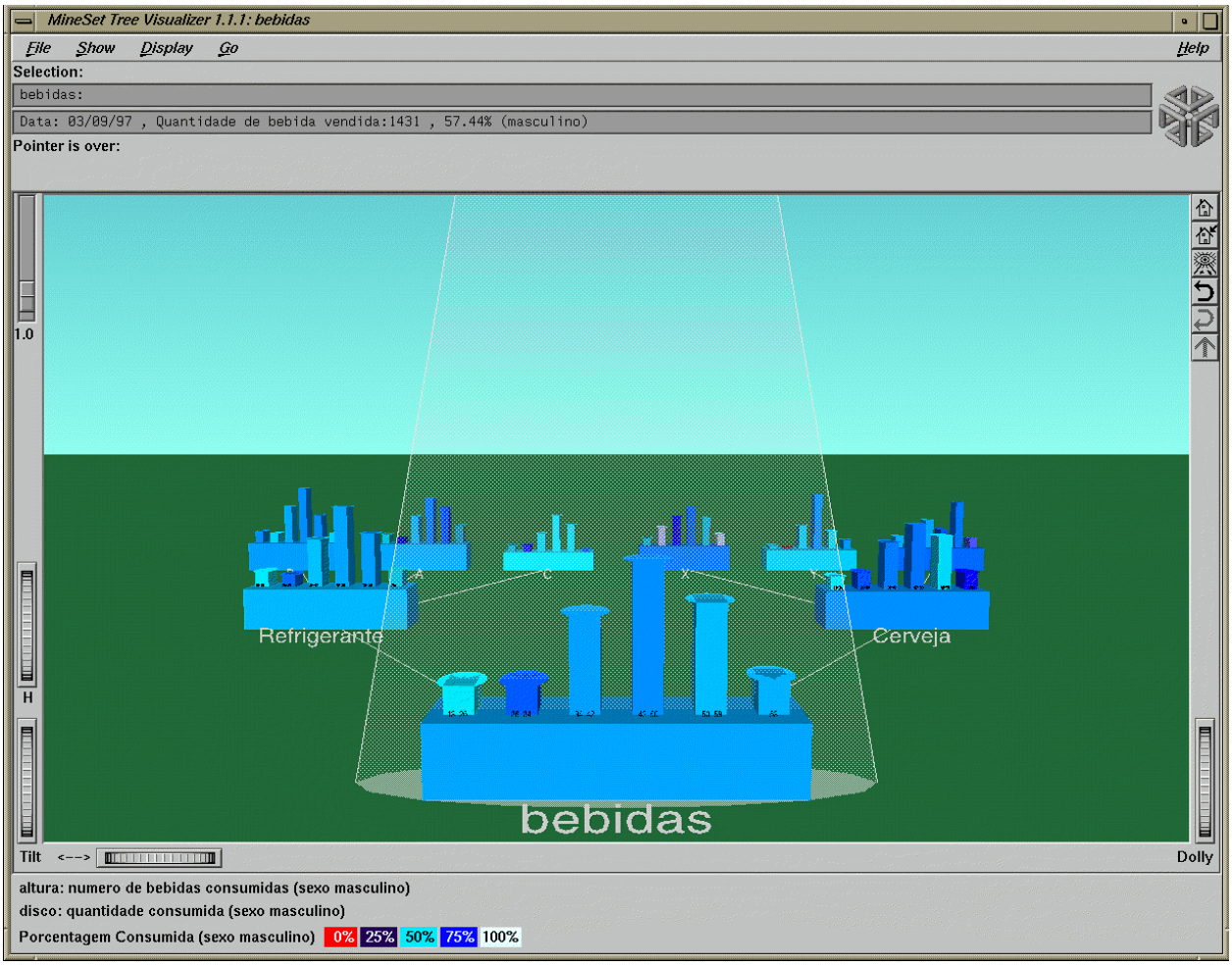

Figura 3. Visualização na ferramenta **Tree Visualizer**

6. Guia de Utilização do Map Visualizer

Este guia de utilização, mostra como realizar visualizações de dados utilizando o **Map Visualizer** sem utilizar o **Tool Manager**.

O aplicativo **Map Visualizer** permite que o usuário visualize dados que estão relacionados em um espaço geográfico como país, estado, cidade, terreno e outros. Esta ferramenta de visualização mostra numa perspectiva tridimensional os dados organizados no **Arquivo de Dados** e definidos no **Arquivo de Configuração**, os dados são mostrados como objetos em um formato geométrico definido no **Arquivo GFX**, tais objetos são mostrados em uma hierarquia definida no **Arquivo de Hierarquia**. A seguir será visto como estão organizados os arquivos de dados e de configuração.

Todos os exemplos mostrados para ilustrar a utilização dos arquivos de dados e de configuração do **Map Visualizer** são hipotéticos e não foram extraídos de nenhum banco de dados real.

6.1 Visão Geral

6.1.1 Tipos de Dados

O **Map Visualizer** suporta os seguintes tipos de dados: : int, float, double, dataString, string e date. A ferramenta também aceita arrays de tamanho fixo.

6.1.2 Expressões

As expressões seguem o padrão da sintaxe da linguagem C. As seguintes operações podem ser feitas:

+ - * / % == != > < >= <= && || ! & | ^ ?:

As seguintes funções também são aceitas:

> **divide**(x,y,z) é equivalente ao comando em C y == 0 ? z : x/y;
> **modulus**(x,y,z) é igual a **divide**, porém para módulo.

Para maiores detalhes veja o *Apendix B: Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer*, pg. 380 do manual do **MineSet™**.

6.2 Arquivo de Dados

O arquivo de dados é um arquivo no formato ASCII organizado em linhas e colunas, sendo que cada linha representa um registro dos dados e cada coluna representa um campo deste registro. A separação dos campos (colunas) deve ser feita com um simples *tab,* mas outras formas de separação podem ser escolhidas. Todas as linhas do arquivo devem conter os mesmos campos. Um exemplo simples de um arquivo de dados é mostrado abaixo:

```
RS    0,10,32,50,80,90,98,99,101        15
SC    0,20,10,40,50,20,30,43,606
PR    0,40,30,50,89,87,99,102,110       17
SP    0,300,330,430,490,499,500,526,550    45
MG    0,120,110,140,180,198,220,240,280    25
RJ    0,110,98,101,120,160,130,140,170     20
```

```
ES    0,18,20,14,38,60,40,45,57 10
MS    0,13,10,14,15,18,29,30,31 7
MT    0,15,14,16,27,32,33,34,37 6
GO    0,11,12,18,21,27,29,30,28 9
BA    0,50,55,43,60,50,70,90,100      15
SE    0,15,10,10,14,16,11,10,19 8
```

Neste exemplo, existem três colunas que estão separadas através de tabs que representam os campos (do arquivo de configuração) *estado*, *vendas* e *lojas*. Na primeira coluna aparecem as siglas dos estados brasileiros. A segunda está divida e separada por vírgulas, e representam as vendas da loja hipotética em cada estado durante os anos de 1989 à 1997. E por fim, o último campo representa o número de lojas situadas no estado.

O arquivo de dados não deve possuir linhas em branco ou comentários. Dados esquecidos ou extras numa linha causam erros durante a interpretação. Se existir dois ou mais *tabs* (ou qualquer outro separador) antes de um campo, os campos anteriores serão considerados como nulos ou seja desconhecidos, mas serão interpretados. O arquivo de dados do **Map Visualizer** possui extensão *.data*.

6.3 O Arquivo de Configuração

O arquivo de configuração do **Map Visualizer** consiste de uma série de seções: **Input**, **Expressions** e **View**. O arquivo de configuração possui a extensão *.mapviz.* Os arquivos padrões e as seções do arquivo de configuração serão detalhadas a seguir.

6.3.1 Arquivos de Padrões

Assim que cada seção é encontrada, um arquivo de configuração especial é lido. Este arquivo é chamado de arquivo padrão. Este arquivo é procurado da seguinte forma:

1. O diretório */usr/lib/MineSet/mapviz*. Neste diretório estão os padrões do sistema.
2. O diretório *~/.MineSet* (o usuário deve criar este diretório) que está na área do usuário. Neste diretório estão os padrões definidos pelo usuário.
3. O diretório atual de trabalho.

Pode existir um arquivo de configuração em cada um dos diretórios citados acima, mesmo com os nomes iguais, porém, assim que vão sendo encontrados, as definições de um arquivo sobrepõem às definições lidas anteriormente. E por fim, as opções dos arquivos de configuração sobrepõem a dos arquivos padrões.

6.3.2 Seção Input

A seção **Input** contém o nome e o formato dos arquivos de dados. É nesta seção que é atribuído o significado de cada coluna do arquivo de dados. Um exemplo seria o seguinte:

```
input
{
        file "vendas.br.data";
        enum int Ano from 1989 to 1997 by 1;
        string estado;
        double vendas[enum Ano] separator ',';
        float lojas;
}
```

Este exemplo define que os dados serão obtidos do arquivo *vendas.br.data*, e que existem três campos que são *estados*, *vendas* e *lojas*. O *estado* é uma string que representa o nome de um estado brasileiro. O campo *vendas* é um array de **double** que é indexado pelo tipo enumerado *Ano*, sendo que cada item do array é separado por uma vírgula. Este array representa as vendas da *loja XX* nos anos de 1989 à 1997. E por fim, o campo *lojas* que representa o número de lojas que estão situadas no estado.

A sintaxe das sentenças **file**, **enum,** e **separator** são idênticas a do aplicativo mostrado anteriormente. (Veja Guia de Utilização do Scatter Visualizer)

6.3.3 Seção Expressions

Nesta seção pode-se definir campos adicionais que são expressões de campos existentes. Um exemplo seria criar um campo como a soma de outros dois campos existentes.

A seção **Expressions** do **Map Visualizer** é idêntica à seção já detalhada do **Scatter Visualizer**. (Veja Guia de Utilização do Scatter Visualizer)

6.3.4 Seção View

Esta seção descreve como os dados são mostrados, incluindo o mapeamento dos tamanhos, cores, eixos e outros no display. Os valores padrões para estas opções estão no arquivo */usr/lib/MineSet/mapviz/viewMap.mapviz.options*. Um exemplo da seção **View** é mostrado a seguir:

```
view map
{
   map objects "br.estados.hierarchy";
   slider enum Ano;
   height vendas;
   height legend label "Altura: Vendas da Loja XX no territorio brasileiro (1990-
1997)";
   color vendas, scale 0 200 400 600 800 1000;
   color colors "gray" "blue" "cyan" "green" "yellow" "red";
   color legend label "Color: Vendas" "0" "200" "400" "600" "800" "1000";
   message "Vendas %,.0f  , %.0f lojas", vendas, lojas;
   execute "xconfirm -t 'Vendas: %,.0f' -t 'N. lojas: %.0f ' > /dev/null",    vendas,
lojas;
   title "Vendas da Loja XX no territorio brasileiro (1990-1997)";
}
```

A sentença **map** especifica como os objetos gráficos são desenhados na janela principal. O formato típico da sentença é

map objects “nomearquivo_hierarquia”;

A sentença **slider** identifica uma chave para ser utilizada como uma dimensão de slider. A chave pode ser utilizada para realizar animações. A sintaxe é a seguinte,

slider [enum] nome_enum;

A sentença **height** descreve como as colunas de dados (campos) são mapeados à altura dos objetos gráficos. Esta sentença é formada por uma série de cláusulas.

A sentença **color** descreve como os valores são mapeados às cores. O formato é parecido com a sentença descrita anteriormente.

A sentença **message** especifica a mensagem que será mostrada quando um objeto é selecionado. A sintaxe é muito parecida com a sentença **printf** da linguagem C.

A sentença **execute** permite que se possa executar um comando da shell do sistema operacional através de um “double-click” com o mouse num objeto da tela. No exemplo acima xconfirm mostra uma janela com informações sobre o item.

Para maiores detalhes sobre a seção **View** veja o *Apendix C: Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer* em [1].

6.4 Arquivo de Hierarquia

O arquivo de hierarquia define a hierarquia dos objetos, permitindo que objetos sejam mostrados em diferente níveis de agregação. O arquivo de hierarquia é referenciado na seção **View** do arquivo de configuração. Um exemplo de parte de um arquivo de hierarquia é apresentado abaixo,

```
estados          regioes          BR
br.estados.gfx   br.estados.gfx   br.estados.gfx
RS       BR_SUL      BRASIL
SC       BR_SUL      BRASIL
PR       BR_SUL      BRASIL
SP       BR_SUDESTE  BRASIL
MG       BR_SUDESTE  BRASIL
RJ       BR_SUDESTE  BRASIL
ES       BR_SUDESTE  BRASIL
```

Este arquivo mostra como os estados estão combinados em regiões (*BR_SUL* E *BR_SUDESTE*) e em um único objeto que engloba todos os estados, *BRASIL*.

A separação padrão é feita por intermédio de um tab.

Para maiores detalhes veja o *Apendix C: Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer* em [1].

6.5 Arquivo GFX

O arquivo **GFX** define a geometria de cada objeto usado pelo **Map Visualizer** dos objetos que serão mostrados. Cada arquivo possui muitos registros, um para cada objeto. Cada registro contém:

- o nome da palavra-chave do GFX
- o nome completo do GFX
- o número de vértices
- o formato
- os pares de vértices

Para se construir um arquivo **GFX**, deve-se seguir as seguintes etapas:

Etapa 1
Converter uma imagem geográfica (obtida através de scanner ou mesmo construídas) em um arquivo de formato RGB. Este arquivo servirá apenas como um modelo, ou um objeto em “background” usado para definir os objetos gráficos da etapa 6. Um exemplo de figura utilizada como modelo é mostrado na figura 4.

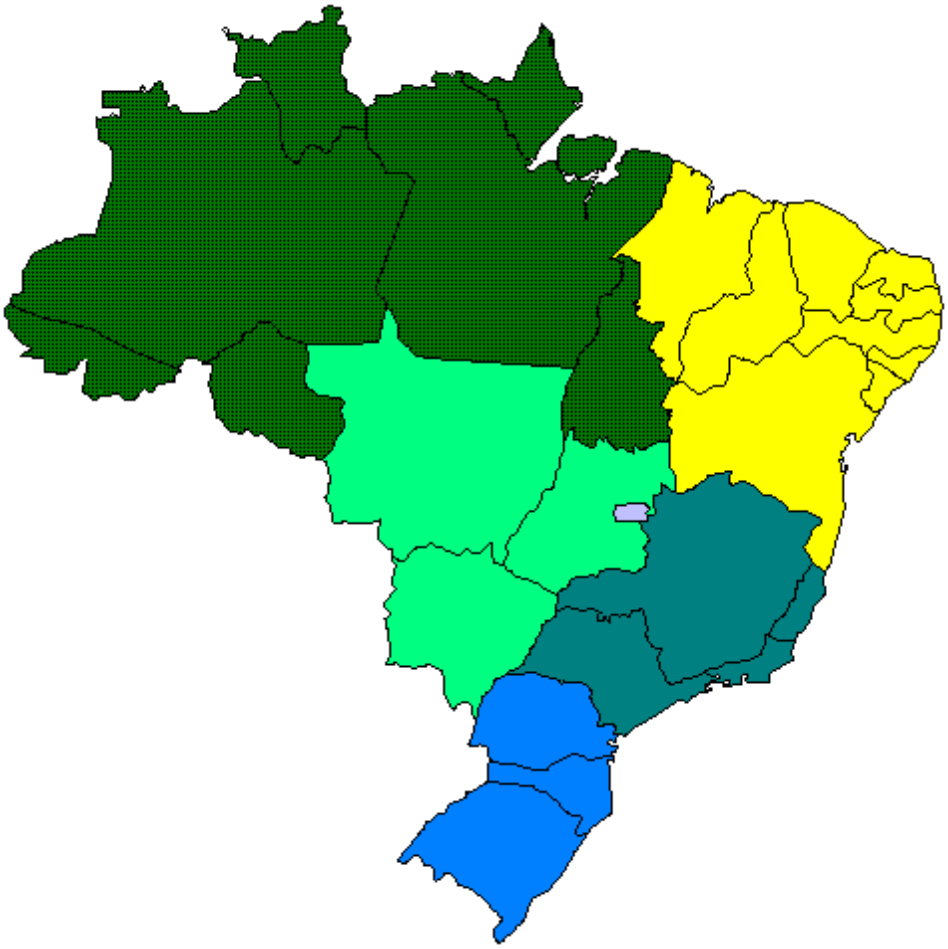

Figura 4. Exemplo de figura modelo que fica em “background” na janela do **i3dm**

Etapa 2
Rodar a aplicação em desenvolvimento num ambiente tridimensional, **i3dm**, residente no diretório */usr/demos/bin/* - **i3dm** é um aplicativo padrão da **Silicon Graphics** e normalmente é distribuído juntamente com o sistema operacional, **IRIX™**. Esta aplicação cria cinco janelas na sua tela: **Menu** na esquerda, **Input** embaixo, e quatro janelas na direita(**TOP**, **Pers**, **Front**, e **Right**).

Etapa 3
Mover o cursor para a janela **Front**.

Etapa 4
Pressionar o botão direito do mouse para ver as opções e selecionar **Load Image**.

Etapa 5
Entrar com o nome do arquivo RGB na janela **Input**. A imagem do arquivo RGB será mostrada na janela **Front**.

Etapa 6
Delinear a forma de cada objeto na imagem apontando e "clicando" nos pontos dos limites de cada objeto (neste caso, os estados). Faça isso no sentido horário para cada objeto. Cada ponto identificado é denominado vértice e é representado por valores numéricos dos eixos x e y. Siga o seguinte procedimento para delinear cada objeto gráfico:

- Posicionar o modelo da imagem ("background") no meio da janela **Front**, de tal forma, que os objetos desejados para delinear-se possam ser vistos, caso não for possível (imagem muito grande), então maximizar a janela **Front** e posicionar a figura.
- Ir para janela **Menu**, e selecionar **Create**.
- Escolher a opção **Line**.
- Começar o processo de delineação (apontar e selecionar) dos vértices com o botão esquerdo do mouse na janela **Front**.
- Uma linha vermelha irá acompanhar os movimentos sobre a imagem enquanto o usuário estiver selecionando cada vértice. E, assim que for completada a forma de um objeto (o primeiro e o último vértice são os mesmos) deve-se parar de delinear o objeto.
- Ir para janela **Menu,** e selecionar **Attrib**.
- Ir para a janela **Input** e entrar com um nome único para identificar o objeto que foi acabado de ser delineado. Isto representará a palavra-chave do objeto **GFX**. No exemplo que foi mostrado os nomes dados representam as siglas dos estados.
- Ir para janela **Menu**, e escolher o botão **Done**.

Etapa 7
Repetir a **Etapa 6** para cada objeto na mesma imagem. Para os vértices novos que coincidam com vértices já selecionados deve-se apontar e selecionar estes vértices com o botão direito para garantir que os vértices novos irão coincidir com os já selecionados.

Etapa 8
Quando todos objetos tiverem sido selecionados, salvar os vértices selecionados num arquivo:

- Ir para a janela **Menu** e selecionar o botão **File**.
- Entrar com o nome do arquivo na janela **Input** (extensão *.i3dm*)

Etapa 9
Sair da aplicação i3dm.

Etapa 10
Converter o arquivo no formato **i3dm** em um arquivo com formato **GFX** utilizando o aplicativo *convert.i3dm*, com a seguinte sintaxe:

/usr/lib/MineSet/mapviz/convert.i3dm nome*arquivo.i3dm* nome*arquivo*.gfx

Para cada objeto:

- confirmar o nome da palavra-chave do objeto (ex. SP)
- declarar o nome completo do objeto (ex. São Paulo)
- declarar se o objeto possui formas côncavas que necessitam de uma manipulação especial.

Para maiores detalhes veja *Appendix C: Creating Data, Configuration, Hierarchy, and GFX Files for the Map Visualizer* em [1].

6.6 Exemplos dos Arquivos de Dados, Configuração, Hierarquia e GFX do Map Visualizer

Exemplos dos arquivos de dados e configuração do **Map Visualizer** podem ser analisados e rodados no diretório */usr/lib/MineSet/examples/mapviz_examples* - este diretório é criado na instalação do **MineSet™**. Para rodar os exemplos, o usuário deve entrar com o comando **mapviz** (arquivo gerado na instalação do **MineSet™**) diretamente no prompt da shell do sistema operacional. O exemplo a seguir mostra como é iniciado o **Map Visualizer,** considerando que o diretório atual seja o citado acima e o arquivo *us.population.mapviz* resida neste diretório:

```
% mapviz us.population.mapviz
```

O usuário pode rodar a aplicação através do **Tool Manager**.

Um exemplo da tela da ferramenta **Map Visualizer** é mostrado na figura 5.

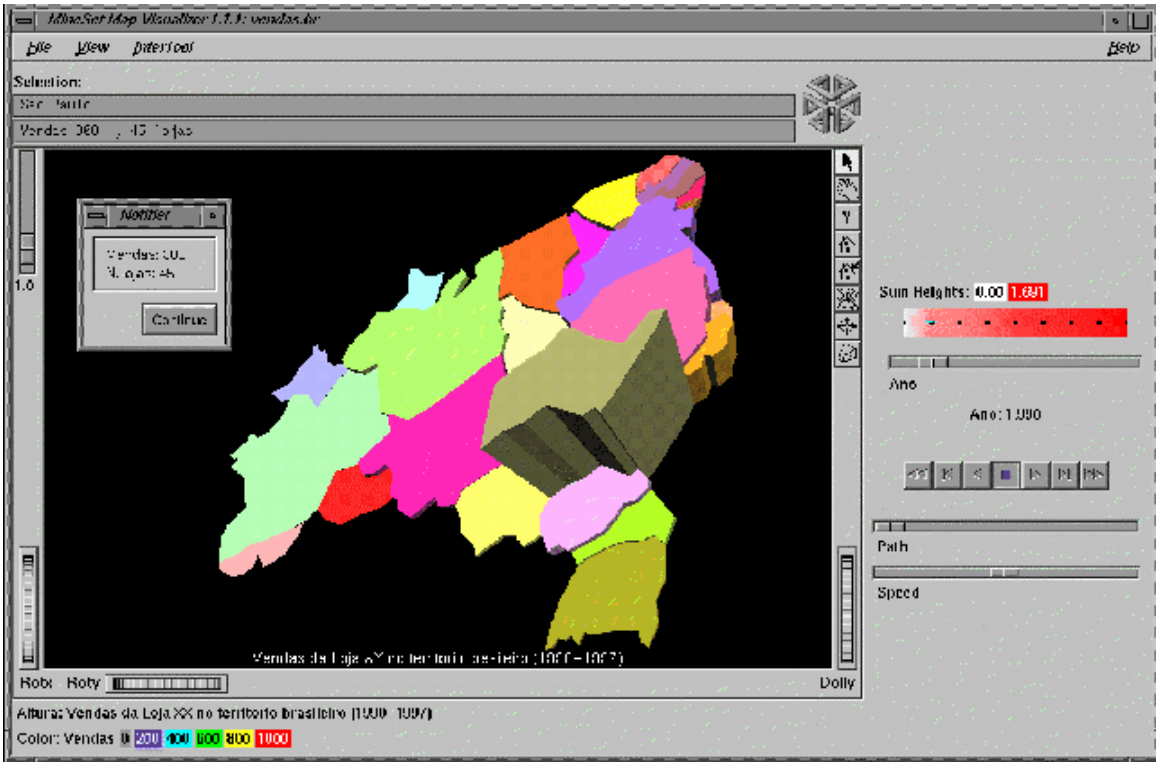

Figura 5. Visualização na ferramenta **Map Visualizer**

7. Guia de Utilização do Splat Visualizer

Este guia de utilização, mostra como realizar visualizações de dados utilizando o **Splat Visualizer** sem utilizar o **Tool Manager**.

A ferramenta de visualização, **Splat Visualizer**, permite ao usuário analisar visualmente relações existentes entre diversas variáveis tanto estaticamente como em animações. Este tipo de visualizador de dados é recomendado para base de dados com muitos registros. Os dados são mostrados em uma perspectiva tridimensional, tais dados estão organizados no **Arquivo de Dados** e que serão interpretados pelo **Arquivo de Configuração**. Os dados serão mostrados em um espaço delimitados por no máximo três eixos de coordenadas. Será visto com profundidade como tais arquivos devem ser criados. Em seguida, uma visão geral dos tipos de dados e expressões permitidas no aplicativo **Splat Visualizer** será apresentada para situar melhor o usuário com as características da ferramenta.

7.1 Visão Geral

O **Splat Visualizer** suporta os seguintes tipos de dados: int, float, double, dataString, string e date.

A ferramenta também aceita os tipos de dados array de tamanhos fixos.

7.2 O Arquivo de Dados

O arquivo de dados é um arquivo no formato ASCII organizado em linhas e colunas, sendo que cada linha representa um registro dos dados e cada coluna representa um campo deste registro. A separação dos campos (colunas) deve ser feita com um simples *tab* - outras formas de separação podem ser escolhidas. Todas as linhas do arquivo devem conter os mesmos campos. Um exemplo de como os dados estão dispostos num arquivo de dados é mostrado abaixo. Os dados deste exemplo foram retirados da base de dados do Programa de Melhoramento Genético da Raça Nelore, da Faculdade de Medicina de Ribeirão Preto.

```
AE0061:DF3919:1990:C 6740:CD4295:3.8:.62:5.1:.66:9.3:.67:15.0:.74:16.0:.75:1.85:9
0257:DA1787:1991:C 6740:BO0684:3.5:.60:5.3:.63:8.8:.64:12.4:.70:18.0:.72:1.77:6
5045:DM1928:1992:F 8200:DB3756:3.6:.54:4.0:.67:8.5:.67:10.0:.73:16.4:.73:1.60:1
8930:DU1549:1993:F 9902:CV1194:3.9:.60:4.1:.68:6.4:.69:11.0:.74:9.4:.73:1.35:34
0170:DX0985:1994:F 9902:CB8869:2.8:.58:4.7:.59:7.8:.58:10.4:.60:10.6:.59:1.32:39
8923:DU1533:1993:F 9902:BP1913:3.8:.60:5.9:.66:9.9:.64:14.1:.66:18.5:.72:1.92:34
AE0031:DF3912:1990:C 6740:CA2645:3.7:.63:4.8:.67:7.0:.69:10.7:.75:16.0:.75:1.59:9
F 2635:DN4731:1992:F 9902:BX0853:3.6:.55:5.7:.57:10.4:.58:12.3:.59:12.5:.56:1.61:6
AE0173:DF3946:1990:E 4568:CE4003:3.5:.55:0.9:.64:1.6:.66:5.0:.74:18.1:.75:1.24:9
1955:DN4834:1992:F 9902:CA6764:3.5:.59:5.5:.59:9.6:.60:13.3:.62:12.9:.61:1.64:24
2855:DN5341:1992:C 6740:AV8472:3.5:.58:4.4:.62:8.0:.64:9.9:.70:12.1:.72:1.41:3
AG0292:DO5699:1992:D 7680:CV1903:3.5:.53:2.1:.65:3.8:.66:8.3:.73:13.8:.67:1.29:9
931733:DS2266:1993:C 6740:CV4700:3.5:.63:8.0:.68:14.5:.68:19.7:.73:25.4:.74:2.54:8
6098:DS2750:1992:F 9902:CI4664:3.5:.63:5.3:.64:8.9:.64:12.9:.66:13.3:.72:1.62:16
8937:DU1554:1993:F 9902:AR3579:3.5:.58:5.9:.65:10.4:.65:13.8:.70:14.8:.71:1.76:34
7183:DX9723:1993:F 9902:DC7111:3.5:.61:4.7:.68:9.3:.69:11.3:.74:15.6:.74:1.65:16
```

Note que no exemplo mostrado a separação entre as colunas de cada linha é feita utilizando-se o caracter ":" (dois pontos).

O arquivo de dados não deve possuir linhas em branco ou comentários. Dados esquecidos ou extras numa linha causam erros durante a interpretação. Se existir dois ou mais separadores antes de um campo, os campos anteriores serão considerados como nulos ou seja desconhecidos, mas serão interpretados.

O arquivo de dados do **Splat Visualizer** possui extensão *.data*.

7.3 O Arquivo de Configuração

O arquivo de configuração do **Splat Visualizer** consiste das seções **Input** e **View**. O arquivo de configuração possui a extensão *.splatviz*. Os arquivos padrões e as seções do arquivo de configuração serão detalhados a seguir.

7.3.1 Arquivos de Padrões

Assim que cada seção é encontrada, um arquivo de configuração especial é lido. Este arquivo é chamado de arquivo padrão. Ele é procurado da seguinte forma:

1. diretório */usr/lib/MineSet/splatviz*, onde estão os padrões do sistema.
2. diretório *~/.MineSet* (o usuário deve criar este diretório) que está na área do usuário, onde estão os padrões definidos pelo usuário.
3. diretório atual de trabalho.

Pode existir um arquivo de configuração em cada um dos diretórios citados acima, mesmo com os nomes iguais, porém, assim que vão sendo encontrados as definições de um arquivo sobrepõem às definições lidas anteriormente. E por fim, as opções dos arquivos de configuração sobrepõem a dos arquivos padrões.

7.3.2 A Seção Input

A seção **Input** contém o nome e o formato dos arquivos de dados. É nesta seção que é atribuído o significado de cada coluna do arquivo de dados. Um exemplo de uma seção **Input** e mostrado a seguir:

```
input {
        file "DEP.data";              #arquivo de dados
        string RGN;
        string RGD;
        string ANO_NASC;
        string RGD_PAI;
        string RGD_MAE;
        float VAL_DEPMP120;      #valor da DEP
        float AC_DEPMP120;       #acuracia da DEP
        float VAL_DEPDP120;
        float AC_DEPDP120;
        float VAL_DEPDP240;
        float AC_DEPDP240;
        float VAL_DEPDP365;
        float AC_DEPDP365;
        float VAL_DEPDP550;
        float AC_DEPDP550;
```

```
        float MGT;                  #merito genetico do animal
        int FAZENDA;                #id da fazenda
        options separator ':';
}
```

A sintaxe da sentença **file** é da forma **file "***nomearquivo.data***"**. Os dados são procurados primeiro (caso não seja dado o caminho absoluto) no diretório onde se encontra o arquivo de configuração, caso não seja encontrado então o diretório atual é procurado.

Na seção **Input** pode-se declarar sentenças de enumerações, **enum** que servem como os índices dos campos de arrays. A sentença **enum** tem as seguintes formas:

enum tipo nome **from** valor1 **to** valor2 **by** incremento;
enum tipo nome **from** valor1 **to** valor2 **across** incremento;
enum tipo nome { valor1, valor2, ..., valorN};

Por exemplo, as seguintes sentenças são equivalentes

```
enum int ano from 1900 to 1990 by 10;
enum int ano from 1900 to 1990 across 10;
enum int ano { 1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990 }
```

e representam um tipo **enum** ano que contém 10 números que vai de 1900 à 1990.

É possível especificar qual será o separador dos dados que serão lidos através da opção *separator* , cuja sintaxe é da forma:

```
options separator `char';
```

onde *char* representa o caracter que servirá como separador dos dados.

7.3.3 A Seção View

Esta seção descreve como os dados são mostrados, incluindo o mapeamento dos tamanhos, cores, eixos e outros. Os valores padrões para estas opções estão no arquivo */usr/lib/MineSet/splatviz/view.splatviz.options*. Um exemplo segue abaixo:

```
view
{
        color MGT, legend on;
        axis ANO_NASC, label "ANO DE NASCIMENTO",color "#a5a5ff";
        axis RGD_PAI, label "REGISTRO DO PAI" ,color "#a5a5ff";
```

```
        axis RGD_MAE, label "REGISTRO DA MAE", color "#a5a5ff";
}
```

Vários tipos de senteças podem ser utilizadas pelo **Splat Visualizer**.

A sentença **slider** identifica um coluna para ser utilizada como uma dimensão do **slider**. Esta sentença é utilizada em visualizações com animações. A sintaxe é:

slider nomedacoluna;

onde o nome da coluna foi declarado anteriormente na seção Input. Pode haver apenas 0, 1, ou 2 duas sentenças de slider. Sendo que a primeira aplica-se ao slider horizontal e a segunda ao slider vertical. E caso não seja especificada uma sentença então nenhuma animação será mostrada.

A sentença **opacity** descreve como uma coluna é mapeada para a opacidade dos splats. Esta sentença consiste de uma série de cláusulas separadas pos vírgulas:

opacity cláusula1, cláusula2,...

A sentença **color** descreve como os valores são mapeados para as cores. É constituída de várias cláusulas que podem ser separadas por vírgulas ou por sentenças múltiplas:

color cláusula1, cláusual2, ...

A sentença **axis** faz com que uma variável seja utilizada como um eixo em 3D. O valor das variáveis determinam onde as entidades são posicionadas nos eixos. O formato constitui de várias cláusulas separadas por vírgulas.

axis cláusula1, cláusula2,...

A sentença **summary** específica agregação de informação a ser calculada para todos os dados definidos pela posição do slider. Esta sentença é utilizada para colorir a janela de desenho no painel de controle da animação. Ela é composta por várias cláusulas separadas por vírgulas:

summary cláusua1, cláusula2, ...

A seção **View** do arquivo de configuração possui vários opções para controlar os parâmetros do display. Estas opções aparecem separadas por vírgulas:

options opção, opção, ...

Para obter maiores detalhes sobre os arquivos de configurações e de dados do **Splat Visualizer** recorrer *ao Appendix E: Creating Data and Configuration Files for the Splat Visualizer* em [1].

7.4 Exemplos dos Arquivos de Dados e Configuração do Splat Visualizer

Exemplos dos arquivos de dados e configuração do **Splat Visualizer** podem ser analisados e rodados no diretório */usr/lib/MineSet/examples/splatviz_examples* - este diretório é criado na instalação do **MineSet™**. Para rodar os exemplos, o usuário pode entrar com o comando **splatviz** (arquivo executável gerado na instalação) diretamente do prompt da shell do sistema operacional. O exemplo a seguir mostra como é iniciado o **Splat Visualizer,** considerando que o diretório atual seja o citado acima e o arquivo adultJob.*splatviz* resida neste diretório:

```
% splatviz adultJob.splatviz
```

O usuário pode rodar a aplicação através do **Tool Manager**. Um exemplo da tela da ferramenta **Splat Visualizer** é mostrado na figura 6.

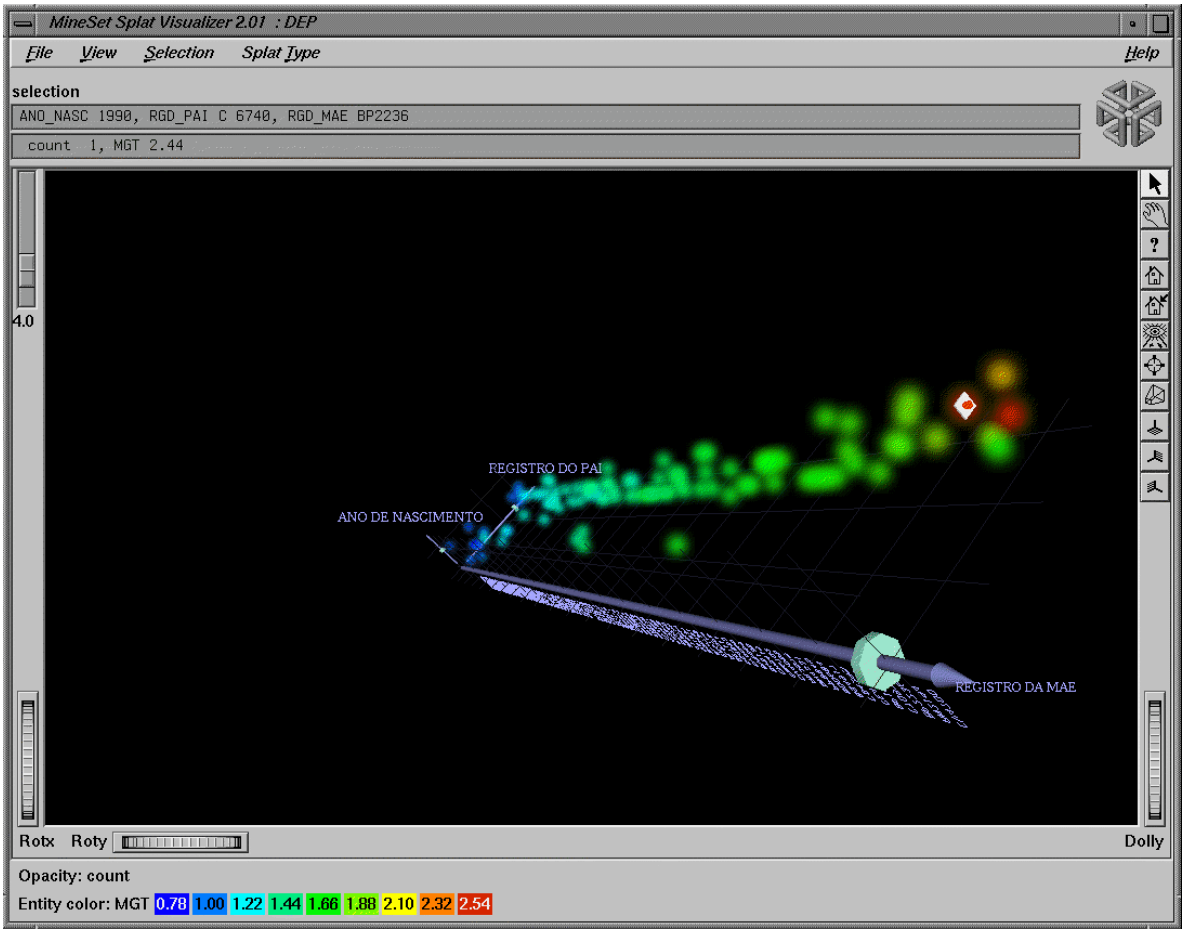

Figura 6. Visualização na ferramenta **Splat Visualizer**

8. Guia de Utilização do Rule Visualizer

Este guia de utilização, mostra como realizar visualizações de dados utilizando o **Rule Visualizer** sem utilizar o **Tool Manager**.

A ferramenta de visualização, **Rule Visualizer**, mostra graficamente as regras resultantes do Gerador de Regras de Associação. O resultados das regras geradas é mostrado em uma perspectiva tridimensional, tais regras estão organizados no **Arquivo de Regras** e que serão interpretadas pelo **Arquivo de Configuração**. Nesta Seção será visto como o arquivo de Configuração deve ser criado pelo usuário. O Arquivo de Regras é gerado pelo Gerador de Regras de Associação. Para saber mais a respeito da criação do arquivo de regras veja *Appendix F: Creating Data and Configuration Files for the Rules Visualizer* em [1].

8.1 O arquivo de Configuração

O arquivo de configuração descreve como os dados do arquivo de regras serão mostrados. Este arquivo consiste de três seções:

- Seção **Input** - específica as regras que serão utilizadas;
- Seção **Expression** - cria novos parâmetros de visualização;
- Seção **View** - específica como os dados são mostrados.

Um exemplo do arquivo de configuração do Rule Visualizer é mostrado em seguida:

```
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;
}
```

8.1.1 Arquivos de Padrões

Assim que cada seção é encontrada, um arquivo de configuração especial é lido. Este arquivo é chamado de arquivo padrão. Ele é procurado da seguinte forma:

1. diretório */usr/lib/MineSet/ruleviz*. Neste diretório estão os padrões do sistema.
2. diretório *~/.MineSet* (o usuário deve criar este diretório) que está na área do usuário. Neste diretório estão os padrões definidos pelo usuário.
3. diretório atual de trabalho.

Pode existir um arquivo de configuração em cada um dos diretórios citados acima, mesmo com os nomes iguais, porém, assim que vão sendo encontrados as definições de um arquivo sobrepõem às definições lidas anteriormente. E por fim, as opções dos arquivos de configuração sobrepõem a dos arquivos padrões.

8.1.2 Seção Input

A seção Input possui a forma:

```
input { file "nomedoarquivoderegras"; }
```

onde a sentença file descreve o arquivo que possui as regras geradas.

8.1.3 Seção Expression

Esta seção define os nomes dos campos que serão utilizados pelo arquivo de configuração. Novos campos são definidos em função dos campos já existentes. Por exemplo, a linha seguinte descreve a taxa (ratio) entre a predicabilidade e a predicabilidade esperada:

```
expression {float ratio= predictability/expected }
```

A seção de expressão utiliza nomes de campos definidos no arquivo de regras. Estes campos e seus respectivos tipos são mostrados a seguir:

Campo	**Tipo**
NumLHS	int
NumRHS	int
Prevalence	float
Predictability	float
Expected	float
LHS	string
RHS	string

Algumas funções também podem ser utilizadas:

- **divide**(x,y,z) - divide x por y, a não ser que y seja zero, o resultado é z; é equivalente a y == 0 ? z : x / y;
- **modulus**(x,y,z) - similar ao **divide**, porém para módulo.

8.1.4 Seção View

A seção **View** descreve como os dados serão apresentados. Uma regra é apresentada na junção dos itens do **left-hand-side (LHS)** e **right-hand-side (RHS)**. Esta seção permite definir o que será mostrado nesta junção.

Essa seção possui a seguinte forma:

```
view {sentençadoview, .... }
```

É possível ver barras, discos, e rótulos nas junções. Existem vários tipos de sentenças que podem ser utilizadas:

A sentença **Height** é utilizada para descrever como as regras são mapeadas às alturas dos discos e barras. Esta sentença consiste de várias cláusulas, separadas por vírgulas. Um exemplo é mostrado a seguir:

```
height  vendas, max 2.0
```

A sentença **Color** descreve como os valores são mapeados para as cores. O seu formato é similar à sentença **Height**, consistindo de várias cláusulas que podem ser separadas por vírgulas ou por sentenças múltiplas.

A sentença **Label** específica o nome da variável e cor para os rótulos que aparecem na frente das barras. Um exemplo é mostrado a seguir:

```
label predictability, color "green"
```

A sentença **Message** específica mensagens mostradas quando o ponteiro move-se sobre um objeto ou quando um objeto é selecionado. A sua sintaxe é parecida com a da função printf da linguagem C. Um exemplo de como pode ser utilizada é mostrada a seguir:

```
message "LHS: %s \NRHS: %s\predictability/expected: %.2f",
            LHS, RHS, predictability/expected;
```

A sentença **Item** descreve os nomes mostrados ao longo dos eixos LHS e RHS. Uma sentença possui a forma:

```
item colors <coresquerda> <cordireita>; ou item <off|on>;
```

A sentença **Grid** descreve o grid no qual as regras são mostrados. Esta sentença possui a forma:

grid color <cor>; ou grid <off|on>;

8.2 Exemplos dos Arquivos de Dados e Configuração do Rule Visualizer

Exemplos dos arquivos de regras e configuração do **Rule Visualizer** podem ser analisados e rodados no diretório */usr/lib/MineSet/examples/ruleviz_examples* - este diretório é criado na instalação do **MineSet™**. Para rodar os exemplos, o usuário pode entrar com o comando **ruleviz** (arquivo executável gerado na instalação) diretamente do prompt da shell do sistema operacional. O exemplo a seguir mostra como é iniciado o **Rule Visualizer,** considerando que o diretório atual seja o citado acima e o arquivo group.*ruleviz* resida neste diretório:

% ruleviz group.ruleviz

O usuário pode rodar a aplicação através do **Tool Manager**. Um exemplo da tela da ferramenta **Rule Visualizer** é mostrado na figura 7.

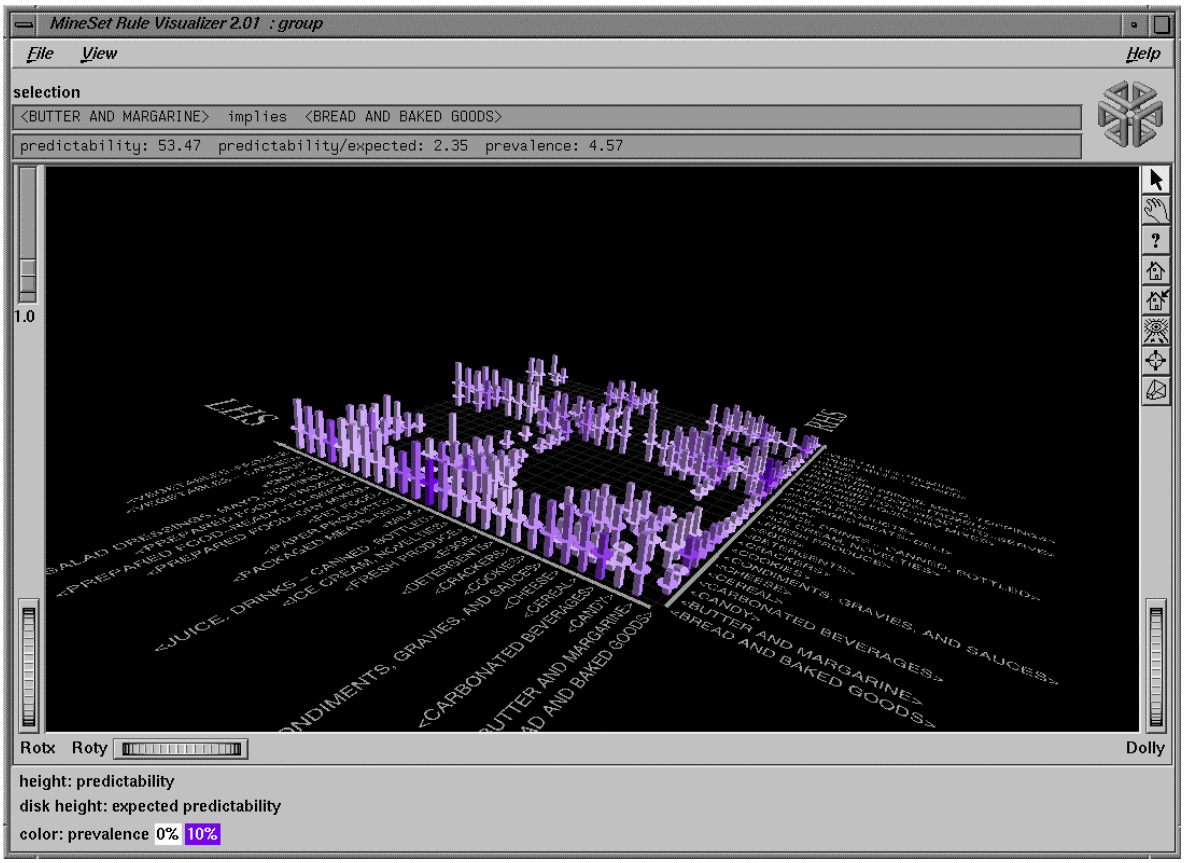

Figura 7. Visualização na ferramenta **Rule Visualizer**

9. Guia de Utilização do Evidence Visualizer

Este guia descreve como deve ser o formato de um arquivo de entrada de dados para a ferramenta de visualização **Evidence Visualizer**. Este arquivo é gerado automaticamente pelo **Tool Manager**, será mostrado apenas como ele está organizado.

9.1 O Arquivo de Dados

O arquivo de dados é composto de rótulos e atributos, acompanhado de *counts* e probabilidades. Estes componentes são utilizados para a construção dos gráficos. Este arquivo é conseguido executando-se o **Evidence Inducer** através do **Tool Manager**. O formato de um arquivo de dados é representado a seguir:

```
<tipo> "<rótulo>" <L>
"<rótulo1>" <count1> <probabilidade1>
"<rótulo2>" <count2> <probabilidade2>
:
"<rótuloL>" <countL> <probabilidadeL>

<M>

<tipo> "<atrib1>" <N1> <importância1>
"<valor1_1>" <count1_1_1> <prob1_1_1> ... <count1_1_L> <prob1_1_L>
"<valor1_2>" <count1_2_1> <prob1_2_1> ... <count1_2_L> <prob1_2_L>
:
"<valor1_N1>" <count1_N1_1> <prob1_N1_1> ... <count1_N1_L> <prob1_N1_L>

<tipo> "<atrib2>" <N2> <importância2>
"<valor2_1>" <count2_1_1> <prob2_1_1> ... <count2_1_L> <prob2_1_L>
"<valor2_2>" <count2_2_1> <prob2_2_1> ... <count2_2_L> <prob2_2_L>
:
"<valor2_N2>" <count2_N2_1> <prob2_N2_1> ... <count2_N2_L> <prob2_N2_L>
:
"<atribM>" <NM> <importânciaM>
"<valorM_1>" <countM_1_1> <probM_1> ... <countM_1_L> <probM_1_L>
"<valorM_1>" <countM_2_1> <probM_2_2> ... <countM_2_L> <probM_2_L>
:
"<valorM_NM>" <countN1_NM_1> <probM_NM_1> ... <countM_NM_L> <probM_NM_L>
```

onde L representa o número de valores de rótulos, M é o número de atributos, e N é o número de valores para o atributo i. Os < > indicam variável. O *count* representa o número de registros na tabela que possui um valor particular. A probabilidade é o número de counts por valor de atributo dividido pelo total do número de *counts*.

Para maiores detalhes *veja Appendix G: Format of the Evidence Visualizer's Data File* em [1].

9.2 Exemplos dos Arquivos de Dados para o Evidence Visualizer

Exemplos dos arquivos de dados do **Evidence Visualizer** podem ser analisados e executados no diretório */usr/lib/MineSet/examples/eviviz_examples* - este diretório é criado na instalação do **MineSet™**. Para rodar os exemplos, o usuário pode entrar com o comando **eviviz** (arquivo executável gerado na instalação) diretamente do prompt da shell do sistema operacional. O exemplo a seguir mostra como é iniciado o **Evidence Visualizer,** considerando que o diretório atual seja o citado acima e o arquivo adult-salary-evi.*eviviz* resida neste diretório:

```
% eviviz adult-salary-evi.eviviz
```

O usuário pode rodar a aplicação através do **Tool Manager**. Um exemplo da tela da ferramenta **Evidence Visualizer** é mostrado na figura 8.

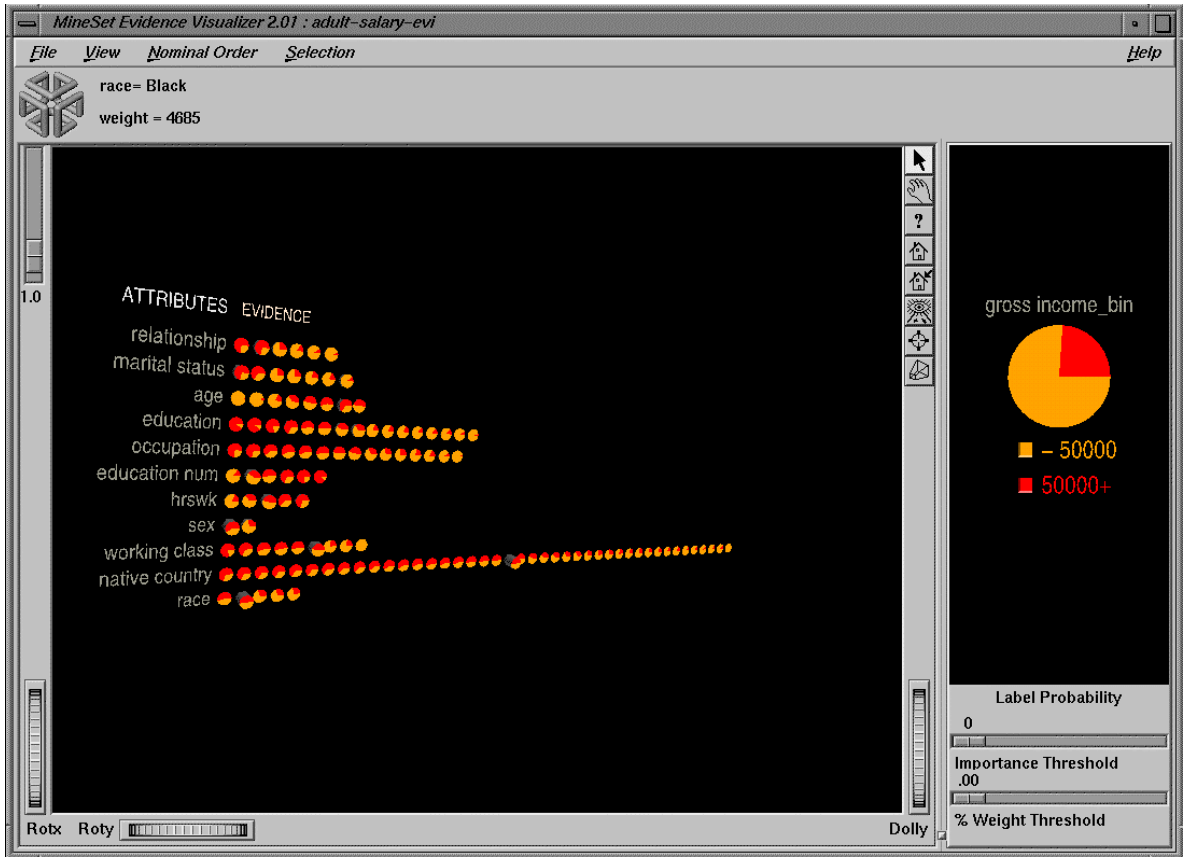

Figura 8. Visualização na ferramenta **Evidence Visualizer**

10. Tratamento de Nulos no MineSet

Nulos representam dados desconhecidos. O MineSet suporta nulos nas ferramentas de mineração assim como para as ferramentas de visualização. Nesta seção será visto como é o tratamento de nulos realizados pelo MineSet.

10.1 Semântica de Nulos

Os valores desconhecidos são tratados como nulos nos arquivos de dados. É possível associar diferentes semânticas para os nulos, porém, as mais comuns representam os nulos como valores perdidos ou desconhecidos. Um exemplo seria se o registro de uma pessoa fosse formado pelo Primeiro Nome, Nome do Meio e Último Nome, e caso os dados do Nome do Meio não forem conhecidos, então este campo seria considerado nulo.

Os valores nulos podem ocorrer nos dados por diversas razões: podem aparecer naturalmente nos dados como um meio de representar dados desconhecidos, ou podem ser resultantes de agregações.

10.2 Representação dos Nulos

Nos arquivos de dados, bem como nas ferramentas de visualização, os valores nulos são representados pelo caracter “?” (interrogação). Tomando como exemplo o registro de uma pessoa com Primeiro Nome, Segundo Nome e Último Nome, se o nome do meio for desconhecido, então o nome dele será representado em um arquivo de dados da seguinte maneira:

João ? Souza

A representação gráfica de nulos nas ferramentas de visualização varia de ferramenta para ferramenta.

10.3 Operações com Nulos

Dada a semântica que representa os nulos, então fica extremamente fácil entender expressões envolvendo valores desta natureza.

10.3.1 Expressões Aritméticas

Operações aritméticas que envolvem nulos sempre resultam em nulos. Por exemplo:

(5+?) resulta em ?
(6/?) resulta ?

10.3.2 Expressões Booleanas

Operações booleanas com nulos podem apresentar resultados diferentes, isto se deve a natureza dos números booleanos. Por exemplo:

? and FALSE resulta FALSE
? and TRUE resulta ?
? or FALSE resulta ?
? or TRUE resulta TRUE

10.3.3 Operações Relacionais

Operações relacionais envolvendo nulos sempre resultam em nulos. Porém, existem alguns casos particulares que devem ser enfatizados. Por exemplo:

? == ? resulta ?, não TRUE
? != ? resulta ?, não FALSE
? != x resulta ?, não FALSE

10.3.4 Testando Nulos

A função isNull() pode determinar se uma variável possui ou não o valor nulo. Por exemplo:
isNull(x) resulta TRUE se x possui um valor nulo ou FALSE caso contrário.

10.3.5 Agregações na Presença de Nulos

As agregações de colunas (SUM,AVG,MIN,MAX e COUNT) ignoram a presença do valor nulo. Um exemplo bem ilustrativo é o de um arquivo de dados representando o número de animais que uma pessoa possui, onde o esquema é representado por um registro contendo o Nome (pessoa) e Número (animais):

Nome	Número
João	3
Maria	?
Paulo	1
Pedro	0
Márcia	?

Então,

SUM(Numero) = 4
COUNT(Numero) = 3
AVG(Numero) = 1,33
MAX(Numero) = 3
MIN(Numero) = 0

10.3.6 Ordem de Ordenação com Nulos

Em uma seqüência ascendente, os valores nulos aparecem antes de valores não nulos, e após os valores não nulos em uma seqüência descendente.

Para maiores detalhes veja *Appendix I: Nulls in MineSet* em [1].

11. Bibliografia

[1] Manual de referência do MineSet.
http://www.sgi.com/Products/software/MineSet/solutions/

[2] Vários artigos descrevendo a tecnologia utilizada no MineSet estão disponíveis em http://www.sgi.com/Products/software/MineSet/tech/

[3] Michael Berry e Gordon Linoff. *Data Mining Techniques*. New York: John Wiley & Sons, 1997. ISBN 0-471-17980-9.
http://www.data-miners.com/http://www.data-miners.com/