UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Aruanda Simões Gonçalves

MINERAÇÃO DE DADOS PARA MODELAGEM DE DEPENDÊNCIA USANDO ALGORITMOS GENÉTICOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. João Bosco da Mota Alves
Orientador

Florianópolis, Abril 2001.

MINERAÇÃO DE DADOS PARA MODELAGEM DE DEPENDÊNCIA USANDO ALGORITMOS GENÉTICOS

Aruanda Simões Gonçalves

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Inteligência Artificial e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Dr. João Bosco da Mota Alves

Orientador

_____

Prof. Dr. Fernando A. Ostuni Gauthier

Coordenador

Banca Examinadora

_____

Prof. Dr. Luís Fernando Jacintho Maia

_____

Prof. Dr. Mauro Roisenberg

_____

Prof. Dr. José Lopes de Siqueira Neto

SUMÁRIO

“Any sufficiently advanced technology

is indistinguishable from magic."

Arthur C. Clarke

Ao meu pai, Artur Pojo,

dedico não apenas este trabalho, mas tudo

o que eu fizer de bom em minha vida.

AGRADECIMENTOS

Ao Prof. Dr. João Bosco da Mota Alves, por aceitar a orientação deste trabalho.

Aos professores Silvia Nassar, Paulo Borges e Rosvelter Coelho, pelos desafios que me ajudaram a selecionar este tema.

Ao Prof. Dr. Alex Freitas, pelo tempo que dedicou às discussões que tanto enriqueceram este trabalho.

Aos amigos Fabiano, João, Otávio, Rita, Marta e todos que contribuíram para o sucesso deste curso de Mestrado, tornando a experiência tão divertida.

A meus pais, Artur e Rosangela, exemplos de que o sucesso é fruto de amor, dedicação e caráter, e vem naturalmente ao encontro dos que não vivem para ele.

A meus irmãos, Ariana, Artur e Amanda, por sermos exatamente como somos, tão diferentes e tão unidos.

À Rede de Informática Ltda., que patrocinou este curso.

Aos Professores que fizeram parte da Banca Examinadora desta dissertação, pelas importantes contribuições.

Os dados utilizados para avaliação da ferramenta são de domínio público, disponibilizados pelo UCI Repository of Machine Learning Databases (MURPHY, 1994) .

RESUMO

O desafio da área de Descoberta de Conhecimento em Bancos de Dados, ou KDD, é analisar de forma eficiente e automática a grande massa de informações disponível, extraindo conhecimento útil. Neste trabalho, apresentamos GenMiner, uma ferramenta de Mineração de Dados para a tarefa de Modelagem de Dependência. Um algoritmo genético, método de otimização da Computação Evolucionária, foi desenvolvido para descobrir regras interessantes em bases de dados relacionais. A avaliação das regras é realizada individualmente, favorecendo regras com alta precisão e, preferencialmente, surpreendentes. A integração a bases de dados relacionais foi viabilizada pela codificação dos cromossomos como expressões em linguagem SQL. GenMiner foi avaliado usando uma base de dados de domínio público, com informações sobre diversos países e suas bandeiras.

ABSTRACT

The challenge of the KDD (Knowledge Discovery in Databases) research area is to efficiently and automatically analyze the available information, extracting useful knowledge from databases. We present GenMiner - a Data Mining tool for the Dependence Modeling task. GenMiner is a genetic algorithm based tool that searches for interesting rules involving correlated items on a relational database. Generated rules are evaluated on an individual basis, favoring precise and surprising rules. The genetic individual encoding provided relational database integration. Chromosomes in our GA are represented by SQL queries. GenMiner evaluation was based on a public domain database including information about countries and their flags.

1 INTRODUÇÃO

A busca pelo conhecimento tem papel de destaque em toda a história da humanidade. Nos tempos atuais, um grande volume de informações está armazenado em bases de dados de forma dispersa e muitas vezes redundante. Como constatado por FAYYAD et al (1996), a capacidade de armazenamento dos sistemas atuais ultrapassou a capacidade humana de analisar os dados. O desafio, portanto, é analisar de forma eficiente e automática esta massa de informações, extraindo conhecimento útil e novo. Diversas técnicas com este objetivo são classificadas como mineração de dados, ou *data mining*.

Grande parte da bibliografia sobre mineração de dados trata da descoberta de regras de associação, conforme proposto em 1993 por AGRAWAL et al. Regras de associação são de grande relevância às áreas de marketing e vendas. No entanto, não têm caráter de previsão, pois são apenas indicadores de co-ocorrência de itens. Seu objetivo é, portanto, descrever dados antigos, e não prever comportamentos futuros.

A tarefa de modelagem de dependência, por outro lado, propõe-se a descobrir padrões que sejam válidos não somente para os dados minerados, mas também para dados futuros.

Do ponto de vista de implementação, as tarefas que envolvem previsão são mais complexas, porque não são determinísticas. Como não há solução única, nem tampouco trivial, decidimos pelo uso de uma técnica não determinística, mas com grande potencial de otimização. Algoritmos genéticos (AG), baseados na teoria da evolução, foram anteriormente usados para auxiliar na mineração de dados em AMO et al (2000) e FREITAS (1997).

Segundo HAN et al (1992), "Algoritmos Genéticos utilizam-se de uma população de indivíduos pontuais para fazer a varredura da superfície de busca". O resultado de um algoritmo genético é a população vencedora, ou seja, um conjunto de soluções possíveis. No caso de mineração de dados, especificamente modelagem de dependência, o resultado seria um conjunto de regras interessantes descobertas pelo algoritmo genético.

Claramente, a questão principal é determinar se uma regra é interessante ou não. O uso de algoritmos genéticos reduz este problema a uma tarefa bem determinada: a especificação da função de avaliação do algoritmo genético. A função de avaliação, ou *fitness*, determina o critério usado na seleção de regras.

Para garantir a diversidade e realizar a evolução das regras pesquisadas, são utilizados os operadores genéticos de cruzamento e mutação.

Este trabalho objetiva o desenvolvimento de uma ferramenta de mineração de dados para modelagem de dependência em bases de dados relacionais, usando a técnica de algoritmos genéticos para a seleção das melhores regras.

No primeiro Capítulo, apresentamos a definição mais amplamente aceita de Mineração de Dados, no contexto do processo de Descoberta de Conhecimento em Bases de Dados. Descrições das principais tarefas de Mineração de Dados são também apresentados. O Capítulo encerra com uma breve discussão sobre integração de Mineração de Dados a Bancos de Dados Relacionais através da Linguagem SQL.

O segundo Capítulo apresenta a técnica evolucionária de Algoritmos Genéticos, utilizada neste trabalho para seleção das melhores regras. O Capítulo inclui uma visão geral da estrutura de um Algoritmo Genético (AG), a definição de dois operadores genéticos e considerações sobre as duas abordagens mais utilizadas para a representação de regras em um AG.

A implementação da ferramenta GenMiner é descrita no Capítulo 3. A codificação dos cromossomos e a função de avaliação utilizada são enfatizados. O Capítulo apresenta, ainda, aspectos de modelagem, incluindo a descrição das principais classes da ferramenta.

No Capítulo 4, as bases de dados utilizadas na avaliação da ferramenta são resumidamente descritas. A seguir, os resultados inicialmente obtidos são apresentados e discutidos. Apresentamos ainda as medidas tomadas para melhorar o desempenho do GenMiner e os resultados finais.

O Capítulo 5 contém as conclusões e planos para trabalhos futuros.

2 DESCOBERTA DE CONHECIMENTO E MINERAÇÃO DE DADOS

A definição de Mineração de Dados é, em muitos casos, confundida com a do processo de KDD. Neste trabalho, adotamos a visão de mineração de dados como um dos passos do processo de KDD, de acordo com a abordagem de FAYYAD(1996).

A seção 2.1 descreve sucintamente as etapas do processo de KDD. Na seção 2.2, a etapa de Mineração de Dados, de interesse neste trabalho, é apresentada com maiores detalhes.

2.1 Processo de KDD

Mineração de dados é parte essencial de um processo mais amplo, de descoberta de conhecimento em bases de dados (KDD, do inglês *Knowledge Discovery in Databases*).

Descoberta de conhecimento em bases de dados é o processo não trivial de identificação de padrões válidos, inéditos, potencialmente úteis e essencialmente compreensíveis (FRAWLEY et al,1991 apud FAYYAD et al, 1996).

FAYYAD complementa esta definição afirmando que uma noção chamada de *interesse* geralmente é tomada como medida geral do valor de um padrão descoberto. O objetivo de um processo KDD é, portanto, a descoberta de padrões interessantes.

Para que este objetivo seja atingido, algumas etapas devem ser cuidadosamente seguidas. Esta seção destaca algumas destas etapas. Para uma visão mais detalhada do processo, recomendamos a leitura de BRACHMAN e ANAND (1996).

2.1.1 Compreensão do Domínio da Aplicação

A interação com o usuário é indispensável durante a maior parte do processo de descoberta de conhecimento. O usuário deve estar intimamente envolvido para que o problema seja bem definido.

BRACHMAN e ANAND (1996) chamam de *analista de dados* a figura usuária participante do processo de KDD. Neste trabalho, usaremos como sinônimos os termos usuário e analista de dados.

A integração do analista de dados à equipe do processo de KDD facilita a compreensão do domínio da aplicação. E somente com amplo domínio da aplicação é possível estabelecer corretamente os objetivos do processo de KDD.

2.1.2 Preparação dos Dados

Antes que a base de dados seja submetida a uma ferramenta de mineração de dados, é necessário garantir a consistência das informações armazenadas.

Nesta fase, deve-se eliminar informações nulas ou redundantes, estabelecendo estratégias para o tratamento de valores não informados.

Ainda na fase de preparação, é feita a seleção das características mais adequadas para a representação dos dados. Com o objetivo de reduzir o espaço de busca do algoritmo de mineração de dados, os atributos considerados irrelevantes ou redundantes são eliminados. A seleção dos atributos é uma atividade importante e aconselhável, mas deve ser realizada cuidadosamente. Em alguns casos, a eliminação de atributos supostamente irrelevantes prejudica a descoberta de regras realmente surpreendentes.

Outra atividade de preparação é a codificação dos dados (AURELIO et al, 1999) [1]. É geralmente interessante transformar dados quantitativos em categóricos, estipulando-se uma identificação para cada intervalo. Por exemplo, o atributo Idade poderia ser representado por faixa etária. Esta medida reduz o tempo de processamento do algoritmo de mineração e favorece a obtenção de regras mais compreensíveis.

2.1.3 Especificação da Ferramenta de Mineração de Dados

Após a definição dos objetivos do processo de KDD, é direta a determinação da tarefa de mineração de dados adequada, seja para a melhor compreensão dos dados, ou para a previsão de dados futuros. A tarefa pode ser de classificação, associação, *clustering*, modelagem de dependência, etc. A seção 2.2 descreve com mais detalhes algumas das tarefas mais relacionadas ao contexto deste trabalho.

---

[1] AURELIO, Marco, VELLASCO, Marley, LOPES, Carlos Henrique. **Descoberta de Conhecimento e Mineração de Dados.** ICA – Laboratório de Inteligência Aplicada, DEE, PUC-Rio. Agosto, 1999.

Para a realização de cada tarefa, existem diversas técnicas disponíveis. Sempre considerando o domínio da aplicação e as informações obtidas sobre os dados durante a fase de preparação, deve-se selecionar o método de mineração de dados, a ser utilizado na fase seguinte.

2.1.4 Mineração dos Dados

Na fase de mineração de dados, o algoritmo selecionado é implementado e adaptado ao problema definido na primeira etapa (AURELIO et al, 1999).

A seguir, a ferramenta produzida é executada, em busca de padrões de interesse do processo. A representação dos padrões obtidos pode ser na forma de regras, árvores de classificação ou diversos outros formatos.

Todo o processo de KDD contribui para o sucesso da Mineração de Dados. A participação do analista de dados nas fases anteriores é um fator crucial para a obtenção de bons resultados.

A seção 2.2 descreve em maiores detalhes as atividades relacionadas à Mineração de Dados.

2.1.5 Consolidação do Conhecimento

Os resultados obtidos pela fase de mineração de dados só serão úteis ao usuário após sua devida interpretação e consolidação na forma de *conhecimento*.

Após a interpretação dos padrões, é muitas vezes necessário voltar a alguma das etapas anteriores. Dada a característica iterativa do processo de KDD, esta fase só será concluída quando os objetivos inicialmente definidos forem satisfatoriamente atendidos.

Em alguns casos, a documentação e apresentação do conhecimento descoberto ao usuário são suficientes. Para facilitar a compreensão das regras obtidas, é recomendável o uso de uma das técnicas de visualização disponíveis.

Freqüentemente, porém, os resultados de um processo de KDD devem ser a especificação de medidas a serem tomadas em decorrência do conhecimento descoberto. Outra possível saída de um processo de KDD é a especificação para a construção de aplicações para análise contínua dos dados (BRACHMAN et al, 1996).

Outro assunto a ser tratado na fase final é a resolução de possíveis conflitos entre o conhecimento descoberto e crenças anteriores.

2.2 Mineração de Dados

A aplicação de um algoritmo de busca de padrões na base de dados é vista como o principal passo de um processo KDD (FREITAS,1999a). Esta etapa, chamada Mineração de Dados ou *Data Mining*, é o principal assunto de pesquisa na área de descoberta de conhecimento.

Existem várias tarefas relacionadas a mineração de dados. Em geral, as tarefas pertencem a duas classes: uma objetivando a descrição, ou descoberta de padrões em dados antigos meramente para informação do usuário; e outra objetivando previsão, ou obtenção de regras que sejam válidas não somente para os dados utilizados no processo de mineração, mas também para dados futuros.

Nesta seção abordaremos três tarefas de mineração de dados, entre as mais utilizadas: descoberta de regras de associação, classificação e modelagem de dependência . A última é objeto deste trabalho, e o objetivo desta seção é esclarecer a diferença entre modelagem de dependência e as duas primeiras tarefas.

2.2.1 Regras de Associação

A mineração de regras de associação em bases de dados foi apresentada em 1993 por AGRAWAL et al. Uma regra de associação na forma X ⇒Y indica, de modo geral, que transações do banco de dados contendo um conjunto de itens de dados X tendem a conter também um outro conjunto de itens de dados Y.

Usando o exemplo clássico, a mineração da base de dados de vendas de um supermercado poderia descobrir a seguinte regra de associação: “30% das transações que contêm cerveja também contêm fraldas descartáveis, sendo que 2% do total de transações contêm ambos os itens”.

A confiabilidade da regra do exemplo acima é 30%. Confiabilidade mede a força da regra. O suporte da regra é 2%, expressando a fração de transações que contêm a união de todos os itens da regra.

Uma ferramenta para descoberta de regras de associação deve permitir que o usuário especifique valores mínimos de suporte e confiabilidade. Algoritmos derivados do proposto em 1993 foram apresentados em diversos trabalhos, como BAYARDO et al (1999a), BAYARDO et al (1999b) e PÔSSAS et al(2000).

Uma aplicação direta para a mineração de regras de associação como a do exemplo de fraldas e cervejas é a maximização de vendas, mediante a melhor disposição de produtos em lojas ou da segmentação baseada em padrões de compras.

2.2.2 Classificação

A tarefa de classificação envolve a previsão do valor de um atributo meta, com base nos valores de outros atributos, chamados previsores. O objetivo de um algoritmo de classificação é aprender uma função que classifique um item de dado em uma entre várias classes (HAND,1981 apud FAYYAD et al, 1996).

É necessário utilizar dois conjuntos distintos de itens de dados. Um dos conjuntos participa do processo de aprendizado da função de classificação. Ao contrário da tarefa de descoberta de regras de associação, no entanto, os resultados de um algoritmo de classificação devem ser avaliados em um conjunto de teste separado. Assim é possível estimar o desempenho deste algoritmo na classificação de dados futuros.

Aplicações típicas de classificação envolvem a identificação de bons candidatos a empréstimos bancários ou de consumidores em potencial para um determinado produto.

2.2.3 Modelagem de Dependência

Pelo caráter de previsão envolvido, modelagem de dependência é similar à tarefa de classificação. Neste caso, no entanto, não há um único, mas um conjunto de atributos meta. As regras obtidas são geralmente no formato:

SE (Antecedente) ENTÃO (Atributo-Meta = Valor),

onde Antecedente é uma expressão condicional que descreve um conjunto de tuplas da relação minerada; e Valor é um valor pertencente ao domínio do Atributo-Meta.

Em alguns casos o usuário deve especificar o conjunto de atributos que deseja classificar. Neste trabalho, como em FREITAS(1997) e outros trabalhos, o Atributo-Meta pode ser qualquer atributo não contido no antecedente da regra. Esta restrição evita a geração de regras indesejáveis, como por exemplo:

SE Sexo = M ENTÃO Sexo = M

Aplicações de modelagem de dependência abrangem sistemas especialistas da área médica, recuperação de informação e modelagem do genoma humano (FAYYAD et al, 1996).

2.3 Data Mining usando SQL

A representação de regras por expressões em linguagem SQL (*Structured Query Language*), promove a integração do algoritmo de mineração de dados a bancos de dados relacionais.

A vantagem mais direta é o uso da otimização de consultas do SGBD (Sistemas de Gerência de Banco de Dados). Mas outras características de um SGBD, como escalabilidade, robustez e concorrência (SARAWAGI et al, 1998) são uma grande contribuição a algoritmos de mineração de dados com tempo de processamento muito longo.

Em muitos algoritmos de mineração de dados, incluindo o Apriori (AGRAWAL et al, 1993), as regras descobertas permitem expressar simplesmente a coexistência de um determinado conjunto de itens de dados em um conjunto de tuplas. Expressões SQL, por outro lado, permitem a representação de quaisquer condições relacionais, por meio dos operadores {>,<., >=, <=, <>, =}, além de combinações lógicas entre condições por meio de operadores lógicos {AND, OR, NOT}.

Em SARAWAGI et al (1998) e THOMAS e SARAWAGI (1998), são apresentados algoritmos que suportam a representação de regras de associação em expressões SQL.

Neste trabalho, seguimos a proposta de FREITAS (1997) para a codificação dos indivíduos do algoritmo genético (cromossomos). As regras são representadas por expressões SQL, conforme será discutido na seção 4.1.

3 ALGORITMOS GENÉTICOS PARA MINERAÇÃO DE DADOS

É óbvia a importância da escolha do algoritmo mais adequado para a seleção das regras mais interessantes. Modelagem de dependência envolve previsão, o que a caracteriza como uma tarefa não determinística. Não é bem definida como a descoberta de regras de associação e outras tarefas descritivas. A principal diferença é que, no caso de previsão, não é possível garantir que as regras descobertas serão eficientes quando avaliadas em uma base de dados de teste, com casos não vistos durante o treinamento (FREITAS,2001[2]). Outro fator relevante é que, como há vários atributos de interesse, a modelagem de dependência trabalha com um espaço de busca consideravelmente maior que o da classificação (NODA et al,1999).

Algoritmos genéticos têm se mostrado uma técnica eficiente em diversas tarefas de mineração de dados, tendo sido utilizados em AMO et al(1996) para descoberta de padrões temporais e em FREITAS(1997) para classificação e modelagem de dependência.

A técnica, inspirada no processo de seleção natural da teoria evolucionária, utiliza um conjunto de indivíduos, chamados cromossomos. A cada geração, ou época, o algoritmo busca melhorar o conjunto de cromossomos da geração anterior. Ao contrário da maioria dos mecanismos de busca, que utilizam apenas um ponto para varrer uma superfície de busca, algoritmos genéticos utilizam-se de uma população de indivíduos pontuais que passa por um processo iterativo de evolução, aumentando a possibilidade de que uma solução satisfatória seja encontrada (MOREIRA et al, 2001) [3].

---

[2] FREITAS, Alex. A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery. A ser publicado em: A. Ghosh and S. Tsutsui. (Eds.) Advances in Evolutionary Computation. Springer-Verlag, 2001.

[3] MOREIRA, Fabiano, GONCALVES, Aruanda, BAPTISTA, João, BORGES, Paulo. Geogene: Layouts for Geophysics Observation. Belém: CESUPA/UFSC, 2001. Em fase de elaboração.

3.1 Estrutura de um AG

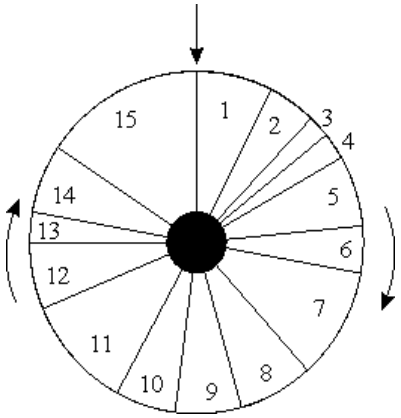
A menor unidade de representação de um AG é um *gene*. Chamamos *alfabeto* o conjunto de valores que um gene pode assumir. Um indivíduo, ou *cromossomo*, é formado por um conjunto (em geral um vetor) de genes. O número de cromossomos em um AG é fixo, e o conjunto formado por todos os cromossomos é chamado *população*.

As etapas de um AG são :

a) Gênesis: Na inicialização da população, os cromossomos são criados e recebem valores aleatoriamente selecionados do alfabeto.

b) Classificação (*Ranking)*: Cada um dos cromossomos é então avaliado de acordo com a função de avaliação, ou *fitness*. A função de avaliação determina quais cromossomos estão mais próximos de uma solução satisfatória (MOREIRA et al, 2001). A população é ordenada de acordo com a avaliação dos cromossomos.

c) Seleção dos mais aptos: Os cromossomos com melhor avaliação têm proporcionalmente mais chances de reprodução. Um método comum para determinar a probabilidade de um cromossomo se reproduzir consiste em dispor os cromossomos em uma roleta. O espaço que cada cromossomo ocupa na roleta é proporcional ao valor de sua função de avaliação, como mostrado na Figura 1. Pelo critério de seleção elitista, além das maiores chances de reprodução, os melhores cromossomos garantem sua sobrevivência até a próxima geração. Já os piores cromossomos são eliminados para dar lugar à descendência dos cromossomos selecionados para reprodução.



Classificação	
1º	15
2º	7
3º	11
4º	5
5º	8
6º	9
7º	1
8º	14
9º	12
10º	2
11º	10
12º	6
13º	13
14º	4
15º	3

Figura 1 Seleção de Cromossomos por Roleta

d) Evolução: De acordo com as probabilidades de reprodução atribuídas a cada cromossomo, os operadores genéticos, apresentados na seção 3.2, são aplicados. Os novos cromossomos, são produzidos em quantidade tal que o número total de cromossomos da população mantenha-se constante.

e) Término: Duas condições, dependendo da aplicação, podem encerrar o processo. Em alguns casos, um valor mínimo é estabelecido para a função de avaliação. O AG continua o processamento até que um cromossomo seja avaliado de forma equivalente ou superior ao valor estabelecido. Em geral, no entanto, o objetivo do AG é um conjunto de boas soluções, e não uma única solução específica. Nestes casos, o recomendado é que a condição de término do AG seja o número máximo de gerações. Se a condição de término especificada não tiver sido atingida, o processo continua a partir da etapa b).

3.2 Operadores Genéticos

Os AG baseiam-se na evolução de uma população inicial de cromossomos para a busca de soluções. A cada iteração, novos cromossomos são gerados para a nova população por meio de diversos operadores genéticos, entre os quais a mutação e o cruzamento são os mais comuns.

3.2.1 Cruzamento

A forma mais utilizada de cruzamento, ou *crossover,* é aplicada sobre dois cromossomos pais, gerando dois cromossomos filhos. O método seleciona aleatoriamente um ponto de corte, e os genes localizados após este ponto nos cromossomos pais são trocados nos cromossomos filhos. A Figura 2 ilustra o cruzamento de cromossomos representados por vetores com valores binários.

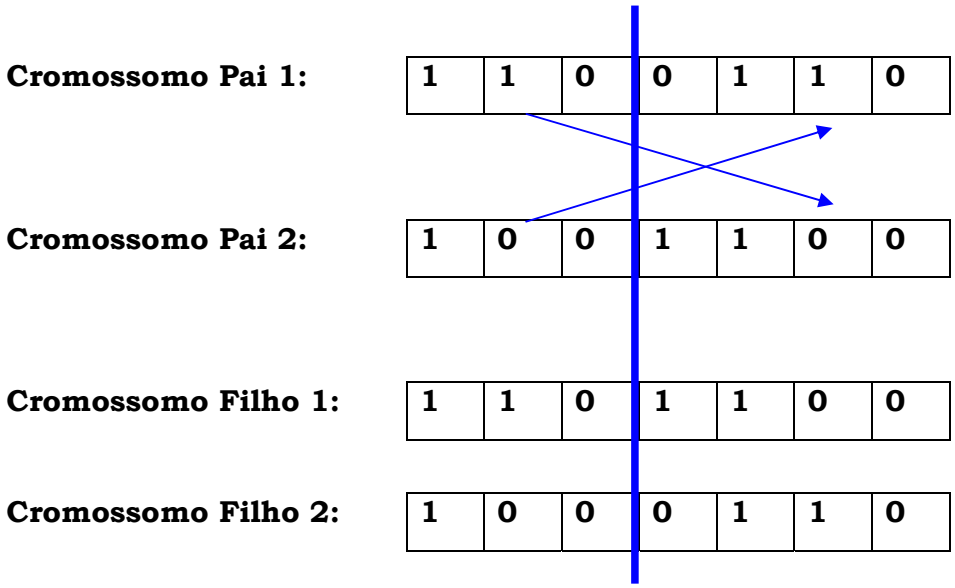


Figura 2 Cruzamento aplicado a dois cromossomos (Pais) para gerar dois novos cromossomos (Filhos).

O cruzamento é em geral o operador mais enfatizado em AG. A evolução se baseia em combinar dois indivíduos bem avaliados para gerar, possivelmente, uma descendência, ou *offspring*, ainda melhor. Caso o contrario ocorra, os descendentes indesejáveis serão eliminados na próxima geração.

3.2.2 Mutação

Para garantir que todos os valores do alfabeto possam participar do processo, a mutação é aplicada a um conjunto de cromossomos aleatoriamente selecionados. Na mutação, um dos genes do cromossomo, aleatoriamente selecionado, é trocado por outro valor do alfabeto, também selecionado aleatoriamente.

É importante frisar que, quando o critério de seleção elitista é aplicado, como acontece na maioria dos AG, um grupo formado pelos melhores cromossomos é poupado de participar da mutação.

A mutação também é um dos mecanismos utilizados para evitar a convergência prematura da população para uma solução. A convergência é causada pelo critério elitista dos AG. Quando um cromossomo tem avaliação muito maior que a dos outros, a tendência é que este seja muito freqüentemente selecionado na roleta para o cruzamento. O resultado é que grande parte da próxima geração descenderá deste cromossomo.

3.3 Representação de Regras

Existem, em geral, duas abordagens para a representação das regras em um Algoritmo Genético para Mineração de Dados. Pela abordagem de Michigan, cada cromossomo representa uma regra, ao passo que pela abordagem de Pittsburgh cada cromossomo representa um conjunto de regras (FREITAS,2001).

Naturalmente, a escolha entre uma das abordagens depende do objetivo da mineração de dados. Em alguns casos, o resultado deve ser um pequeno conjunto de regras muito interessantes ou surpreendentes. As regras devem ser avaliadas individualmente e, para tanto, a abordagem de Michigan deve ser adotada.

Por outro lado, para alcançar uma solução completa para uma tarefa de classificação, por exemplo, regras individualmente interessantes não são suficientes. Neste caso, deve-se adotar a abordagem de Pittsburgh, enfatizando a interação entre as regras.

A abordagem de Pittsburgh é consideravelmente mais simples, exigindo menor tempo de processamento. A abordagem de Michigan, por sua vez, exige operadores genéticos mais complexos.

Neste trabalho, a abordagem adotada foi a de Michigan, pois desejamos descobrir regras interessantes e, se possível, surpreendentes. Para este objetivo a qualidade de cada regra deve ser isoladamente avaliada.

4 GENMINER: AG PARA MODELAGEM DE DEPENDÊNCIA

Esta seção apresenta nosso AG para modelagem de dependência. Seguindo a abordagem de Michigan, cada cromossomo do GenMiner representa uma regra, no formato:

SE <uma condição é verdadeira> ENTÃO <o valor do Atributo-Meta é V>

Na seção 4.1, apresentamos a maneira pela qual uma regra neste formato é codificada em um cromossomo de um algoritmo genético. Para construção da regra, e para cálculo da precisão dos cromossomos, é necessário montar uma matriz de confusão, apresentada na seção 4.2. O processo de construção da regra é explicado na seção 4.3. O cálculo da função de avaliação, assim como um critério adicional usado para a classificação dos cromossomos são detalhados na seção 4.4. A seção 4.5 encerra o capítulo com a apresentação dos operadores genéticos utilizados.

4.1 Codificação do Cromossomo

A codificação dos cromossomos foi baseada em FREITAS(1997). Cada cromossomo é formado por uma expressão SQL, no formato da Figura 3. O ATRIBUTO-META é selecionado aleatoriamente entre os atributos da RELAÇÃO. A CONDIÇÃO é uma combinação lógica de expressões relacionais, na sintaxe da linguagem SQL. A expressão Count(*) determina o número de tuplas que satisfazem a CONDIÇÃO, para cada valor do ATRIBUTO-META.

```
Select  ATRIBUTO-META, Count (*)
From  RELAÇÃO
Where CONDIÇÃO
```

Figura 3 Expressão SQL para a codificação dos cromossomos

4.1.1 Construção de Expressões SQL

As expressões SQL são criadas automaticamente, selecionando-se de forma aleatória cada atributo e literal utilizado. Assim é garantida a diversidade das expressões.

As condições foram codificadas no formato de árvore. Um cromossomo está relacionado a uma expressão condicional, que pode ser terminal ou não. Expressões terminais têm formato:

<ATRIBUTO> <OP-RELACIONAL><VALOR>

Para a construção de uma expressão terminal, o ATRIBUTO é selecionado aleatoriamente do conjunto de atributos da relação. Em seguida, o VALOR é selecionado, também de forma aleatória, do domínio de valores existentes para ATRIBUTO. Para OP-RELACIONAL, os valores possíveis são {=, <,>,<=,>=,<>}, sendo que o valor = tem mais chances de ser selecionado.

A probabilidade do valor = fazer parte de uma condição é um parâmetro da ferramenta. O usuário pode ajustar este parâmetro de acordo com a quantidade de atributos categóricos na relação alvo. Por exemplo, se a ferramenta for utilizada para minerar uma relação que não inclua nenhum atributo contínuo, como Salário ou Idade, o ideal é atribuir 1 para este parâmetro.

Expressões não terminais têm formato diferente:

<OP-LOGICO><EXPRESSÃO1>{<EXPRESSÃO2>}.

Uma expressão não terminal liga uma ou duas outras expressões a um operador lógico, cujo domínio é {AND,OR,NOT}. A segunda expressão não ocorre somente no caso do operador lógico ser NOT. As duas novas expressões também podem ser terminais ou não terminais.

O algoritmo decide aleatoriamente o tipo de cada expressão criada. A probabilidade de uma nova expressão ser terminal ou não é dada pela taxa de profundidade, um parâmetro da ferramenta. Este parâmetro foi introduzido para controle da complexidade das regras, evitando condições muito longas. A utilização da taxa de profundidade é vantajosa em relação a especificação de um número máximo de

condições relacionais, pois permite, na verdade, a ocorrência de regras de todos os tamanhos. No entanto, regras mais curtas são favorecidas, pois são em geral consideradas mais simples, compreensíveis para o usuário, e em geral consideradas mais interessantes (FREITAS, 1999b). Em nossos testes, a taxa de profundidade especificava que 30% das expressões seriam não terminais.

4.2 Matriz de Confusão

A avaliação da expressão da Figura 3 sobre RELAÇÃO é o primeiro passo para o cálculo da função de avaliação do cromossomo. A partir dos valores retornados pela expressão Count(*), a Matriz de Confusão da Figura 4 é montada.

	V_1	…	V_N	Total
Satisfazem CONDIÇÃO	C_{11}	…	C_{1N}	C_{1+}
Não Satisfazem CONDIÇÃO	C_{21}	…	C_{2N}	C_{2+}
Total	C_{+1}	…	C_{+N}	C_{++}

Figura 4 Matriz de Confusão

A matriz tem ordem 3 x N+1, onde N é o número de valores distintos do Atributo-Meta. Na primeira linha, os valores C_{1j}, j=1…N, correspondem ao número de tuplas que satisfazem a CONDIÇÃO e contêm o valor V_j para o Atributo-Meta. Na segunda linha, os valores C_{2j}, j=1…N, indicam o número de tuplas que não satisfazem a CONDIÇÃO e contêm o valor V_j para o Atributo-Meta. A matriz é estendida com totais de tuplas que satisfazem ou não a CONDIÇÃO, e totais de tuplas com cada valor do Atributo-Meta.

4.3 Construção da Regra

A expressão SQL em um cromossomo representa uma regra do tipo:

SE <CONDIÇÃO> ENTÃO <ATRIBUTO-META = V>

A CONDIÇÃO e o ATRIBUTO-META são determinados diretamente pelo conteúdo da expressão SQL. O valor V, no entanto, só é determinado após a avaliação da expressão SQL e montagem da Matriz de Confusão.

Com o objetivo de maximizar a qualidade da regra, escolhemos o valor Vj correspondente ao maior valor Count(*) da primeira linha da matriz de confusão.

Um exemplo típico de construção de regra em nosso AG seria o seguinte. Considerando uma relação contendo informações sobre bandeiras de diversos países, um cromossomo poderia ser formado pela com a expressão abaixo.

Select RELIGIAO, COUNT(*) From BANDEIRA

Where TEXTO = Sim AND VERDE = Sim

A regra correspondente procuraria determinar a religião de países cuja bandeira apresenta um texto qualquer e a cor verde. A regra parcialmente montada seria:

Se (TEXTO = Sim) E (VERDE = Sim)

Então RELIGIAO = ???

Após a montagem da matriz de confusão, o algoritmo determinaria o valor de RELIGIAO para maximizar a avaliação do cromossomo. A regra final teria o formato abaixo.

Se (TEXTO = Sim) E (VERDE = Sim)

Então RELIGIAO = Católica.

4.4 Função de Avaliação

Para realizar a classificação dos cromossomos a cada geração, precisamos definir uma função de avaliação que indique as regras mais interessantes. Dois critérios são usualmente considerados para classificar uma regra como interessante: se ela é precisa e se ela é surpreendente.

O primeiro critério é privilegiado pelo GenMiner, pois para que uma regra desperte o interesse do usuário, é preciso acima de tudo que haja segurança de que os resultados da avaliação da regra serão precisos. Mas quando a função de avaliação leva em consideração apenas a precisão, a tendência é que as regras encontradas sejam de conhecimento prévio do usuário.

Para obter regras realmente interessantes, deve-se buscar regras que reflitam padrões diferentes dos que o usuário poderia supor. SILBERCHATZ (1996) faz uma

explanação sobre descoberta de regras surpreendentes, apresentando métricas subjetivas, que envolvem o conhecimento prévio do usuário sobre o domínio. Neste trabalho, usamos uma única métrica objetiva para determinar a surpresa relacionada à regra. Outras métricas possíveis, também objetivas, foram sugeridas por FREITAS(1998).

4.4.1 Precisão

Para medir a precisão da regra, a função de avaliação deve considerar a quantidade de acertos e erros. Quando uma regra é usada para prever o valor de um atributo meta para um exemplo, quatro diferentes resultados podem ser obtidos, dependendo do valor previsto e do valor real do atributo meta neste exemplo (FIDELIS, 2000):

a) Verdadeiro Positivo (Vp): a condição da regra é satisfeita e o valor do atributo meta corresponde ao previsto.

b) Verdadeiro Negativo (Vn): a condição da regra não é satisfeita e o valor do atributo meta não corresponde ao previsto.

c) Falso Positivo (Fp): a condição da regra é satisfeita, mas o valor do atributo meta não corresponde ao previsto.

d) Falso Negativo(Fn): a condição da regra não é satisfeita, mas o valor do atributo meta corresponde ao previsto.

FIDELIS (2000) e BOJARCZUK(2000), determinam a função de avaliação de seus algoritmos genéticos pelo produto de dois indicadores, conforme a Equação 1. Os indicadores Sens (Sensibilidade) e Esp (Especificidade) são baseados no resultado da avaliação da regra. O objetivo da métrica de precisão é maximizar ambos os indicadores.

Equação 1:

$$\textbf{Precisão} = \textbf{Sens} * \textbf{Esp}$$

onde:

Equação 2:

$$\mathbf{Sens = Vp / (Vp + Fn)}$$

Equação 3:

$$\mathbf{Esp = Vn / (Vn + Fp)}$$

Os valores da Matriz de Confusão da Figura 4 são utilizados para calcular os indicadores de Sensibilidade e Especificidade. Considerando que o valor previsto para o atributo meta seja V_i, temos:

$$Vp = C_{1i}$$

$$Fn = C_{2i}$$

$$Vn = C_{2+} - C_{2i}$$

$$Fp = C_{1+} - C_{1i}$$

Portanto, a medida de precisão da Equação 1 depende do número de tuplas enquadradas em cada célula da matriz de confusão. Uma versão simplificada desta matriz para efeito de determinação da qualidade da regra, nestes critérios, é mostrada na Figura 5.

Número de Tuplas em que	Valor é o previsto	Valor é diferente do previsto
CONDIÇÃO é verdadeira	Vp	Fp
CONDIÇÃO é falsa	Fp	Vn

Figura 5 Matriz de Confusão Simplificada

4.4.2 Surpresa

A outra métrica utilizada é baseada no ganho de informação dos atributos contidos no antecedente da regra para o atributo meta. Quando a especificação de um valor para um atributo do antecedente da regra contribui consideravelmente para a determinação

do valor do atributo meta, dizemos que o ganho de informação deste atributo é alto FREITAS(1999a).

Por exemplo, em uma regra sobre alistamento militar, o sexo é um atributo com alto ganho de informação. Para indivíduos do sexo feminino, a probabilidade de alistamento é consideravelmente menor do que para indivíduos do sexo masculino. Podemos afirmar, então, que ao sabermos o sexo do indivíduo, temos um alto ganho de informação sobre o alistamento militar.

A princípio, poder-se-ia considerar desejável a presença de atributos com alto ganho de informação. No entanto, normalmente estes atributos indicam padrões já conhecidos pelo usuário. No exemplo do alistamento militar, uma regra dizendo que pessoas do sexo feminino em geral não se alistam no exército não seria uma regra tão interessante. Por outro lado, o sexo pode ser um atributo com baixo ganho de informação para a formação do indivíduo. Justamente pelo fator surpresa, uma regra afirmando que pessoas do sexo feminino em geral fazem pós graduação pode ser muito mais interessante.

Consideramos, portanto, que regras envolvendo atributos com alto ganho de informação não são, em geral, surpreendentes (FREITAS,1999a). Uma boa medida de atributos surpreendentes em uma regra é fornecida pela equação abaixo (NODA,1999).

Equação 4:

$$GanhoInfo(A) = Info(M) - Info(M|A)$$

A Equação 4 permite o cálculo do ganho de informação do atributo A quando o atributo meta é M. As sub-equações são definidas abaixo.

Equação 5:

$$Info(M) = \sum_{j=1}^{n} \Pr ob(M) \log_2 \Pr ob(M)$$

Equação 6:

$$Info(M|A) = \sum_{i=1}^{m} \Pr ob(A)(-\sum_{j=1}^{n} \Pr ob(M|/A) \log_2 \Pr ob(M|A))$$

onde *n* é o número de valores para o atributo meta e *m* é o número de valores para o atributo A.

Portanto, para um atributo meta M , a medida de surpresa de uma regra é dada pela Equação 7, onde *r* é o número de atributos ocorrendo no antecedente da regra.

Equação 7:

$$Surpresa = \frac{r}{\sum_{k=1}^{r} GanhoInfo(A)}$$

4.4.3 Utilizando a métrica Surpresa para a Classificação

O GenMiner utiliza a métrica Surpresa da Equação 7 como critério de desempate entre regras, durante a fase de classificação dos cromossomos.

Segundo FREITAS(1999a), se duas regras tiverem a mesma avaliação pelos demais critérios, a regra que contiver atributos com baixo ganho de informação é mais interessante, ou surpreendente, que a regra contendo atributos com alto ganho de informação.

Portanto, para o caso de duas regras com mesma precisão, a regra com maior valor para a métrica Surpresa deve ser mais bem classificada. O problema é que dificilmente dois cromossomos são avaliados com valores idênticos. Para favorecer ainda mais as regras mais surpreendentes, foi especificada uma diferença mínima entre os valores da precisão de dois cromossomos para que se possa afirmar que um é maior que o outro. A diferença mínima é um parâmetro do sistema, e em nossos testes utilizamos o valor 0.05.

Na verdade, portanto, a função de avaliação dos cromossomos é dada pela medida de precisão da Equação 1. No entanto, durante a classificação, a medida de surpresa é levada em consideração. Portanto, para o GenMiner, nem sempre a ordem dos cromossomos após a classificação corresponderá à ordem decrescente de precisão. Com freqüência, cromossomos com atributos de baixo ganho de informação subirão algumas posições na classificação.

4.5 Operadores Genéticos

Dois operadores genéticos foram utilizados para a evolução das regras: cruzamento e mutação. O objetivo do primeiro é promover a interação entre as melhores regras, formando possivelmente regras ainda melhores. A mutação traz diversidade à população, facilitando a participação de todos os atributos da relação minerada como atributo meta.

4.5.1 Mutação

GenMiner utiliza dois operadores de mutação: um para mutação de atributo e outro para mutação de operador.

Em ambos os casos, a seleção do cromossomo que sofrerá mutação é realizada de forma aleatória.

A taxa de mutação é um parâmetro da ferramenta que indica quantos cromossomos sofrerão mutação a cada geração. Em nossos testes, foram utilizadas taxas de mutação de 2% ou 5%.

Em seguida, é preciso decidir qual dos dois operadores utilizar. Entre os cromossomos selecionados para mutação, outro parâmetro da ferramenta especifica quantas vezes o operador de mutação de atributo é utilizado. Em nossos testes, utilizamos taxa de mutação de atributo de 50%.

No entanto, ressaltando o critério elitista de AG, um grupo de melhores cromossomos são poupados da mutação. Para garantir que os melhores cromossomos sejam levados sem modificação alguma para a próxima geração, foi inserido um novo parâmetro, a taxa de preservação. A taxa de preservação utilizada em nossos testes foi de 20%.

4.5.1.1 Mutação de Atributo

O operador de mutação de atributo é utilizado para garantir que todos os atributos da relação minerada tenham chance de ser atributo meta.

A mutação, neste caso, troca o atributo meta de uma regra por outro atributo aleatoriamente selecionado do domínio da relação.

Um problema relacionado à mutação é a possibilidade do novo atributo meta já fazer parte do antecedente da regra em questão. Como solução, o algoritmo prevê que a operação de mutação será repetida até que um atributo que não faça parte do antecedente da regra seja selecionado.

Após a seleção do novo atributo meta, é preciso determinar o valor classificado pela regra. Este processo é idêntico ao da construção de regras (seção 4.3), envolvendo a montagem da matriz de confusão e a determinação do valor relacionado ao atributo meta que maximize a avaliação para a condição existente. A Figura 6 apresenta um exemplo de regra passando pelo processo de mutação de atributo.

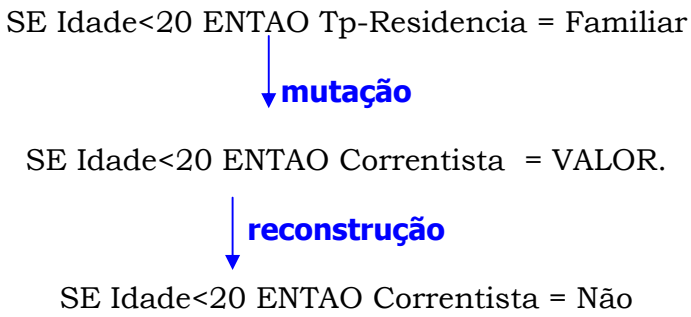


Figura 6 Operador de Mutação de Atributo

4.5.1.2 Mutação de Operador

A mutação de operador lógico

O processo de mutação seleciona aleatoriamente um novo operador lógico (AND, OR ou NOT). Este operador substitui o anterior na expressão condicional da regra representada pelo cromossomo selecionado para mutação.

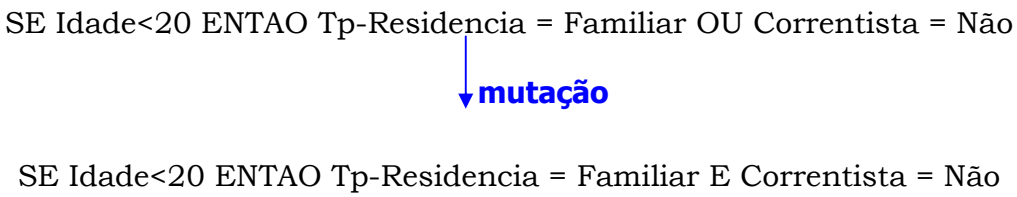


Figura 7 Operador de Mutação de Operador

4.5.2 Cruzamento

Enquanto a mutação baseia-se no atributo meta da regra, o cruzamento é baseado no seu antecedente, ou seja, na expressão condicional relacionada à regra. Após a seleção de dois cromossomos para o cruzamento, através do método de roleta da Figura 1, parte das condições de suas regras são trocadas.

O objetivo é combinar expressões condicionais de cromossomos bem classificados para produzir novos cromossomos, diferentes dos pais, com possibilidade significativa de boa colocação na próxima classificação.

As expressões condicionais, terminais ou não terminais, são codificadas no cromossomo em forma de árvore, como explicado na seção 4.1.1. O cruzamento do GenMiner troca as condições à esquerda do nó principal de cada expressão. No caso de expressões terminais, a única condição é trocada.

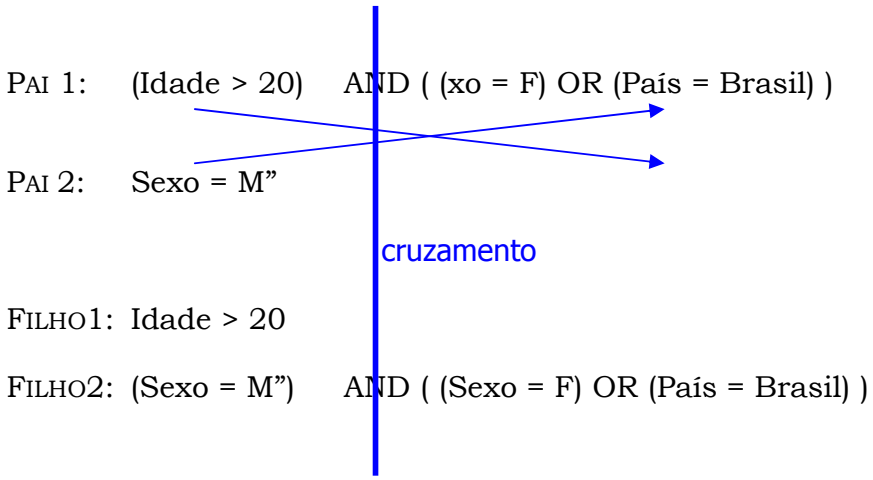


Figura 8 Cruzamento entre expressões condicionais

Um problema geralmente associado ao operador de cruzamento é a geração de expressões indesejáveis. Na Figura 8, a expressão Filho2 contém inconsistência, ao prever sexo tanto feminino quanto masculino. No entanto, considerando que já na construção original das regras os atributos e valores são selecionados aleatoriamente, este problema não é exclusivamente causado pelo cruzamento. A geração de regras indesejáveis, inconsistentes ou apenas não interessantes faz parte do processo de

evolução. Estas regras terão provavelmente um mau desempenho no momento da classificação, sendo naturalmente eliminadas da próxima geração.

Por outro lado, um segundo problema relacionado ao cruzamento exige a intervenção do algoritmo. Trata-se do caso da expressão condicional em um cromossomo filho incluir o atributo meta. Como solução a este problema, utilizamos o operador de mutação (FREITAS, 1997) para substituir o atributo meta por um novo atributo adequado. Neste caso específico, o cromossomo filho terá um atributo meta diferente dos pais.

4.6 Implementação

Nesta seção apresentaremos alguns detalhes da implementação do GenMiner. Não é nosso objetivo apresentar a documentação completa das fases de desenvolvimento, mas apenas destacar alguns aspectos técnicos relevantes à compreensão do trabalho.

A análise do software foi feita em UML (Unified Modeling Language). A Figura 9 contém o diagrama de classes em que se baseou o GenMiner. Nas sessões seguintes apresentaremos cada uma das classes do diagrama.

As classes relacionadas a Ag foram reutilizadas do projeto Geogene (MOREIRA et al, 2001[4]).

O GenMiner foi desenvolvido em Java, utilizando a máquina virtual Java da Microsoft, Microsoft Virtual Machine (MVM). A extensão para Java puro, no entanto, é relativamente simples. A escolha do ambiente proprietário foi feita apenas para acelerar o processo de desenvolvimento. Os recursos da MVM foram utilizados exclusivamente para a interface, para a ordenação no método ranking da classe Geração, e para o acesso relacional aos dados.

---

[4] MOREIRA, Fabiano, GONCALVES, Aruanda, BAPTISTA, João, BORGES, Paulo. Geogene: Layouts for Geophysics Observation. Belém: CESUPA/UFSC, 2001. Em fase de elaboração.

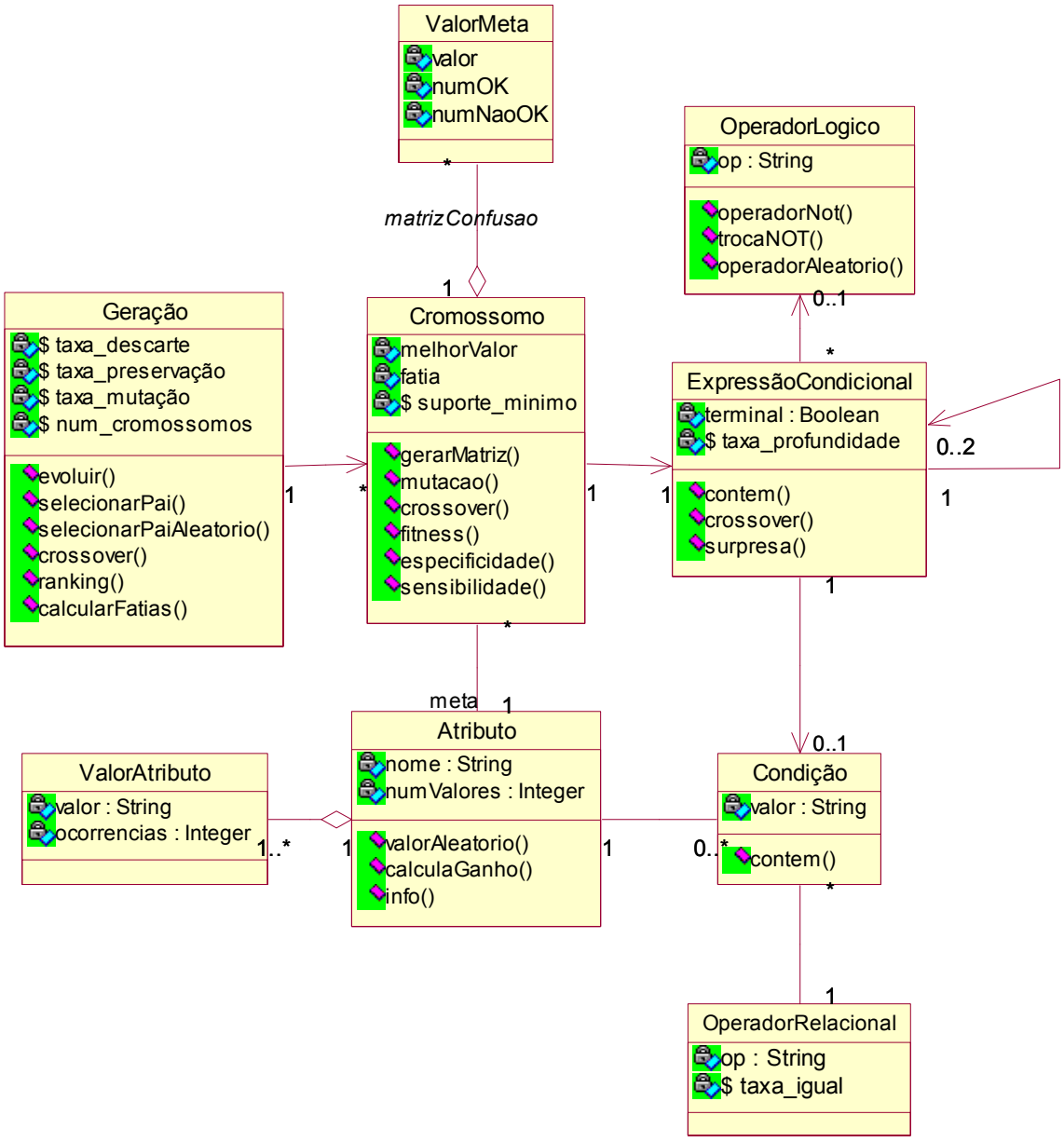


Figura 9 Diagrama de Classes

4.6.1 Classe Geração

A classe Geração está associada a um conjunto (um vetor) de cromossomos, representando a população de indivíduos que passará pelo processo de evolução.

O número de cromossomos é um parâmetro especificado em tempo de execução. Em nossos testes, utilizamos 50, 70 ou 100 cromossomos.

O método principal, **evoluir**, é processado a cada iteração do processo evolutivo. Inicialmente, o número de cromossomos a serem descartados da próxima geração é dado pela taxa de descarte. Para cada cromossomo descartado, dois cromossomos são selecionados para o cruzamento.

A seleção é implementada por uma chamada ao método **selecionarPai**, baseado no método de roleta simples, pelo qual quanto maior a avaliação de um cromossomo, maior sua chance de seleção.

O método ***crossover*** da classe Geração é chamado para cada par de cromossomos selecionados. Este método cria clones, ou filhos idênticos aos cromossomos pais, e faz uma chamada ao método ***crossover*** da classe Cromossomo para processar a recombinação dos filhos. A seguir, faz chamadas ao método **gerarMatriz**, também da classe Cromossomo, para atualizar a matriz de confusão dos novos cromossomos.

A seguir o método **evoluir** determina quantos cromossomos sofrerão mutação, de acordo com a taxa de mutação. Um número de melhores cromossomos, determinado pela taxa de preservação, é poupado da mutação. Para os demais cromossomos, a escolha é aleatória.

Uma observação adicional é pertinente sobre o método **evoluir**, relacionada à característica elitista da técnica de algoritmos genéticos e o conseqüente problema de convergência para uma única solução.

Quando a avaliação de um cromossomo, ou de um pequeno grupo de cromossomos, for consideravelmente maior que a dos demais, a roleta tende a selecionar sempre o(s) mesmo(s) cromossomo(s). O resultado é indesejável, pois a população terá vários cromossomos idênticos, reduzindo o espaço de busca efetivo.

Para minimizar este problema, evitando interferir no critério elitista de seleção do mais apto, o método **evoluir** seleciona sempre pais diferentes para o cruzamento. O par de cromossomos pais é selecionado aleatoriamente, mas se forem cromossomos iguais, um deles é aleatoriamente substituído até que um cromossomo diferente seja selecionado. Assim, favorecemos a diversidade de soluções, ou regras, na população.

Também importante, o método ***ranking*** é utilizado para ordenar o vetor de cromossomos de acordo com sua avaliação. Após a classificação, há sempre uma chamada ao método **calcularFatias**, que determina o tamanho da fatia de cada cromossomo na roleta, ou seja, sua probabilidade de seleção para o cruzamento.

4.6.2 Classe Cromossomo

Um cromossomo está associado a um atributo meta, na classe Atributo, e a uma expressão condicional, na classe ExpressãoCondicional.

A matriz de confusão da regra codificada pelo cromossomo é montada através de um vetor de objetos da classe ValorMeta. Cada objeto de ValorMeta é uma coluna da matriz de confusão. O índice do melhor valor na matriz de confusão, ou seja, do valor previsto para o atributo meta na regra relacionada, é indicado por melhorValor. O cromossomo contêm ainda totalizadores para cada linha da matriz de confusão.

A probabilidade de um cromossomo ser selecionado pela roleta é especificada no atributo fatia.

O suporte mínimo, ou cobertura das regras, é especificado pelo parâmetro suporte_minimo. Utilizamos em alguns testes suporte mínimo de 5%. No entanto, a especificação de suporte mínimo elimina muitas regras interessantes, que refletem exceções. Como a função de avaliação utilizada já penaliza regras com cobertura muito restrita, optamos nos processamentos finais por suporte_minimo=0.

Entre os métodos da classe cromossomo, estão os operadores genéticos de mutação e cruzamento e o método **gerarMatriz**, que cria instâncias da classe ValorMeta para montar a matriz de confusão.

O método ***fitness*** determina se a cobertura da regra é maior ou igual ao suporte mínimo especificado e faz chamadas aos métodos **sensibilidade** e **especificidade** para determinar a precisão do indivíduo.

4.6.3 Classe ValorMeta

Cada coluna da matriz de confusão de um cromossomo é representada por uma instância da classe ValorMeta.

Uma instância da classe ValorMeta inclui um valor do domínio do atributo meta; o número de tuplas que apresentam este valor para o atributo meta e satisfazem a expressão condicional; e o número de tuplas que apresentam este valor para o atributo meta e não satisfazem a expressão condicional.

4.6.4 Classe Atributo

A classe Atributo inclui informações sobre cada um dos atributos da relação minerada, como nome e número de valores distintos.

Cada atributo também contém um vetor de valores, representados pela classe ValorAtributo.

Finalmente, um atributo contém um vetor que indica o ganho de informação de cada um dos outros atributos, para quando este atributo for meta de uma regra.

A criação das instâncias de Atributo é realizada em duas fases, no início da execução do GenMiner. Primeiro todos os objetos são criados, com seus respectivos vetores de ValorAtributo, de acordo com a relação selecionada para mineração.

A seguir, é possível preencher os vetores de ganho de informação dos atributos para cálculo da medida de surpresa das regras, usando o método **calculaGanho**.

Este método calcula o ganho de informação de cada um dos outros atributos da relação, de acordo com a Equação 4.

O método **info** é sobrecarregado (RICARTE, 1998[5]). Na versão sem parâmetro de entrada, é calculado o ganho de informação do atributo meta (Equação 5). Na versão em que o parâmetro de entrada é outro atributo, a Equação 6 é implementada. Ambas as versões são chamadas pelo método calculaGanho.

O método **valorAleatorio** seleciona aleatoriamente e retorna um valor do vetor de ValorAtributo.

---

[5] RICARTE, Ivan Luiz. **Programação Orientada a Objetos:** uma abordagem com Java. DCA, FEEC, Unicamp. Setembro, 1998. Disponível em <http://www.dca.fee.unicamp.br/courses/PooJava/>. Acesso em 04 de Abril de 2001.

4.6.5 Classe ValorAtributo

Cada instância de ValorAtributo contêm o valor e o número de vezes que este ocorre na relação minerada para o atributo em questão.

4.6.6 Classe ExpressãoCondicional

Existem dois tipos de ExpressãoCondicional. Cada expressão pode ser terminal ou não. O tipo da expressão é determinado pelo atributo lógico terminal.

Expressões terminais estão relacionadas à classe Condição, enquanto que expressões não terminais estão relacionadas a duas outras expressões condicionais e um operador lógico da classe OperadorLógico.

A probabilidade de uma expressão não ser terminal é dada por taxa_profundidade.

No construtor da classe, há restrições para o caso de expressões não terminais em que o operador lógico é NOT. Neste caso, a expressão deverá associar-se a apenas uma outra expressão. Além disso, evita-se seqüência de dois operadores NOT, como no caso da expressão:

(Idade<20) AND (NOT (NOT (Sexo=F))

Figura 10 Dupla Negação

Pelas características da classe, os métodos em geral são recursivos.

O método **contém**, utilizado pelo operador de mutação, verifica se a condição ou cada uma das expressões relacionadas contém um determinado atributo.

O método **surpresa** calcula a medida de surpresa da condição relacionada, para expressões terminais, ou faz chamadas recursivas nos outros casos.

Finalmente, o método ***crossover*** implementa a troca de sub-expressões ou condições entre duas expressões condicionais, de acordo com o procedimento descrito em 4.5.2.

4.6.7 Classe OperadorLogico

Esta classe contém os operadores AND, OR e NOT e um método para selecionar aleatoriamente um destes operadores.

O método **trocaNOT** é utilizado quando ocorre dupla negação, como na Figura 10.

O método **operadorNOT** retorna verdadeiro ou falso, dependendo se o operador em questão é NOT ou não.

4.6.8 Classe Condição

Uma condição é associada às classes OperadorRelacional e Atributo.

O construtor da classe seleciona aleatoriamente o atributo, seu valor e o operador relacional da condição.

O método **contém**, chamado pelo método de mesmo nome da classe ExpressãoCondicional, verifica se o atributo relacionado é igual ao passado como parâmetro de entrada.

4.6.9 Classe OperadorRelacional

Os operadores relacionais possíveis são: {"=",">","<","<=",">=","<>"}.

Conforme explicado na seção 4.1.1, é desejável, em casos nos quais a maior parte dos atributos são categóricos, que o operador de igualdade seja utilizado com maior freqüência.

Para tanto a taxa_igual indica a probabilidade do operador gerado ser o de igualdade.

5 RESULTADOS OBTIDOS

O GenMiner foi testado em uma base de informações sobre as bandeiras de diversos países. A base FLAG tem número reduzido de exemplos, o que facilitou a constante avaliação da ferramenta em sua fase de desenvolvimento.

Este capítulo apresenta uma breve descrição da base utilizada, exemplos de regras descobertas pela ferramenta, e comentários sobre os diversos parâmetros disponíveis.

5.1 Base FLAG

FLAG contém detalhes sobre diversos países e suas bandeiras. A base é formada por 194 exemplos, ou tuplas. Os atributos de FLAG são descritos na Tabela 1.

ATRIBUTO	DESCRIÇÃO	NÚMERO VALORES
NOME	Nome do país	194
CONTINENTE	1=America do Norte, 2=America do Sul, 3=Europa, 4=África, 4=Ásia, 6=Oceania	6
ZONA	Quadrante geográfico, baseado no Equador e Greenwich 1-Nordeste; 2-Sudeste;3-Sudoeste;4-Noroeste.	4
AREA	Em 1000 km^2	136
POPULACAO	Em milhões de habitantes	48
LINGUA	1= Inglês; 2= Espanhol; 3= Francês; 4= Alemão; 5 = Eslavo; 6= Outro indo-europeu; 7 – Chinês; 8 = Árabe; 9= Japonês/Turco/Finlandês; 10= Outras	10
RELIGIAO	0= Católica; 1= Outras Cristãs; 2= Muçulmana; 3= Budista; 4= Hindu; 5= Étnica; 6= Marxista; 7= Outras	8
BARRAS	Número de barras verticais na bandeira	5
LISTRAS	Número de listras horizontais na bandeira	12
CORES	Número de cores diferentes na bandeira	8
VERMELHO	1 se a cor está presente; 0 caso contrário	2

AZUL	1 se a cor está presente; 0 caso contrário	2
VERDE	1 se a cor está presente; 0 caso contrário	2
DOURADO	1 se a cor (ou amarelo)está presente; 0 caso contrário	2
BRANCO	1 se a cor está presente; 0 caso contrário	2
PRETO	1 se a cor está presente; 0 caso contrário	2
LARANJA	1 se a cor (ou marrom)está presente; 0 caso contrário	2
TOM	Tom predominante na bandeira; desempate pela cor mais acima, ou mais central, ou mais à esquerda, nesta ordem.	8
CIRCULOS	Número de círculos	4
CRUZES	Número de cruzes (†)	3
CRUZESDIAG	Número de cruzes diagonais (X)	2
DIVISOES	Número de divisões	3
ESTRELAS	Número de símbolos de estrela ou sol	14
LUA	1 se o símbolo de uma lua crescente está presente; 0 caso contrário	2
TRIANGULO	1 se um triângulo está presente; 0 caso contrário	2
OBJETO	1 se um objeto inanimado está presente (ex.: barco); 0 caso contrário	2
SERVIVO	1 se uma imagem de ser vivo (ex.: árvore) está presente; 0 caso contrário	2
TEXTO	1 se qualquer letra presente; 0 caso contrário	2
SUPESQUERDO	Cor do canto superior esquerdo da bandeira. Desempate pela cor mais à direita	7
INFDIREITO	Cor do canto inferior direito da bandeira. Desempate pela cor mais à esquerda.	10

Tabela 1 Atributos de FLAG

Como mencionado, a utilização da base FLAG foi útil à etapa de desenvolvimento do trabalho. O número reduzido de tuplas permitia a execução relativamente rápida (em torno de 04 horas) da ferramenta . Assim, foi possível realizar uma constante avaliação de resultados gerados pelas mais diversas configurações.

No entanto, a falta de um usuário especialista no domínio dificultou a avaliação semântica das regras efetivamente descobertas para a base FLAG.

Apresentamos abaixo, com o objetivo de ilustrar o processo evolutivo percorrido pela ferramenta, os valores para precisão máxima e média alcançadas pelo GenMiner para a base FLAG.

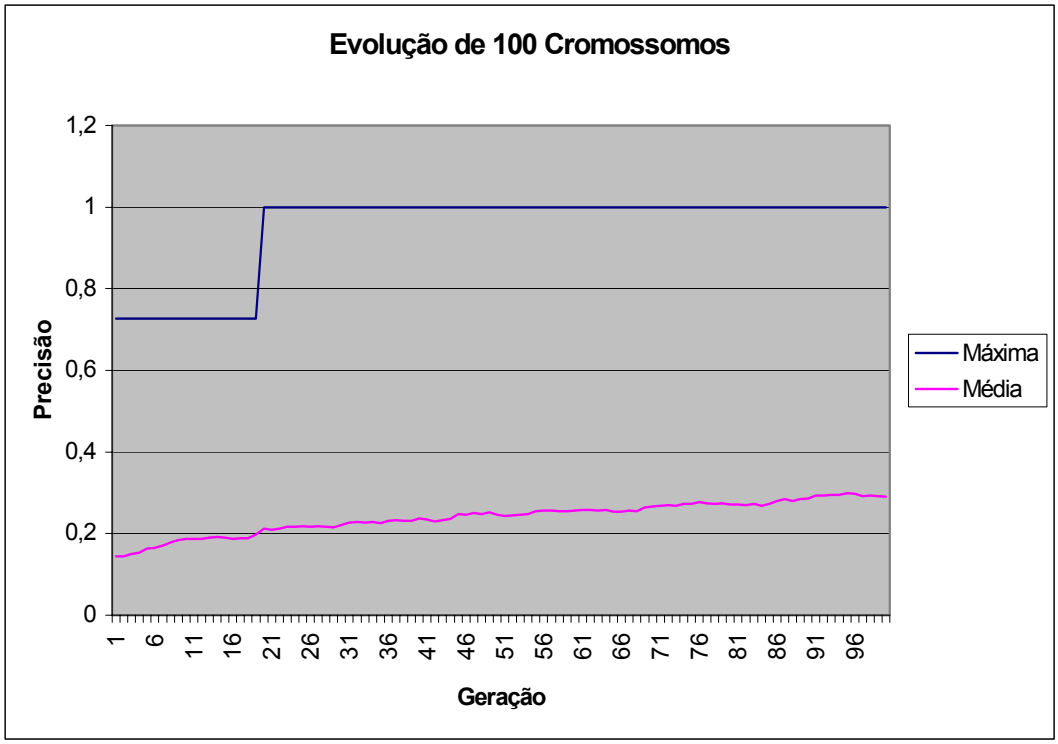


Figura 11 Evolução da Precisão para FLAG

5.2 Regras Descobertas

A Tabela 2 apresenta as 20 melhores regras obtidas pelo processamento de 100 gerações do GenMiner com população de 100 cromossomos.

PARÂMETRO	VALOR
Taxa Mutação	10%
Taxa Preservação	50%
Número Cromossomos	100
Número Gerações	100
Diferença Mínima Precisão	0
Taxa Descarte	20%

	Se	Então	Precisão	Suporte	Surpresa
0	Circulos=4	Area=00047	1	0,005154639	19.990.144
1	InfDireito=laranja	Area=00070	1	0,005154639	0,5015134
2	Populacao=0077	Area=01973	1	0,005154639	0,24154854
3	NOT(NOT(Cruzes=0))	Cruzesdiag=0	0,8333333	0,8505155	42.092.166
4	(((Circulos=1)OR((Texto=0)AND(Divisoes=4)))AND(Tom=laranja))OR(Lingua=04)	SupEsquerdo=laranja	0,7263158	0,015463918	34.534.235
5	Tom=dourado	SupEsquerdo=dourado	0,6134752	0,020618556	11.875.349
6	Cruzesdiag=0	Cruzes=0	0,58549565	0,8505155	4.209.217
7	Cruzes=0	Divisoes=0	0,5750916	0,80927837	59.980.516
8	Cruzesdiag=1	Divisoes=1	0,54674554	0,072164945	5.630.697
9	Cruzesdiag=1	Cruzes=1	0,54674554	0,072164945	4.209.217
10	Cores=2	Area=00001	0,53191495	0,020618556	0,6073028
11	Cruzesdiag=0	Divisoes=0	0,525641	0,8453608	5.630.697
12	Listras=06	Area=00236	0,5	0,005154639	0,6589134
13	InfDireito=marrom	Area=00011	0,49739584	0,005154639	0,5015134
14	Lingua=02	Religiao=0	0,49675325	0,10309278	0,8840855
15	Religiao=7	Area=00021	0,4921875	0,005154639	0,46040103
16	Continente=5	Religiao=2	0,48874828	0,10309278	0,95749164
17	Lingua=06	Continente=3	0,4460018	0,087628864	0,99712235
18	SupEsquerdo=azul	InfDireito=azul	0,44463742	0,12371134	13.973.954
19	Marrom_Laranja=1	Servivo=1	0,4377171	0,0927835	7.466.848
20	Continente=4	Verde=1	0,43123868	0,22164948	57.693.653

Tabela 2 Vinte Melhores Regras Descobertas para a Base de Dados FLAG

5.3 Parâmetros do GenMiner

5.3.1 Taxa de Preservação

A taxa de preservação reforça o aspecto elitista dos algoritmos genéticos. Favorece os melhores cromossomos, impedindo que estes sejam eliminados durante o processo evolutivo.

Um valor baixo para a taxa de preservação prejudica a qualidade dos resultados, permitindo que indivíduos com alta precisão sejam substituídos por indivíduos mal avaliados. Por outro lado, uma taxa de preservação alta prejudica a diversidade dos resultados, reduzindo a margem de cromossomos novos a serem gerados por cruzamento a cada geração.

O gráfico da Figura 12 mostra a precisão média de uma população de 100 cromossomos, durante 100 gerações (ou épocas), para diferentes valores da taxa de preservação. O gráfico destaca a influência que a taxa de preservação exerce sobre os resultados. O valor de 0.5 foi selecionado para a taxa de preservação nas execuções de GenMiner que avaliavam outros parâmetros.

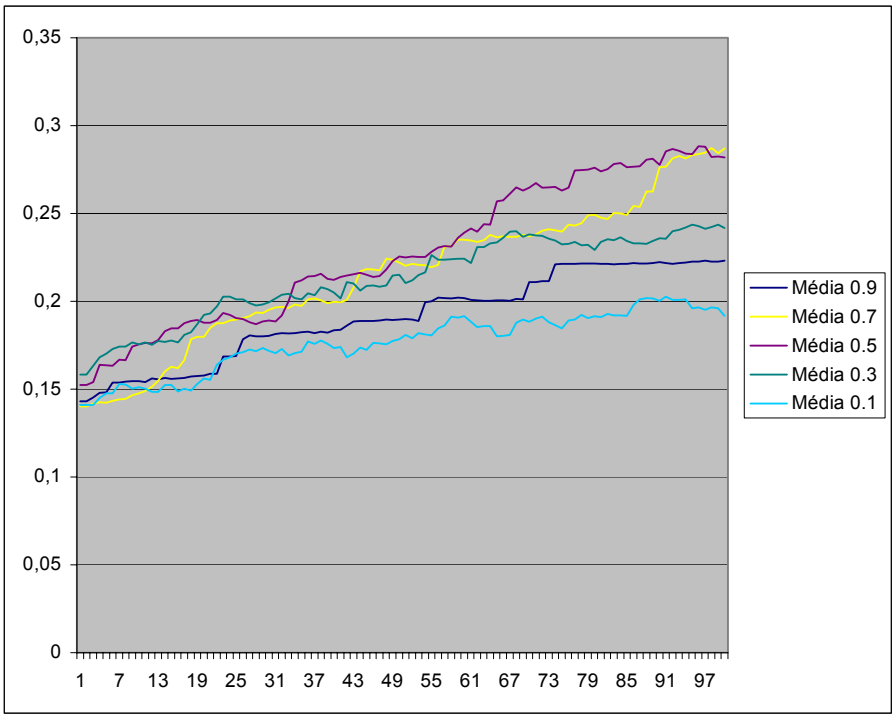


Figura 12 Taxa de Preservação

5.3.2 Taxa de Descarte

O número de cruzamentos entre cromossomos realizados a cada geração é determinado pela taxa de descarte.

Para cada cromossomo descartado, dois cromossomos são selecionados para o cruzamento.

Uma boa taxa de descarte deve ser baixa o suficiente para permitir que os cruzamentos entre bons cromossomos tragam novos bons candidatos para a população. Contudo, a taxa também não pode ser alta a ponto de eliminar bons cromossomos.

O gráfico abaixo mostra execuções com vários valores diferentes para a taxa de descarte, justificando a escolha do valor 0.2 para esta taxa nas demais execuções de GenMiner.

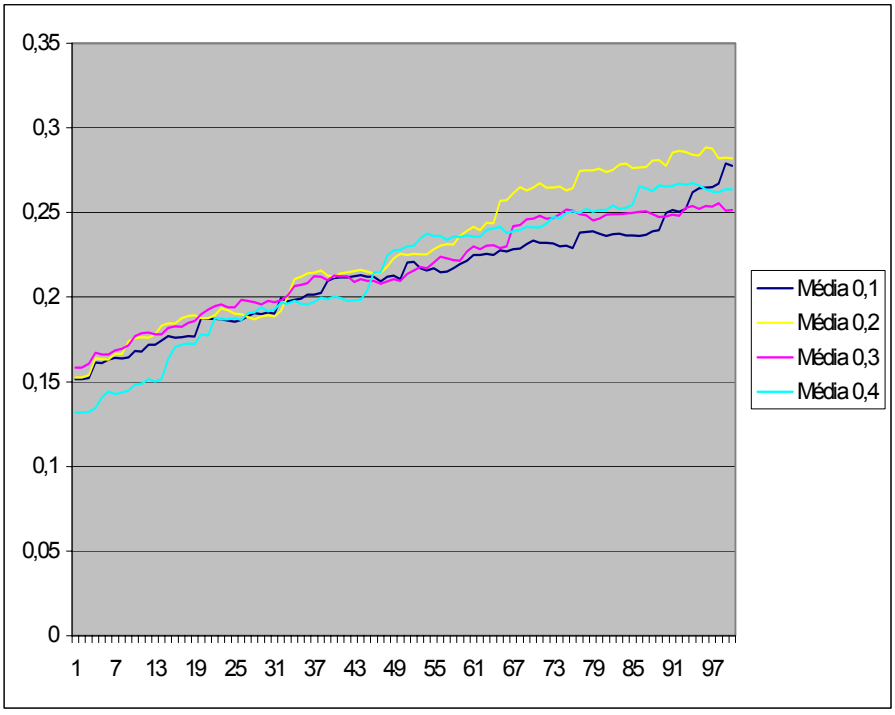


Figura 13 Taxa de Descarte

5.3.3 Taxa de Mutação

A taxa de mutação é um dos fatores de maior influência sobre o processo evolutivo. É através da mutação que todas as regras possíveis têm chance de participar da evolução.

Valores baixos para esta taxa, portanto, causam gerações sem novidades, praticamente limitando a evolução aos resultados dos cruzamentos.

O gráfico abaixo ilustra o efeito do aumento da taxa de mutação na precisão das regras.

Obviamente, mutação em excesso caracteriza um processo aleatório, acabando por prejudicar os resultados.

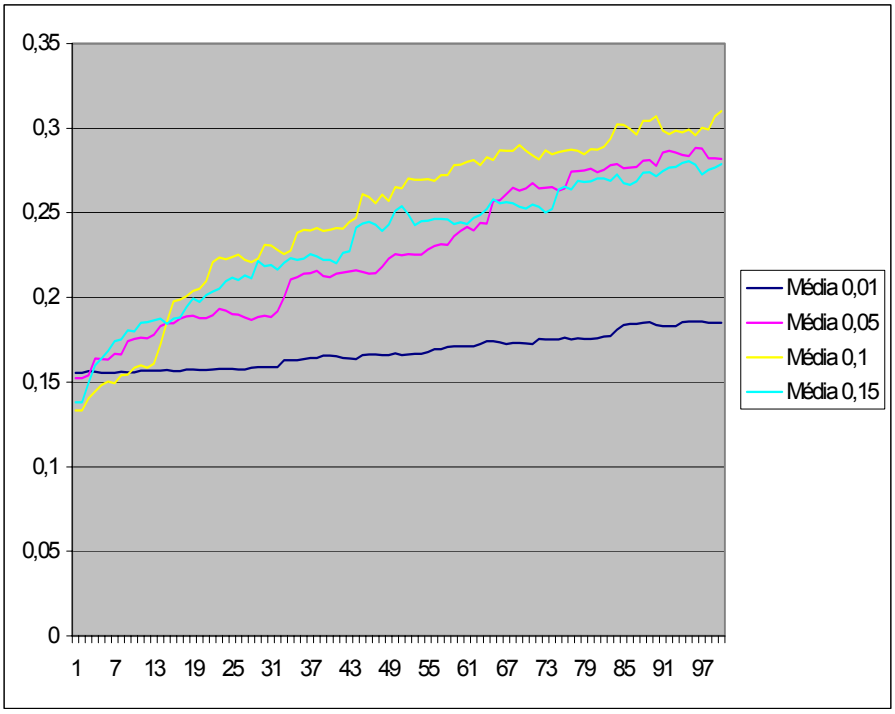


Figura 14 Taxa de Mutação

5.3.4 Surpresa

Na Tabela 2,o valor usado para a diferença mínima de precisão foi zero, ou seja, o desempate por surpresa valia apenas no caso de regras com exatamente a mesma precisão.

Para favorecer ainda mais regras com associações menos previsíveis entre atributos, podemos variar o valor da diferença mínima de precisão. Com a diferença mínima de precisão igual a 10%, obtivemos as regras da Tabela 3.

As regras da Tabela 3 não estão em ordem rigorosa de precisão. Como abordado na seção 4.4.3, quando a diferença entre as medidas de precisão é menor que o mínimo estabelecido, as regras mais surpreendentes têm melhor classificação.

Na Tabela 3, destaca-se a medida de Surpresa das últimas regras. Apesar de não serem regras tão precisas quanto as primeiras colocadas, seriam certamente mais interessantes em um domínio real.

Por exemplo, a regra que relaciona a cor Verde ao Continente Africano tem melhor classificação que uma regra mais previsível relacionando a presença da cor azul ao uso desta cor no topo da bandeira.

Estas são apenas as vinte melhores regras eventualmente obtidas, mas é importante destacar que durante todo o processo de 100 gerações as regras mais surpreendentes foram sucessivamente privilegiadas, para que este resultado final fosse obtido.

PARÂMETRO	VALOR
Taxa Mutação	10%
Taxa Preservação	50%
Número Cromossomos	100
Número Gerações	100
Diferença Mínima Precisão	10%
Taxa Descarte	20%

	Se	Então	Precisão	Suporte	Surpresa
0	Divisoes=4	Area=00076	1.0	0.005154639	2.7509549
1	Circulos=2	Area=01566	1.0	0.005154639	1.9990144
2	Listras=13	Populacao=0231	1.0	0.005154639	1.3827751
3	Cores=1	Area=01760	1.0	0.005154639	0.6073028
4	Cores=1	Area=01760	1.0	0.005154639	0.6073028
5	Religiao=0	Lingua=02	0.8422791	0.10309278	0.8840858
6	Verde=1	Continente=4	0.54739976	0.22164948	5.7693634
7	Cruzesdiag=0	Cruzes=0	0.58549565	0.8505155	4.209217
8	((Divisoes=1) OR (Area=00005)) OR (Cruzesdiag=1)	Cruzes=1	0.5869822	0.082474224	3.7491324
9	Azul=1	SupEsquerdo=azul	0.62913907	0.22164948	2.849161
10	NOT(((Religiao=0)OR(Continente=5))OR (Religiao=2))	Lingua=01	0.60803944	0.21649484	0.9188053
11	Religiao=1	Area=00000	0.60220593	0.13402061	0.46040103
12	Preto=0	Triangulo=0	0.4976713	0.68041235	14.557313
13	NOT((Circulos=1) OR (Cruzesdiag=1))	Cruzes=0	0.5145265	0.7474227	8.01644
14	(Marrom_Laranja=1) AND(Lingua=8)	InfDireito=marrom	0.49739584	0.005154639	3.6269195
15	Cores=2	Area=00001	0.53191495	0.020618556	0.6073028
16	Servivo=0	Cruzesdiag=0	0.3661616	0.7474227	43.88449
17	Marrom_Laranja=0	Cruzesdiag=0	0.34690657	0.80927837	32.252525
18	Branco=1	Barras=0	0.33962265	0.6494845	28.005856
19	Amarelo_Dourado=0	Continente=5	0.3885856	0.14948453	22.330439

Tabela 3 Regras para FLAG usando Desempate por Surpresa

6 CONCLUSÕES

Neste trabalho objetivamos desenvolver uma ferramenta baseada em Algoritmos Genéticos para modelagem de dependência. Entre as tarefas da mineração de dados, modelagem de dependência é a mais beneficiada pelas características não determinísticas de AG, porque o espaço de busca é normalmente amplo demais para uma busca exaustiva.

Uma ressalva é necessária sobre a avaliação da ferramenta. De acordo com a visão de mineração de dados como uma das etapas do processo de KDD, é impossível avaliar adequadamente uma ferramenta de mineração de dados sem que todos os passos do processo tenham sido percorridos. O ideal é que a mineração seja precedida pelo tratamento dos dados, incluindo a seleção dos atributos ideais para a composição das regras. Esperamos, portanto, resultados ainda melhores pela inclusão do GenMiner em um processo completo de KDD em bases de dados reais.

Logo, a avaliação dos resultados obtidos é limitada aos critérios técnicos especificados no AG. Podemos afirmar que regras com alta precisão foram encontradas, através da avaliação por especificidade e sensibilidade. Aplicamos também de forma bem sucedida o critério de desempate, que favoreceu regras potencialmente surpreendentes. A avaliação das regras pelo critério de surpresa merece destaque pela valorização ao grau de interesse da regra, em contraste à tradicional avaliação por cobertura ou confiabilidade.

Como trabalho futuro, a autora pretende incorporar novos critérios à avaliação de regras. Uma idéia seria avaliar regras com baixo suporte com uma função de avaliação diferente (FREITAS, 1998). Regras cobrindo um pequeno número de exemplos podem refletir exceções interessantes, ao passo que regras válidas para a maior parte dos exemplos são geralmente conhecidas pelos usuários. Esta idéia é corroborada pelos resultados obtidos neste trabalho. Inicialmente, o valor especificado para o suporte mínimo das regras era de 0.05. Observamos, no entanto, que este critério eliminava regras potencialmente interessantes. Ao final, as execuções de GenMiner utilizaram suporte mínimo igual a 0.005, ou mesmo 0.

Outra idéia de trabalho futuro é a extensão da ferramenta para a tarefa de classificação, descrita na seção 2.2.2. Como modelagem de dependência é uma generalização de classificação (FREITAS,1997), a extensão tem, a princípio, fácil implementação. Para oferecer suporte a classificação, o GenMiner precisaria permitir a seleção de um atributo meta entre os atributos da relação minerada. No entanto, a avaliação de regras para classificação é mais rigorosa do que para o caso tratado, no sentido de que o conjunto de regras obtido pela ferramenta deve ser capaz de prever qualquer valor do atributo meta. Esta é uma exigência inviável para modelagem de dependência, em razão do grande número de atributos a serem previstos. O objetivo da modelagem de dependência é oferecer ao usuário um conjunto de regras individualmente interessantes.

7 REFERÊNCIAS

AGRAWAL, Rakesh, IMICLINSKI, Tomasz; SWAMI, Arun. Mining Association Rules between Sets of Items in Large Databases. **ACM SIGMOD Conference on Management of Data**, Washington D.C., Maio 1997. 207 – 216. Disponível em : <http://www.almaden.ibm.com/cs/people/ragrawal/papers/sigmod93.ps>. Acesso em: 23/03/2001.

AMO, Sandra de, FERNANDES, Márcia A., SILVA, Flávio R., SOUZA, João N. Mining Temporal Constraints in Databases Using Genetic Programming. **XV Simpósio Brasileiro de Banco de Dados**, João Pessoa, Outubro 2000.172 – 187.

BAYARDO Jr, Roberto, AGRAWAL, Rakesh. Mining the Most Interesting Rules. *ACM SIGKDD International Conference. on Knowledge Discovery and Data Mining*, 5, Agosto, 1999.Disponível em : <http://www.almaden.ibm.com/cs/quest/papers/kdd99.ps>. Acesso em: 24/03/2001

BAYARDO Jr., Roberto, AGRAWAL, Rakesh, GUNOPULOS, Dimitrios. Constraint-Based Rule Mining in Large, Dense Databases. **Proc. of the 15th International Conference on Data Engineering.** Sydney, Australia: Março 1999. Disponível em: <http://www.almaden.ibm.com/cs/people/ragrawal/pubs.html>. Acesso em: 04 de Abril de 2001.

BOJARCZUK, Celia, LOPES, Heitor, FREITAS, Alex. Discovering comprehensible classification rules using genetic programming: a case study in a medical domain. **Genetic and Evolutionary Computation Conference (GECCO-99).** Orlando, FL, USA. Julho 1999. 953-958.

BRACHMAN, Ronald. ANAND,Tej. The Process of Knowledge Discovery in Databases. **In:** FAYYAD, Usama et al. **Advances in Knowledge Discovery and Data Mining**. Cambridge, Massachusetts e Londres: The MIT Press,1996. P. 37-57.

FAYYAD, Usama et al. **Advances in Knowledge Discovery and Data Mining**. Cambridge, Massachusetts e Londres: The MIT Press,1996. 611p.

FIDELIS, Marcos Vinicius, LOPES, Heitor, FREITAS, Alex. Discovering Comprehensible Classification Rules with a Genetic Algorithm. Classification Rules with a Genetic Algorithm. **Congress on Evolutionary Computation.** La Jolla, CA, USA, Julho 2000. 805-810.

FREITAS, Alex Alves. A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction. **2nd Annual Conference Genetic Programming 1997**. Morgan Kaufmann, Julho 1997. Disponível em <http://www.ppgia.pucpr.br/~alex/papers.html > Acesso em: 03 de Abril de 2001.

FREITAS, Alex. On objective measures of rule surprisingness. Principles of Data Mining & Knowledge Discovery **PKDD98**. Lecture Notes in Artificial Intelligence N.1510, P.1-9. Nantes, France, Setembro 1998.

FREITAS, Alex. A Genetic Algorithm for Generalized Rule Induction. In: ROY, R. et al. Advances in Soft Computing - Engineering Design and Manufacturing.. **3rd On-Line World Conference on Soft Computing.** Springer-Verlag, 1999. 340-353

FREITAS, A.A. On rule interestingness measures. **Knowledge-Based Systems Journal.** Elsevier Science, 1999. 12 (5-6), 309-315.

HAN, Jiawei, CAI, Yandong, CERCONE, Nick. Knowledge Discovery in Databases: An Attribute Oriented Approach. **Proceedings of the 18th VLDB Conference,** Vancouver, Canada, 1992. Disponível em <http://citeseer.nj.nec.com/han92knowledge.html>. Acesso em : 24 de Março de 2001.

MURPHY, P.M., AHA, D.W. UCI Repository of Machine Learning Databases. Irvine, EUA, 1994. Disponível em <http://www.ics.uci.edu/~mlearn/MLRepository.html> . Acesso em 08 de Fevereiro de 2001.

NODA, Edgar, FREITAS, Alex, LOPES, Heitor. Discovering Interesting Prediction Rules with a Genetic Algorithm. **Congress on Evolutionary Computation**. Washington D.C., EUA, July 1999. 1322-1329

PÔSSAS, B., MEIRA Jr, W., CARVALHO, M., RESENDE, R. Using Quantitative Information for Efficient Association Rule Generation. **XV Simpósio Brasileiro de Banco de Dados**, João Pessoa, Outubro 2000. 361- 374.

SARAWAGI, Sunita, THOMAS, Shiby, AGRAWAL, Rakesh. Integrating Association Rule Mining with Relational Database Systems. **ACM SIGMOD International Conference on Management of Data.** Seattle, Washington, Junho 1998. Disponível em: <http://www.almaden.ibm.com/cs/people/ragrawal/pubs.html> Acesso em: 03 de Abril de 2001.

SILBERSCHATZ, Avi, TUZHILIN, Alexander. What Makes Patterns Interesting in Knowledge Discovery Systems. **IEEE Transactions on Knowledge and Data Engineering.** V.8 N.6. P. 970-974. 1996.

THOMAS, Shiby, SARAWAGI, Sunita. Mining Generalized Association Rules and Sequential Patterns Using SQL Queries. **International. Conference on Knowledge Discovery and Data Mining** (KDD98). N.4. New York City, New York, EUA, Agosto 1998. P. 344—348.

8 REFERÊNCIAS CONSULTADAS

BRIN, Sergey, MOTWANI, Rajeev, ULLMAN, Jeffrey. Dynamic Itemset Counting and Implication Rules for Market Basket Data. **ACM SIGMOD International Conference on Management of Data.** Tuscon, Arizona. Maio, 1996.255-264.

CHEN, Ming-Syan. Data Mining: An Overview from Database Perspective. **IEEE Transactions on Knowledge and Data Engineering.** Taipei, Taiwan. 1996. Disponível em : < http://citeseer.nj.nec.com/chen96data.html> Acesso em : 24 de Março de 2001.

FLOCKHART, Ian, RADCLIFFE, Nicholas. A Genetic Algorithm- Based Approach to Data Mining. **Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.** Oregon,1996. 299-302.

FOGEL, David. **Evolutionary Computation:**Toward a New Philosophy of Machine Intelligence. Piscataway, NJ: IEEE Press, 1998. 2nd Edition. 640p.

FOWLER, Martin. **UML Distilled**: Applying the Standard Object Modeling Language. EUA. Addison Wesley Longman, 1997. 183p.

FREITAS, A., Understanding the Crucial Differences Between Classification and Discovery of Association Rules – A Position Paper. **ACM SIGKDD Explorations**. ACM, 2000. 2(1), 65-69.

SRIKANT, Ramakrishnan, VU, Quoc, AGRAWAL, Rakesh. Mining Association Rules with Item Constraints. **3rd International Conference on Knowledge Discovery in Databases and Data Mining**. Newport Beach, California, Agosto 1997. Disponível em: <http://www.almaden.ibm.com/cs/people/ragrawal/pubs.html>. Acesso em: 24 de Março de 2001.