ANDRÉA DE FÁTIMA CAVALHEIRO

GADBMS – UM ALGORITMO GENÉTICO MINERADOR DE DADOS PARA BASE DE DADOS RELACIONAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof.ª Dr.ª Aurora T. Ramirez Pozo

CURITIBA
2002

UFPR

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática da aluna ***Andréa de Fátima Cavalheiro***, avaliamos o trabalho intitulado ***"GADBMS - Um Algoritmo Genético Minerador de Dados para Base de Dados Relacionais"***, cuja defesa foi realizada no dia 02 de abril de 2002, às nove horas, no anfiteatro A do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação da candidata.

Curitiba, 02 de abril de 2002.

Profª. Dra. Aurora Trindad Ramirez Pozo
DINF/UFPR - Orientadora

Prof. Dr. Alex Alves Freitas
PUC/PR

Prof. Dr. Martin Alejandro Musicante
DINF/UFPR

A meus pais, ***João*** *e* ***Amélia Cavalheiro****,*
A meu esposo ***Francival****, pela paciência, incentivo e compreensão,*
A meu sogro e sogra, pelo carinho durante todos estes anos de estudo.

AGRADECIMENTOS

A Profª. Drª. ***Aurora Ramirez Pozo*** pela amizade, orientação e apoio ao longo do curso realizado.

Ao ***Curso de Pós-Graduação em Informática*** pela oportunidade de realização deste Mestrado.

Aos demais colegas do mestrado pelo convívio cordial e enriquecedor durante o curso.

A todos que de uma maneira ou outra contribuíram para a elaboração deste estudo.

Finalmente, agradeço a meu esposo ***Francival***, pelo apoio, tranquilidade e equilíbrio que me proporciona em todos os momentos de minha vida.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Resumo

O presente trabalho introduz GADBMS, uma ferramenta de Mineração de Dados para a tarefa de classificação que utiliza um algoritmo genético restrito por listas Tabu para efetuar a busca das regras. Algoritmos genéticos têm diversas vantagens, entre elas: poder trabalhar com dados imprecisos, facilidade de ajustar os parâmetros de acordo com o domínio, possibilidade de paralelização e distribuição da carga de processamento. Apesar do exposto, a tarefa de classificação exige a adoção de alguma estratégia adicional sobre os algoritmos genéticos tradicionais. Algumas dessas estratégias, que têm o objetivo comum de melhorar algoritmos genéticos, habilitando-os a trabalhar com problemas complexos do tipo multiobjetivo-multimodal foram estudadas, especificamente: Busca tabu, Sharing e Evolução Cooperativa. Após este estudo, decidiu-se utilizar um algoritmo genético restrito por listas Tabu. Nesta aproximação as regras pertencentes a qualquer uma das classes podem ser encontradas em apenas uma execução. A ferramenta foi testada em quatro bases de dados e comparada a vinte e três outros algoritmos baseados em regras, obtendo resultados promissores.

Abstract

The present work introduces GADBMS, a Data Mining tool for the classification task which uses a genetic algorithms restricted by Tabu lists to search the rules. Genetic algorithms have several advantages: they can work with imprecise data, they are easily adjusted for different domains of data and possibility of paralelization and loading distribution. Despite the mentioned advantages, the classification task requires some additional strategies to be introduced to the traditional model of genetic algorithms. Some these strategies (which have the common goal to improve genetic algorithms) enable them to work with multiobjective-multimodal complex problems. In this work we studied, specifically: Tabu Search, Sharing and Cooperative Coevolution. After such study, we decided to use a genetic algorithm restricted by Tabu lists. In this approach, rules that belong to any class can be found in one single execution. The tool was tested in four datasets and compared to other twenty-three rule based algorithms, obtaining promising results.

1. Introdução

A mineração de dados é uma etapa na Descoberta do Conhecimento em Banco de Dados que visa buscar relações e modelos dentro de um grande volume de dados. Este processo vai muito além da simples consulta a um banco de dados, pois permite aos usuários explorar e inferir informação útil a partir dos dados, descobrindo relacionamentos possivelmente ainda escondidos.

Uma tarefa de mineração de dados está relacionada a um problema a ser resolvido (FRAWLEY; PIATETSKY-SHAPIRO; MATHEUS, 1991). Para cada tipo de conhecimento que se deseja obter, existe um tipo de aplicação que pode se adequar melhor a ele. Uma vez definida a aplicação, deve ser feita a seleção da tarefa a ser realizada e definir qual técnica será utilizada no processo. Na literatura, existem inúmeras tarefas de mineração de dados, como o agrupamento, a regressão linear, a sumarização, a modelagem de dependência e a classificação, e uma grande variedade de técnicas ou métodos que podem ser utilizados, como árvores de decisão, regras de produção, redes neurais, programação genética e algoritmos genéticos.

Algoritmos Genéticos (AGs) têm provado serem úteis em uma variedade de problemas de busca e otimização. Nos últimos anos, algoritmos genéticos têm sido aplicados em problemas de otimização de soluções, aprendizado de máquina, desenvolvimento de estratégias e fórmulas matemáticas, análise de modelos econômicos, problemas de engenharia, simulação de bactérias, sistemas imunológicos, ecossistemas, descoberta de formato e propriedades de moléculas orgânicas (MITCHELL, 1997).

Os algoritmos genéticos são apropriados para problemas complexos, mas algumas melhorias devem ser feitas no algoritmo básico. Muitas abordagens foram propostas com o objetivo comum de melhorar AGs. O primeiro grupo de estudos foca na manutenção da diversidade na população (DE JONG; SPEARS, 1989; ESHELMAN; SHAFFER, 1991; GOLDBERG, 1989; GOLDBERG, 1990; TSUTSUI; FUJIMOTO, 1993; TSUTSUI; FUJIMOTO, 1994). O segundo grupo visa melhorar o desempenho da capacidade de busca de algoritmos genéticos usando hibridização (COSTA, 1995; GLOVER, 1994; GLOVER;

KELLY; LAGUNA, 1995; KITANO, 1990; MALEK;GURUSWAMY, 1989; MANTAWY; ABDEL-MAGID; SELIM, 1999; MUHLENBEIN; GORGES-SCHELEUTER; KRAMER, 1988; MUHLENBEIN, 1992; POWEL, TONG; SKOLNICK, 1989; ULDER; AARTS; BANDELT; LAARHOVEN; PASCH, 1991). O último grupo de estudos foca em problemas de funções de otimização, ou em problemas para encontrar soluções ótimas de Pareto (CANTU-PAZ, 1999; COELHO, 1999; SCHAFFER, 1985; SRINIVAS; DEB, 1993; TAMAKI; KITA; KOBAYASHI, 1996; FONSECA; FLEMING, 1993; HIROYASU; MIKI; WATANABE, 1999; HORN; NAFPLIOTIS, 1993).

Algoritmos Genéticos apresentam vantagens em relação a algumas outras técnicas, como por exemplo, a capacidade de lidar com dados inválidos ou imprecisos (FREITAS; LAVINGTON, 1998), o ajuste fino de parâmetros de acordo com o domínio (MITCHELL, 1997), a possibilidade de paralelização e distribuição da carga de processamento (FREITAS;LAVINGTON, 1998) e a utilização de heurísticas que podem melhorar seu desempenho, fatos estes que motivaram a utilização do mesmo no desenvolvimento do sistema.

Devido à grande variedade de abordagens, que torna difícil a escolha de qual heurística utilizar, foram implementadas três técnicas: Compartilhamento de Recursos ou Sharing (GOLDBERG; RICHARDSON, 1987), Evolução Cooperativa (POTTER; DE JONG, 2000) e a integração de algoritmos genéticos com Busca Tabu (KURAHASHI; TERANO, 2000), as quais permitiram a execução de alguns testes, que serão mostrados no decorrer do trabalho.

1.1. Objetivos

O presente trabalho introduz o GADBMS, uma ferramenta de Mineração de Dados para a tarefa de classificação que utiliza um algoritmo genético restrito por x listas Tabu, onde x representa a quantidade de classes existentes no domínio. Nesta abordagem as regras pertencentes a qualquer uma das classes podem ser encontradas em apenas uma

execução. O valor de aptidão dos indivíduos é calculado diretamente na base de dados, utilizando-se a própria regra como se fosse um comando SQL.

Para desenvolvimento da ferramenta proposta, foram utilizados como base, a abordagem proposta por (KURAHASHI;TERANO, 2000) que trata da integração de algoritmos genéticos e busca Tabu para otimização de funções multimodal e multiobjetivo, e a abordagem utilizada por (LOPES;POZO, 2001), na qual foi desenvolvida uma ferramenta de mineração de dados que utiliza um algoritmo genético restrito por listas Tabu.

1.2. Organização dos Capítulos

Os capítulos deste trabalho estão organizados como segue. No próximo capítulo é apresentada uma revisão do processo de KDD, assim como algumas tarefas e técnicas da etapa de mineração de dados.

No capítulo 3, são introduzidas as informações necessárias para a compreensão do funcionamento de um algoritmo genético. Alguns conceitos como cruzamento, mutação, seleção de indivíduos e a estrutura básica de um algoritmo genético são detalhados.

No quarto capítulo, são descritas três técnicas que têm o objetivo comum de melhorar algoritmos genéticos: o Compartilhamento de Recursos (Sharing), a Evolução Cooperativa e a Busca Tabu, a qual foi utilizada para desenvolvimento do classificador que será mostrado no capítulo 6.

O capítulo 5 explica o problema objetivo utilizado para a execução dos experimentos realizados com as técnicas descritas anteriormente, assim como os resultados obtidos na execução dos experimentos.

O capítulo 6 apresenta o objetivo principal do trabalho, que consiste no desenvolvimento de uma ferramenta de Mineração de Dados para a tarefa de classificação utilizando um algoritmo genético restrito por listas Tabu para efetuar a busca das regras. Os principais módulos da ferramenta e a interface criada para execução dos testes são comentados.

No capítulo 7 são descritas as tabelas utilizadas para execução dos experimentos, os trabalhos utilizados como base para comparação do classificador e os resultados obtidos com a ferramenta.

Finalmente no capítulo 8 são apresentadas as conclusões do trabalho e algumas sugestões para prosseguimento desta pesquisa.

2. Mineração de Dados

Segundo (FAYYAD et al, 1996), o termo "Descoberta do Conhecimento em Banco de Dados" ou KDD (*Knowledge Discovery in Databases*) pode ser definido como um processo não trivial para identificar padrões válidos, novos, potencialmente úteis e compreensíveis em dados. O processo KDD pode ser resumido em três etapas principais: pré-processamento, mineração de dados e pós-processamento, como mostra a figura 1:

Figura 1 – Etapas do KDD

Para cada uma dessas etapas, alguns passos devem ser executados:

- Pré-Processamento
  - Seleção de um conjunto ou uma amostra de dados, os quais serão analisados. Somente dados relevantes devem estar neste conjunto ou amostra de dados;
  - Limpeza dos dados, aonde serão eliminados problemas como ruídos e dados incompletos;
  - Transformação dos dados que consiste em realizar a projeção ou redução dos dados, tentando encontrar principalmente os dados que não variam;
- Mineração dos Dados, aonde é realizada uma busca por padrões escondidos, muitas vezes de forma repetitiva, aplicando algoritmos. O principal objetivo desse passo é transformar os dados de uma maneira que permita a identificação mais fácil de informações importantes;

- Pós-processamento
  - Interpretação e avaliação das informações descobertas. Neste passo, o resultado da mineração é avaliado, visando determinar se algum conhecimento adicional foi descoberto, assim como definir a importância dos fatos gerados.

Assim sendo, mineração de dados é uma etapa na descoberta do conhecimento em banco de dados que visa buscar relações e modelos dentro de um grande volume de dados. Vai muito além da simples consulta a um banco de dados, no sentido de que permite aos usuários explorar e inferir informação útil a partir dos dados, descobrindo relacionamentos possivelmente ainda escondidos.

Conforme mencionado anteriormente, uma tarefa de mineração de dados está relacionada a um problema a ser resolvido (FRAWLEY; PIATETSKY-SHAPIRO; MATHEUS, 1991). Na literatura, existem inúmeras tarefas de mineração de dados, como por exemplo, o agrupamento, a regressão linear e a classificação. A seguir procura-se, descrever de forma resumida algumas dessas tarefas segundo (FAYYAD et. al, 1996), e algumas das principais técnicas de mineração de dados.

2.1. Tarefas de Mineração de Dados

Através da mineração de dados é possível alcançar duas metas primárias: a *previsão,* que utiliza algumas variáveis para prever valores desconhecidos ou futuros, e a *descrição,* que procura por padrões que descrevem os dados e são interpretáveis por seres humanos. Para alcançar estas metas, é necessária a realização de tarefas básicas de mineração de dados, como as que serão descritas a seguir.

2.1.1. Agrupamento (Clustering)

Agrupamento, ou clusterização, é um aprendizado que identifica um conjunto finito de categorias ou agrupamentos para descrever os dados, ou seja, é uma classificação não

supervisionada. As categorias podem ser mutuamente exclusivas, serem categorias hierárquicas, ou ainda, possuírem áreas em comum.

Na tarefa de agrupamento, o objetivo é dividir uma coleção de dados em um número finito de grupos, tais que os exemplos dentro de um mesmo grupo sejam similares, de acordo com alguma métrica. Geralmente, esta métrica é definida de tal forma que as similaridades dentro de um grupo são maximizadas e as similaridades entre grupos são minimizadas.

Exemplos de aplicações de agrupamento incluem descoberta de sub-populações homogêneas para consumidores do mercado e identificação de subcategorias de um espectro de medidas (FAYYAD et. al, 1996).

2.1.2. Regressão Linear

É o aprendizado de uma função que mapeia um item de dado para uma variável com valor real. Os métodos de regressão linear permitem a discriminação dos dados através da combinação linear dos atributos de entrada, o que equivale a determinar retas de separação dos dados.

Como exemplo, um modelo de regressão pode ser utilizado para estimar a probabilidade de um paciente sobreviver dado o resultado de um conjunto de diagnósticos de exames, ou então, predizer a soma da biomassa presente em uma floresta, fornecida por medidas com sensores remotos de microondas.

2.1.3. Sumarização

Esta tarefa envolve métodos para encontrar uma descrição compacta para um subconjunto de dados. Métodos mais sofisticados envolvem a derivação de regras de resumo e descobertas de relações funcionais entre variáveis. As técnicas de sumarização são geralmente aplicadas à analise exploratória de dados e à geração automática de relatórios.

2.1.4. Modelagem de Dependências

Consiste em encontrar um modelo que descreva dependências significativas entre variáveis. O modelo de dependência pode ser de nível estrutural ou quantitativo. No nível estrutural, o modelo está geralmente representado de uma forma gráfica e com variáveis localmente dependentes em relação a outras, enquanto que, no nível quantitativo, o modelo especifica a força das dependências com alguma escala numérica.

2.1.5. Classificação

A classificação é uma função de aprendizado onde um registro é mapeado em uma das diversas classes predefinidas (FAYYAD et. al, 1996; HAND, 1981; WEISS, KULIKOWSKI, 1991; MCLACHLAN, 1992). Em geral, algoritmos de classificação incluem árvores de decisão, algoritmos genéticos ou redes neurais.

Nesta tarefa, os atributos de um conjunto de dados são divididos em dois grupos, onde um dos grupos conterá apenas um atributo, representando a classe, e o outro grupo conterá todos os atributos que serão utilizados para fazer a predição da classe.

O objetivo da tarefa de classificação visa gerar um modelo de classificação a partir de um conjunto de treinamento, de tal forma que esse modelo permita a classificação de exemplos que não foram utilizados para a geração do mesmo.

Após gerar o modelo de classificação, é necessário estimar a precisão do classificador. Esta precisão pode ser medida pela taxa de erro, a qual mede o número de predições incorretas feitas em relação ao conjunto de dados analisado.

Uma técnica utilizada para se calcular a taxa de erro é executar o modelo de classificação que foi gerado no conjunto de treinamento sobre um conjunto de teste que contenha exemplos que não foram utilizadas na fase de geração das regras.

Outra técnica, também bastante utilizada, é a validação cruzada (cross-validation), onde um conjunto de dados é aleatoriamente dividido em k sub-conjuntos de tamanhos aproximadamente iguais, sem exemplos em comum. Para cada um desses K subconjuntos, os registros não presentes naquele sub-conjunto são utilizados para a geração do modelo de classificação. Este modelo é testado, utilizando-se o sub-conjunto correspondente. Desta

maneira são gerados *k* modelos, cada um com sua própria taxa de erro. A taxa de erro final é obtida pela média de todas as k estimativas dos sub-conjuntos.

Um exemplo de aplicação de classificação é a classificação de tendências do mercado financeiro e a identificação automática de objetos de interesse em grandes bases de dados de imagem (FAYYAD et. al, 1996).

2.2. Técnicas de Mineração de Dados

Há uma grande variedade de técnicas ou métodos que podem ser utilizados na mineração de dados. A seguir, são apresentadas algumas das técnicas mais utilizadas na área.

2.2.1. Árvores de decisão

Uma árvore de decisão é uma forma simples de representação que classifica exemplos em um número finito de classes. Numa árvore de decisão, os nodos internos da árvore correspondem aos nomes dos atributos, as ligações (arestas ou ramos) representam valores do atributo e as folhas representam as diferentes classes a que pertencem as entidades. Um objeto é classificado seguindo o caminho da raiz da árvore até a folha, enquanto os seus atributos têm valores que satisfazem as condições associadas com os nodos internos e suas ligações.

2.2.2. Regras de Produção

As regras de produção podem ser geradas a partir de árvores de decisão, traduzindo os conceitos aprendidos por uma árvore de decisão de uma forma mais compreensível, ou seja, as regras são escritas considerando o trajeto da raiz até a folha da árvore.

Alternativamente, regras de produção podem ser descobertas por um algoritmo de indução de regras, independente de árvores de decisão.

2.2.3. Redes Neurais

As redes neurais constituem um método proveniente da área de Inteligência Artificial baseado na simulação do funcionamento do cérebro, através de estruturas de dados computacionais (HAYKIN, 1994). A grande motivação deste método está na nítida superioridade do cérebro quando comparado aos computadores digitais na realização de tarefas que demandem tolerância a falhas, flexibilidade, lógica imprecisa e paralelismo.

As redes neurais utilizam critérios mais complexos e implícitos baseados no aprendizado a partir de exemplos, não havendo uma codificação de programas a fim de representar explicitamente o conhecimento sobre o problema. Através do aprendizado, que pode ser tanto supervisionado quanto não-supervisionado, as redes neurais lêem exemplos fornecidos sobre um problema e criam um modelo de resolução do problema. Esse modelo pode ser constituído por exemplos de elementos que serão reconhecidos ou classificados no futuro, ou exemplos de elementos que a rede seja capaz de generalizar quando situações similares ocorrerem.

2.2.4. Computação Evolucionária

Computação evolucionária envolve métodos adaptativos e independentes de domínio de aplicação que podem ser utilizados para resolver problemas de busca e otimização. Existem duas classes principais destes métodos: Algoritmos Genéticos e Programação Genética.

Algoritmos Genéticos são algoritmos de busca baseados nos mecanismos de seleção natural e na genética. Nesta técnica, os indivíduos da população são utilizados como possíveis soluções do problema. Esses indivíduos são analisados segundo algum critério e novas soluções são criadas através da seleção de indivíduos bem adaptados e aplicação de operadores genéticos.

Programação genética é uma extensão da técnica de algoritmos genéticos (KOZA, 1992; 1994). A principal diferença entre eles é a forma de representação das estruturas que

eles manipulam e o significado da representação. Em programação genética é comum utilizar árvores para representar os programas de uma população.

Apesar da grande variedade de técnicas que podem ser utilizadas no processo de Mineração de Dados, optou-se pela utilização de algoritmos genéticos para efetuar a busca das regras. Como a tarefa de classificação exige estratégias adicionais sobre os algoritmos genéticos tradicionais que permitam trabalhar com problemas complexos tipo multiobjetivo-multimodal, três heurísticas que têm o objetivo comum de melhorar AGs foram estudadas. Nos próximos capítulos será feita uma discussão mais detalhada sobre estes assuntos.

3. Algoritmos Genéticos

O desenvolvimento de simulações computacionais de sistemas genéticos teve início nos anos 50 e 60 através de muitos biólogos, mas foi John Holland que começou a desenvolver as primeiras pesquisas no tema. Em 1975, Holland publicou "*Adaptation in Natural and Artificial Systems*", ponto inicial dos Algoritmos Genéticos (AGs). David E. Goldberg, aluno de Holland, nos anos 80 obteve seu primeiro sucesso em aplicação industrial com AGs. Desde então os AGs são utilizados para solucionar problemas de otimização e aprendizado de máquinas.

Esses algoritmos simulam processos naturais de sobrevivência e reprodução das populações, essenciais em sua evolução. Na natureza, indivíduos de uma mesma população competem entre si, buscando principalmente a sobrevivência, seja através da busca de recursos como alimento, ou visando a reprodução. Os indivíduos mais aptos terão um maior número de descendentes, ao contrário dos indivíduos menos aptos. Os requisitos para a implementação de um AG são:

- Representações das possíveis soluções do problema no formato de um código genético;
- População inicial que contenha diversidade suficiente para permitir ao algoritmo combinar características e produzir novas soluções;
- Existência de um método para medir a qualidade de uma solução potencial;
- Um procedimento de combinação de soluções para gerar novos indivíduos na população;
- Um critério de escolha das soluções que permanecerão na população ou que serão retirados desta;
- Um procedimento para introduzir periodicamente alterações em algumas soluções da população. Desse modo mantém-se a diversidade da população e a possibilidade de se produzir soluções inovadoras para serem avaliadas pelo critério de seleção dos mais aptos.

A idéia básica de funcionamento dos algoritmos genéticos é a de tratar as possíveis soluções do problema como "indivíduos" de uma "população", que irá "evoluir" a cada iteração ou "geração". Para isso é necessário construir um modelo de evolução onde os indivíduos sejam soluções de um problema. A execução do algoritmo pode ser resumida nos seguintes passos:

- Inicialmente escolhe-se uma população inicial, normalmente formada por indivíduos criados aleatoriamente;

- Avalia-se toda a população de indivíduos segundo algum critério, determinado por uma função que avalia a qualidade do indivíduo (função de aptidão ou "fitness");

- Em seguida, através do operador de "seleção", escolhem-se os indivíduos de melhor valor (dado pela função de aptidão) como base para a criação de um novo conjunto de possíveis soluções, chamado de nova "geração";

- Esta nova geração é obtida aplicando-se sobre os indivíduos selecionados operações que misturem suas características (chamadas "genes"), através dos operadores de "cruzamento" ("crossover") e "mutação";

- Estes passos são repetidos até que uma solução aceitável seja encontrada, até que o número predeterminado de passos seja atingido ou até que o algoritmo não consiga mais melhorar a solução já encontrada.

Os principais componentes mostrados na figura 2 são descritos a seguir em mais detalhes.

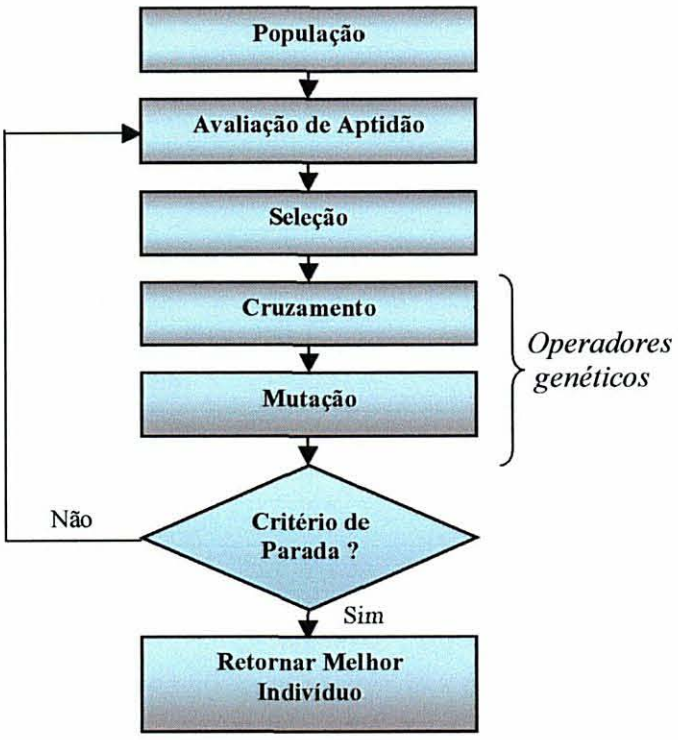

Figura 2 - Estrutura básica de um Algoritmo Genético

3.1. População

A população de um algoritmo genético é o conjunto de indivíduos que estão sendo cogitados como solução e que serão usados para criar o novo conjunto de indivíduos para análise. O tamanho da população pode afetar o desempenho global e a eficiência dos algoritmos genéticos. Populações muito pequenas têm grandes chances de perder a diversidade necessária para convergir a uma boa solução, pois fornecem uma pequena cobertura do espaço de busca do problema. Entretanto, se a população tiver muitos indivíduos, o algoritmo poderá perder grande parte de sua eficiência pela demora em avaliar a função de aptidão de todo o conjunto a cada iteração, alem de ser necessário trabalhar com maiores recursos computacionais.

3.1.1. Indivíduos

O ponto de partida para a utilização de um algoritmo genético como ferramenta para solução de problemas é a representação destes problemas de maneira que os algoritmos genéticos possam trabalhar adequadamente sobre eles. Uma das principais formas é representar cada atributo como uma sequência de bits e o indivíduo como a concatenação das sequências de bits de todos os seus atributos. Outras variações de codificações binárias podem ser encontradas em (HOLLAND, 1975; CARUANA; SCHAFFER, 1988).

A codificação usando o próprio alfabeto do atributo que se quer representar (letras, códigos, números reais, etc.) para representar um indivíduo também é muito utilizada. Alguns exemplos podem ser encontrados em (MEYER; PACKARD, 1992; KITANO, 1994).

Diversas outras formas são possíveis, normalmente a forma mais apropriada está fortemente ligada ao tipo de problema.

3.2. Avaliação de Aptidão (Fitness)

Na avaliação de aptidão, será calculado, através de uma determinada função, o valor de aptidão de cada indivíduo da população. Este é o componente mais importante de qualquer algoritmo genético. É através desta função que se mede quão próximo um indivíduo está da solução desejada ou quão boa é esta solução.

É essencial que esta função seja muito representativa e diferencie na proporção correta as más soluções das boas. Se houver pouca precisão na avaliação, uma ótima solução pode ser posta de lado durante a execução do algoritmo, além de gastar mais tempo explorando soluções pouco promissoras.

3.3. Seleção

Dada uma população em que a cada indivíduo foi atribuído um valor de aptidão, existe vários métodos para selecionar os indivíduos sobre os quais serão aplicados os operadores genéticos. Há diversas formas de seleção, entre eles há o método de seleção por Roleta e o método de seleção por Torneio.

No método de seleção por Roleta (figura 3), cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, para indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos indivíduos de aptidão mais baixa, é dada uma porção relativamente menor.

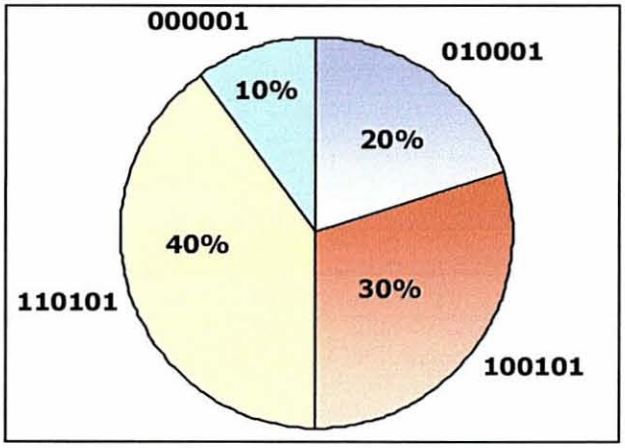

Figura 3 – Método de seleção por Roleta

Neste método, um dos problemas encontrados pode ser o tempo de processamento, já que o método exige duas passagens por todos os indivíduos da população.

Um exemplo da implementação deste método, segundo (MITCHELL, 1997) é mostrado a seguir na figura 4:

```
Inicio
        T = soma dos valores de aptidão de todos os indivíduos da população

        Repita N vezes para selecionar n indivíduos
                r =  valor aleatório entre 0 e T
                Percorra sequencialmente os indivíduos da população, acumulando
                em S o valor de aptidão dos indivíduos já percorridos
                Se  S >= r então
                        Selecione o indivíduo corrente
                Fim se

        Fim Repita
Fim
```

Figura 4 – Algoritmo básico do método de seleção por Roleta

Um outro método é a seleção por Torneio, onde um número *n* de indivíduos da população é escolhido aleatoriamente para formar uma sub-população temporária. Deste grupo, o indivíduo selecionado dependerá de uma probabilidade *k* definida previamente. Um exemplo básico da implementação deste algoritmo (MITCHELL 1997) é mostrado na figura 5, onde n=2:

```
Inicio
        k = 0.75
        Repita N vezes
                Escolha 2 indivíduos da população aleatoriamente
                r =  valor aleatório entre 0 e 1
                Se  r < k
                        O melhor indivíduo é escolhido
                Senão
                        O pior indivíduo é escolhido
                Fim se
        Fim Repita
Fim
```

Figura 5 – Algoritmo básico do método de Seleção por Torneio

Este método é o mais utilizado, pois oferece a vantagem de não exigir que a comparação seja feita entre todos os indivíduos da população (BANZHAF, 1998).

3.4. Operadores Genéticos

O principio básico dos operadores genéticos é transformar a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório. Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores. Os operadores de cruzamento e de mutação têm um papel fundamental em um algoritmo genético.

3.4.1. Cruzamento (Crossover)

Este operador é considerado o operador genético predominante. Através do cruzamento são criados novos indivíduos misturando características de dois indivíduos "pais". Esta mistura é feita tentando imitar (em um alto nível de abstração) a reprodução de genes em células. Trechos das características de um indivíduo são trocados pelo trecho equivalente do outro. O resultado desta operação é um indivíduo que potencialmente combine as melhores características dos indivíduos usados como base.

Alguns tipos de cruzamento bastante utilizados são o cruzamento em um ponto e o cruzamento em dois pontos, mostrados nas Figuras 6 e 7:

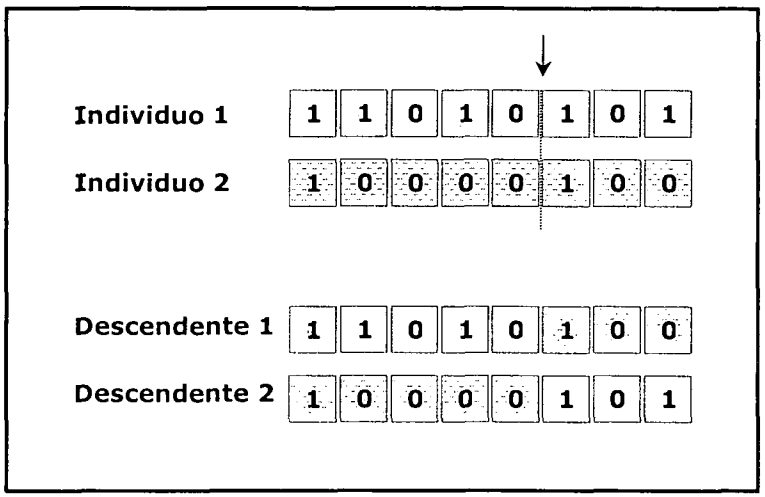

Figura 6 – Cruzamento em um ponto

Com um ponto de cruzamento, seleciona-se aleatoriamente um ponto de corte do cromossomo. Cada um dos dois descendentes recebe informação genética de cada um dos pais (Figura 6).

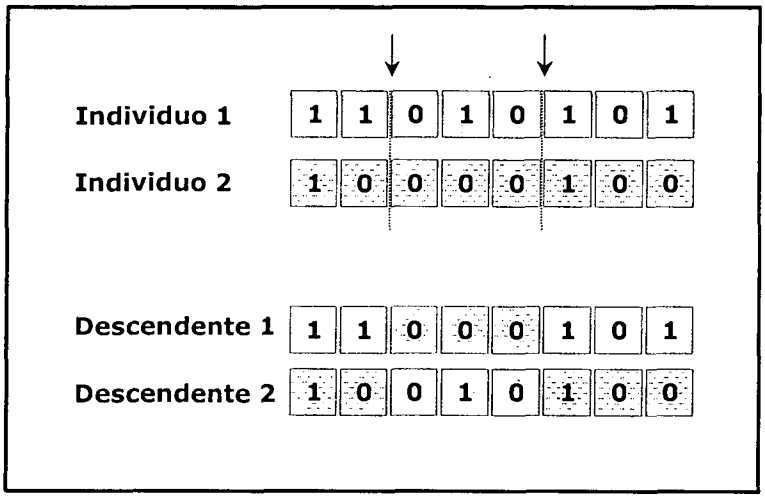

Figura 7 – Cruzamento em dois pontos

Com dois pontos de cruzamento, um dos descendentes fica com a parte central de um dos pais e as partes extremas do outro pai e vice versa (Figura 7).

3.4.2. Mutação

Esta operação simplesmente modifica aleatoriamente alguma característica do indivíduo sobre o qual é aplicada (ver Figura 8). Esta troca é importante, pois acaba por criar novos valores de características que não existiam ou apareciam em pequena quantidade na população em análise. O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população. Desta forma, a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca possivelmente não será zero. O operador de mutação é aplicado aos indivíduos através de uma taxa de mutação geralmente pequena.

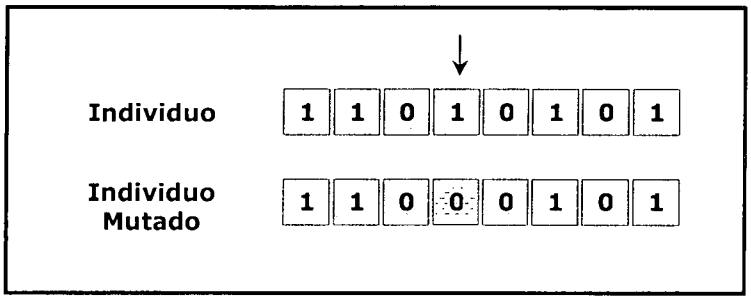

Figura 8 – Mutação Simples

3.5. Geração

A cada passo, um novo conjunto de indivíduos é gerado a partir da população anterior. A este novo conjunto dá-se o nome de "Geração". É através da criação de uma grande quantidade de gerações que é possível obter resultados dos Algoritmos Genéticos.

3.6. Considerações finais sobre AGs

Os algoritmos genéticos são apropriados para problemas complexos, mas algumas melhorias devem ser feitas no algoritmo básico. Muitas aproximações foram propostas com o objetivo comum de melhorar AGs. O primeiro grupo de estudos foca na manutenção da diversidade na população (DE JONG; SPEARS, 1989; ESHELMAN; SHAFFER, 1991; GOLDBERG, 1989; GOLDBERG, 1990; TSUTSUI; FUJIMOTO, 1993; TSUTSUI; FUJIMOTO, 1994) e inclui: métodos de compartilhamento de recursos que utilizam algumas funções sharing para evitar a convergência de indivíduos semelhantes, métodos crowding que obrigam a substituição de indivíduos novos, restrições de cruzamento, etc.

O segundo grupo visa melhorar o desempenho da capacidade de busca de algoritmos genéticos usando hibridização (COSTA, 1995; GLOVER, 1994; GLOVER; KELLY; LAGUNA, 1995; KITANO, 1990; MALEK;GURUSWAMY, 1989; MANTAWY; ABDEL-MAGID; SELIM, 1999; MUHLENBEIN; GORGES-SCHELEUTER; KRAMER, 1988; MUHLENBEIN, 1992; POWEL, TONG; SKOLNICK, 1989; ULDER; AARTS; BANDELT; LAARHOVEN; PASCH, 1991). Nesta

abordagem algoritmos genéticos são usados com um dos seguintes paradigmas: busca tabu, redes neurais artificiais, simulated annealing, etc. Entretanto, a maioria dos estudos na literatura têm focado na busca global através de AGs, enquanto a busca local tem sido feita por outros métodos.

O último grupo de estudos foca em problemas de funções de otimização, ou em problemas para encontrar soluções ótimas de Pareto (CANTU-PAZ, 1999; COELHO, 1999; SCHAFFER, 1985; SRINIVAS; DEB, 1993; TAMAKI; KITA; KOBAYASHI, 1996; FONSECA; FLEMING, 1993; HIROYASU; MIKI; WATANABE, 1999; HORN; NAFPLIOTIS, 1993). Estes estudos incluem: métodos para dividir indivíduos em subgrupos, cada um representando uma função objetivo, combinação de torneio e métodos de compartilhamento de recursos, métodos para dividir soluções de Pareto em algumas áreas, entre outros.

Devido a grande variedade de abordagens, que torna difícil a escolha de qual utilizar, foram implementadas três dessas técnicas: Compartilhamento de Recursos ou Sharing (GOLDBERG; RICHARDSON, 1987), Evolução Cooperativa (POTTER; DE JONG, 2000) e a integração de algoritmos genéticos com Busca Tabu (KURAHASHI; TERANO, 2000), as quais serão detalhadas no próximo capítulo.

4. Técnicas para Manter Diversidade Populacional em Algoritmos Genéticos

Um dos grandes problemas em algoritmos genéticos é o problema de convergência prematura, onde os genes de alguns indivíduos relativamente bem adaptados, contudo não ótimos, podem rapidamente dominar a população causando que o algoritmo convirja a um máximo local.

Para tentar escapar deste problema algumas técnicas podem ser utilizadas em conjunto com algoritmos genéticos, como é o caso do Compartilhamento de Recursos (Sharing), da Evolução Cooperativa e da Busca Tabu, descritos a seguir.

4.1. Compartilhamento de Recursos (Sharing)

A analogia da natureza é que dentro de um ambiente existem diferentes nichos que podem suportar diferentes tipos de vidas (espécies ou organismos). O número de organismos contidos dentro de um nicho é determinado pela fertilidade do nicho e pela eficiência de cada organismo para explorar essa fertilidade.

CAVICCHIO (1970), fez um dos primeiros estudos para tentar induzir nicho como comportamento em algoritmos genéticos. Ele introduziu um mecanismo, o qual chamou *preselection*. Nesse esquema, um descendente substitui o indivíduo pai se o valor de aptidão dele for maior que o valor de aptidão do pai. Desta maneira, a diversidade é mantida na população porque indivíduos tendem a substituir indivíduos similares a eles mesmos.

Outro mecanismo denominado *crowding*, foi proposto por DE JONG (1975) para manter diversidade na população. Neste esquema, a substituição de indivíduos na população é modificada para fazer com que novos indivíduos substituam outros similares com menor valor para a função de aptidão dentro da população.

GOLDBERG e RICHARDSON (1987) introduziram um mecanismo de compartilhamento de recursos, conhecido como Sharing. Neste mecanismo, o objetivo é reduzir o valor de aptidão de indivíduos que têm membros altamente similares dentro da população. Um esquema prático que diretamente usa sharing para induzir nicho e espécie é mostrada na figura 9. Neste esquema, uma função sharing é definida para determinar a vizinhança e o grau de compartilhamento para cada indivíduo da população.

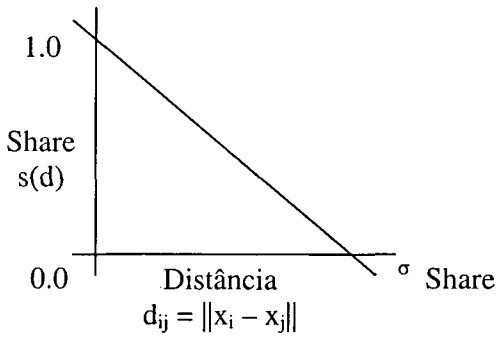

Figura 9 – Função Sharing Triangular (GOLDBERG; RICHARDSON, 1987)

Para um determinado indivíduo o grau de compartilhamento é determinado somando os valores de função sharing contribuídos por todos os outros indivíduos na população. Indivíduos muito similares a outros indivíduos requerem um grau muito alto de compartilhamento, próximo a 1.0, e indivíduos menos similares requerem um grau muito pequeno de compartilhamento, próximo a 0.0. Se um indivíduo é idêntico a um outro indivíduo, seu grau de compartilhamento será igual a 1.0.

Depois de acumular o número total de compartilhamentos, o indivíduo que está sendo avaliado terá seu valor de aptidão reduzido, através da divisão de seu valor de aptidão pela soma acumulada do total de compartilhamentos.

$$f_s(x_i) = \frac{f(x_i)}{\sum_{j=1}^{I} s(d(x_i, x_j))}$$

Onde $f(x_i)$ é o valor de aptidão do indivíduo que está sendo avaliado, $s(d(x_i, x_j))$ é a soma do limiar de distância entre os dois indivíduos.

Como resultado, este mecanismo limitará o crescimento descontrolado de espécies particulares dentro de uma população. A estrutura básica de um algoritmo genético com sharing pode ser vista a seguir:

```
Sharing()
{

     Iniciar a população aleatoriamente
     Avaliar o valor de aptidão dos indivíduos da população,
     reduzindo o valor de acordo com a quantidade de indivíduos
     identicos ou similares dentro da população

      Enquanto não atingir o número de gerações ou o objetivo do
      problema
      {
            Enquanto não atingir o número de indivíduos da
            população
            {
                  Selecionar indivíduos para reprodução
                  Aplicar operadores genéticos para produzir
                  descendentes
            }

            Substituir a população com os descendentes

            Avaliar o valor de aptidão dos indivíduos da nova
            população, reduzindo o valor de acordo com a quantidade
            de indivíduos idênticos ou similares dentro da
            população

      }
}
```

Figura 10 - Estrutura básica de um algoritmo genético com compartilhamento de recursos

Nesta técnica o algoritmo genético tradicional é modificado no módulo de avaliação de aptidão. Cada indivíduo tem seu valor de aptidão reduzido de acordo com a quantidade de indivíduos idênticos ou similares dentro da população. Com isto, o algoritmo tem uma menor probabilidade de que muitos indivíduos do mesmo nicho sejam selecionados, forçando uma maior diversidade populacional.

4.2. Evolução Cooperativa

Outra abordagem que lida com problemas complexos é a evolução cooperativa, proposta por POTTER e DE JONG (2000). Esta arquitetura modela um ecossistema consistindo de duas ou mais espécies. Nesta técnica, as espécies são geneticamente isoladas, ou seja, indivíduos somente cruzam com outros membros de sua espécie. Restrições de cruzamento são forçadas simplesmente por evoluir as espécies em populações separadas. As espécies interagem entre si dentro de um modelo de domínio compartilhado e têm um relacionamento cooperativo. O modelo básico desta abordagem é mostrado na figura 11:

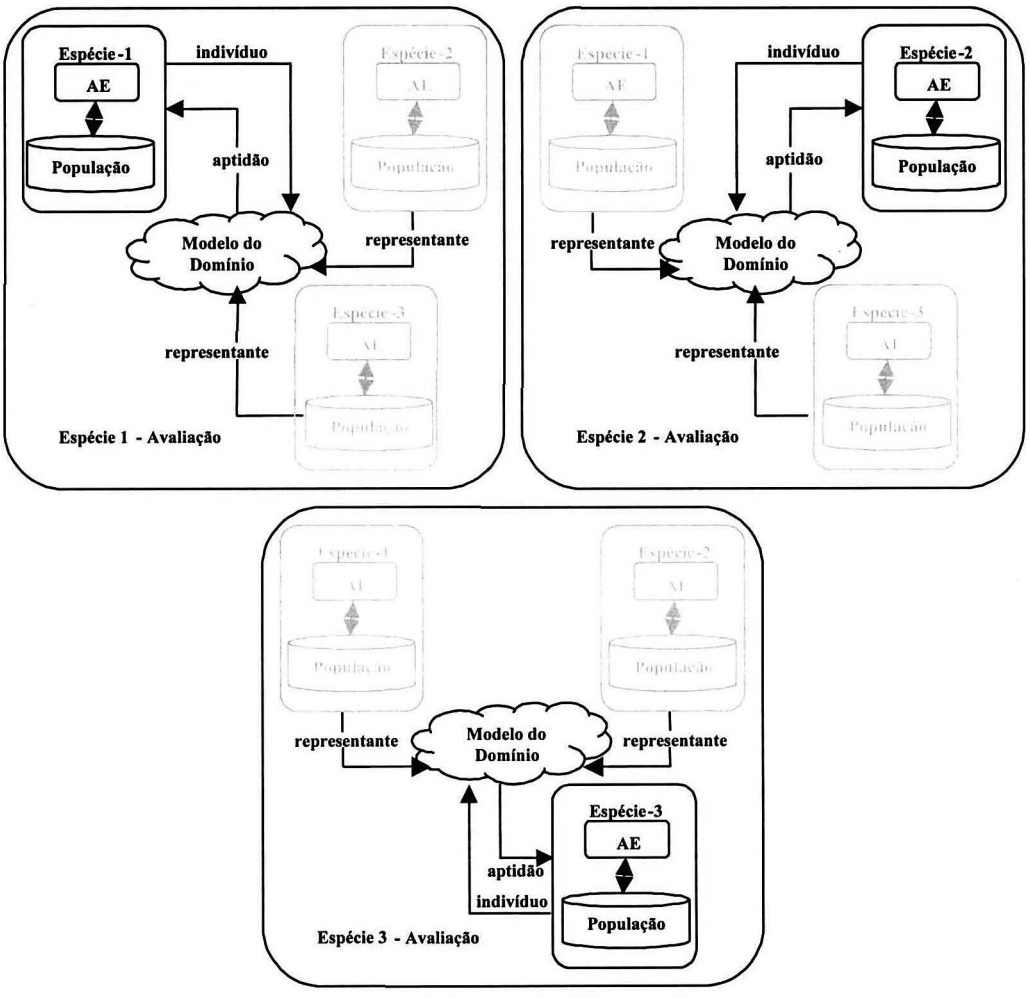

Figura 11 - Modelo de iteração de espécies (POTTER; DE JONG, 2000).

Neste modelo cada espécie é evoluída em sua própria população. A figura mostra a fase de avaliação de aptidão de cada uma das três espécies. Para avaliar uma população são formadas colaborações com representantes de cada espécie.

Há muitos métodos possíveis para escolher os representantes com os quais colaborar. Em alguns casos é apropriado permitir que o melhor indivíduo corrente de cada população seja o representante. Em outros casos, estratégias alternativas são preferidas. Um algoritmo básico desta técnica pode ser visto na figura 12:

```
Inicio
{
      t = 0
      Para cada espécie S
      Inicializar Pt(S) com indivíduos aleatórios

      Para cada espécie S
      Avaliar o valor de aptidão de cada indivíduo em Pt (S)

      Repita Enquanto condição de termino é falsa
      {     Inicio
                  Para cada espécie S
                  Inicio
                        Selecionar indivíduos para reprodução de
                        Pt(S) baseado no  valor de aptidão
                        Aplicar operadores genéticos ao grupo de
                        reprodução para produzir descendentes
                        Avaliar o valor de aptidão dos descendentes
                        Substituir os membros de Pt (S) com os
                        descendentes para produzir Pt+1(S)
                  Fim
                  t = t + 1
            Fim
      }
}
Fim
```

Figura 12 - Algoritmo básico de Evolução Cooperativa (POTTER; DE JONG, 1997).

O algoritmo começa por criar um número fixo de populações. O valor de aptidão de cada membro de cada espécie é então avaliado. Se uma solução satisfatória para o problema objetivo não é encontrada inicialmente, todas as espécies são evoluídas.

Para cada espécie, o algoritmo consiste em selecionar indivíduos para reprodução baseados em seu valor de aptidão, como por exemplo através da seleção de fitness

proporcional; aplicar operadores genéticos como cruzamento e mutação para criação de descendentes; avaliar o valor de aptidão dos descendentes e substituir membros da população velha com novos indivíduos.

O algoritmo de avaliação do valor de aptidão dos indivíduos em uma das espécies é mostrado a seguir:

```
Inicio
{
      Escolher representante de cada uma das outras espécies

      Para cada indivíduo i de S faça avaliação

      Inicio
            Formar colaboração entre i e representantes de outras
            espécies
            Avaliar o valor de aptidão de colaboração através do
            problema objetivo
            Atribuir o valor de aptidão de colaboração a  i
      Fim
}
Fim
```

Figura 13 - Avaliação de aptidão dos indivíduos da espécie S (POTTER; DE JONG, 1997).

Os indivíduos não são avaliados isoladamente, eles são combinados primeiro em algum domínio dependente com um representante de cada uma das outras espécies. POTTER e DE JONG se referem a isto como uma colaboração porque os indivíduos serão julgados no final em quão bem eles trabalham juntos para resolver o problema objetivo.

Se a evolução estagnar, pode ser que existam poucas espécies no ecossistema com o qual construir uma boa solução, então uma nova espécie será criada e sua população aleatoriamente inicializada. A estagnação pode ser descoberta monitorando a qualidade das colaborações pela aplicação da seguinte equação:

$$f(t) - f(t - K) < C,$$

Onde $f(t)$ é o valor de aptidão da melhor colaboração no tempo t , C é uma constante especificando o aumento do valor de aptidão, considerando ter uma melhoria significante, e K é uma constante especificando o tamanho de uma janela evolutiva na qual uma melhoria significante deve ser feita.

Um resumo do algoritmo implementado para esta técnica pode ser visto na figura a seguir:

```
Evolução Cooperativa()
{
      Iniciar cada população com indivíduos aleatórios
      Avaliar o valor de aptidão dos indivíduos de cada população
      Enquanto não atingir o número de gerações ou o objetivo do
      problema
      {     Para cada população Repita Enquanto não atingir o
            número de indivíduos da população
            {
                  Selecionar indivíduos para reprodução
                  Aplicar operadores genéticos p/produzir
                  descendentes
            }

            Avaliar o valor de aptidão dos descendentes de cada
            população
            Substituir a população com os descendentes

            A cada geração
            {
                  Escolher representantes de cada população
                  Avaliar o valor de aptidão de colaboração para o
                  indivíduo i da população X, verificando se i está
                  colaborando para atingir o problema objetivo.
                  Atribuir o valor de aptidão de colaboração
            }
            A cada n gerações
            {
                  Se população X não está contribuindo ou está
                  convergindo para um mesmo padrão
                  {
                     Iniciar população com indivíduos aleatórios
                     Avaliar o valor de aptidão dos indivíduos
                  }
            }
      }
}
```

Figura 14 - Estrutura básica do algoritmo de Evolução Cooperativa

No algoritmo, a cada geração é feita uma cooperação entre as populações, onde é atribuído um valor de aptidão para a população que está sendo avaliada. A cada n gerações, se a população que está sendo avaliada possui um valor de aptidão muito baixo, essa população é descartada, e então criada uma nova população, a qual irá substituí-la nas próximas gerações.

4.3. Busca Tabu

Busca Tabu (*"proibido"*) é um procedimento heurístico proposto por FRED GLOVER para resolver problemas de otimização combinatória. A idéia básica é evitar que a busca por soluções ótimas termine ao encontrar um mínimo local (GLOVER, 1986). Este tipo de algoritmo faz uma busca agressiva no espaço de soluções do problema de otimização com o intuito de obter sempre as melhores alternativas que não sejam consideradas tabu. A heurística Busca Tabu algumas vezes aceita a solução considerado tabu (proibida), baseado no critério de aspiração que determina quando as restrições tabu podem ser ignoradas.

Para implementação do algoritmo Busca Tabu, alguns elementos básicos devem ser especificados, tais como:

- Movimentos: operadores utilizados para transformar uma solução em outra;
- Lista Tabu: onde serão armazenadas todas as soluções anteriores durante o processo de busca do algoritmo. Essa lista é introduzida, no sentido de guardar características dos movimentos realizados, para evitar possíveis retornos a soluções já visitadas;
- Critério de Aspiração: determinará quando uma restrição Tabu deve ser ignorada, realizando o movimento independente se classificado como proibido. Um critério de aspiração comum é ignorar a restrição talvez quando isso produzir uma solução melhor do que todas as soluções geradas anteriormente;

- Término: o processo deve ser finalizado quando não existir mais nenhum movimento possível a ser realizado, ou quando atingir o número máximo de iterações definidas pelo usuário;
- Parâmetros: deve ser informado o tamanho da lista de restrições, número máximo de iterações, regras de parada, solução inicial e critério de aspiração.

Para melhor visualização, a figura 15 mostra um esquema geral do processo de busca Tabu, onde *N(x)* denota o conjunto de soluções vizinhas à *x* no espaço de busca e *T* representa a lista Tabu.

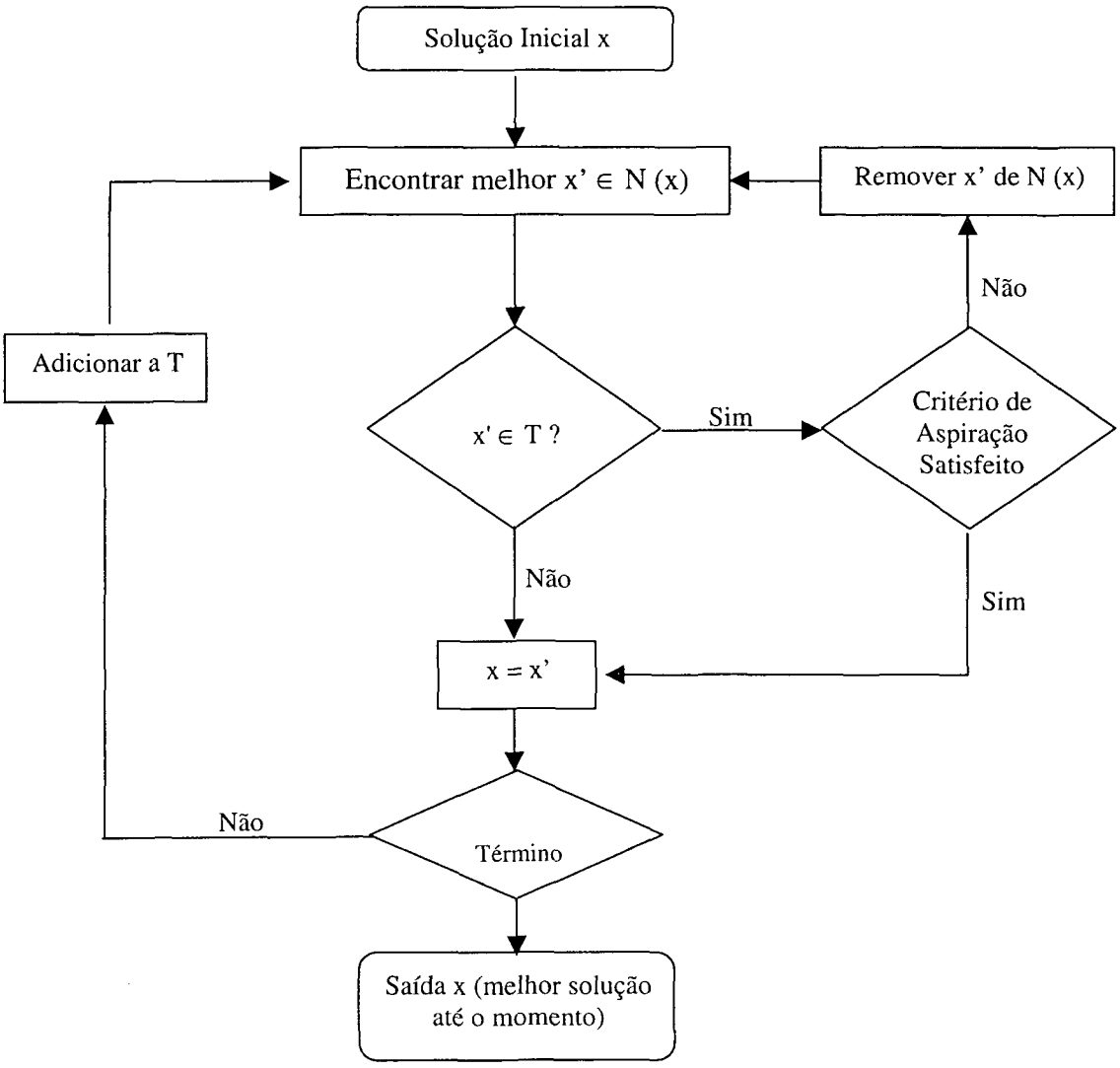

Figura 15 – Método de Busca Tabu (KRISHNAMACHARI, 1999)

O processo inicia selecionando uma solução aleatória *(x)*. Uma busca local é então realizada, procurando todas as soluções vizinhas, *N(x)*. A partir dessas soluções, seleciona-se a melhor solução (*x'*), não sendo necessário que a mesma seja melhor que a solução inicial. A solução inicial é então movida para a melhor solução vizinha adicionando-se a nova solução à lista Tabu. A partir dessa nova solução, realiza-se novamente uma busca local e novamente a melhor solução vizinha é selecionada como candidata para o próximo movimento.

Para evitar que um movimento reverso seja realizado, verificam-se os movimentos das iterações anteriores armazenados na lista de restrições. Se o movimento não se encontra na lista Tabu ou se satisfeito o critério de aspiração, o movimento é aceito, senão testa-se a próxima melhor solução. Esse processo é executado até encontrar uma solução vizinha que não se encontre na lista Tabu. Enquanto não satisfeito um critério de término, o processo é repetido.

KURAHASHI e TERANO (2000) propõem um Algoritmo Genético utilizando-se de múltiplas listas Tabu, auxiliando o algoritmo a alcançar a solução em problemas multimodais ou com mais de uma função objetivo a otimizar.

A maioria dos métodos convencionais utiliza algoritmos genéticos para explorar candidatos globais e algoritmos adicionais para explorar pontos ótimos locais. O trabalho de KURAHASHI e TERANO propõe um novo algoritmo para diretamente armazenar indivíduos dentro de múltiplas listas.

A idéia geral do algoritmo é a utilização de duas listas de restrições: Lista Longa, com tamanho *m* e Lista Curta, com tamanho *n*, onde *m* e *n* são ajustados de acordo com o problema. Estas listas terão o objetivo de armazenar os melhores indivíduos das gerações anteriores, manter o elitismo, manter a diversidade populacional e evitar a convergência a um ponto ótimo local.

A dinâmica dos algoritmos baseia-se na idéia que ao final de cada geração, o melhor indivíduo será armazenado em ambas as listas. Quando se inicia a próxima geração e novos indivíduos são selecionados para a reprodução, as listas de restrições não deixam que indivíduos similares presentes nas mesmas sejam selecionados. Estas listas de

restrições podem ser aplicadas para somente um dos indivíduos escolhidos para gerar os descendentes.

Na Lista Curta serão armazenados apenas os indivíduos das iterações mais recentes. Quando a lista é preenchida completamente, para que um novo indivíduo seja adicionado à mesma, o indivíduo mais antigo deverá ser retirado. Os indivíduos pertencentes a esta lista podem ter o mesmo genótipo.

Na Lista Longa, serão armazenados indivíduos de todas as gerações anteriores. Os indivíduos presentes na lista não poderão ter genótipo idêntico ou similar. Caso surja um indivíduo com um genótipo similar a ser adicionado na Lista Longa, este somente será adicionado, se possuir um valor de aptidão superior e mediante a retirada do outro indivíduo. Este indivíduo retirado da lista sofrerá mutação e será recolocado na população a fim de participar das próximas gerações.

Assim, as soluções serão gradualmente armazenadas na Lista Longa, ou seja, até no máximo m soluções. Ao final do processo, o conjunto solução, será formado pelos indivíduos presentes na Lista Longa.

Testes de otimizações de funções realizados por este algoritmo foram feitos por KURAHASHI e TERANO (2000). Algumas de suas conclusões parciais relatam que, com a adoção desta estratégia, o algoritmo genético utilizando listas de restrições conseguiu cobrir uma área maior do espaço de busca, se comparado a um algoritmo genético puro. Para evitar que as soluções convirjam a um pico em uma função multimodal, eles definem uma medida de distancia entre um indivíduo na lista tabu e um novo candidato. São empregadas três medidas de distância, entre elas a distância de Hamming, onde é calculada a diferença de bits entre dois genótipos (conforme formula a seguir), e que será utilizada na implementação desse algoritmo para comparação com as outras técnicas.

$$d_H(a,b) = \sum_{i=1}^{n} |a_i - b_i|$$

Quando indivíduos gerados por operações de algoritmos genéticos são selecionados via o método de seleção por torneio, apenas um deles é comparado com os indivíduos pertencentes à lista tabu (KURAHASHI; TERANO, 2000). Se a diferença de bits entre o indivíduo selecionado e cada um dos indivíduos da lista está dentro de uma distância pré-

definida, os dois indivíduos selecionados são descartados e uma nova seleção é executada.

A estrutura básica deste algoritmo pode ser vista na Figura 16.

```
AG Restrito()
{     Iniciar população aleatoriamente
      Avaliar o valor de aptidão dos indivíduos da população
      Enquanto não atingir o número de gerações ou o objetivo
      do problema
      {     Selecionar o melhor indivíduo da população

            Retornar indivíduo da lista longa que seja similar ao
            melhor indivíduo da população

            Se limiar de distância entre o melhor indivíduo
            da população e o indivíduo similar da lista < d
                  Inserir indivíduo na Lista Longa
            Senão
                  Verificar se indivíduo selecionado possui valor de
                  aptidão > que indivíduo da lista
                  Se sim
                              Retirar indivíduo da lista
                              Aplicar mutação
                              Colocar indivíduo retirado da lista
                              longa na nova população
                              Colocar indivíduo selecionado na Lista
                              Longa
                  Senão
                              Descartar indivíduo selecionado
            }
            Se Lista Curta não estiver completa
                  Inserir indivíduo na Lista Curta
            Senão
            {
                  Descartar o indivíduo mais antigo
                  Inserir novo indivíduo na Lista Curta
            }
            Enquanto não atingir o número de indivíduos da
            população
            {     Selecionar indivíduos para reprodução e verificar
                  se o primeiro indivíduo selecionado não possui
                  indivíduos idênticos ou similares na Lista Curta
                  ou na Lista Longa
                  Aplicar operadores genéticos para produzir
                  descendentes
            }
            Avaliar o valor de aptidão dos indivíduos da nova
            população
            Substituir a população com os descendentes
}
```

Figura 16 - Estrutura básica do algoritmo genético restrito

No AG Restrito é possível visualizar as modificações que foram efetuadas no algoritmo genético tradicional para implementação desta técnica. Após a população ter sido criada aleatoriamente e o valor de aptidão ter sido calculado, o melhor indivíduo da população é selecionado para fazer parte das listas de restrições. Obedecidos aos critérios de inserção na lista longa e na lista curta, o próximo passo é selecionar indivíduos para reprodução, verificando se o primeiro indivíduo selecionado não possui indivíduos idênticos ou similares nas listas. Caso negativo, aplicam-se operadores genéticos, senão dois novos indivíduos são selecionados. Este processo é repetido até que o número de indivíduos na nova população seja atingido. Por fim, o valor de aptidão dos indivíduos é calculado e a população substituída pelos novos descendentes.

Para uma melhor comparação da eficiência de cada uma das três técnicas mencionadas anteriormente, foram implementadas três ferramentas, as quais tornaram possível a comparação através da aplicação das mesmas no problema objetivo descrito no próximo capítulo.

5. Estudos Preliminares

Neste capítulo, será mostrado o estudo efetuado sobre as três técnicas que têm o objetivo comum de melhorar algoritmos genéticos, assim como os resultados obtidos com os experimentos executados após implementação das ferramentas.

Como problema objetivo, foi utilizado um conjunto composto por 8 strings de 64 bits baseado no problema utilizado por POTTER e DE JONG (2000) mostrado na Figura 17:

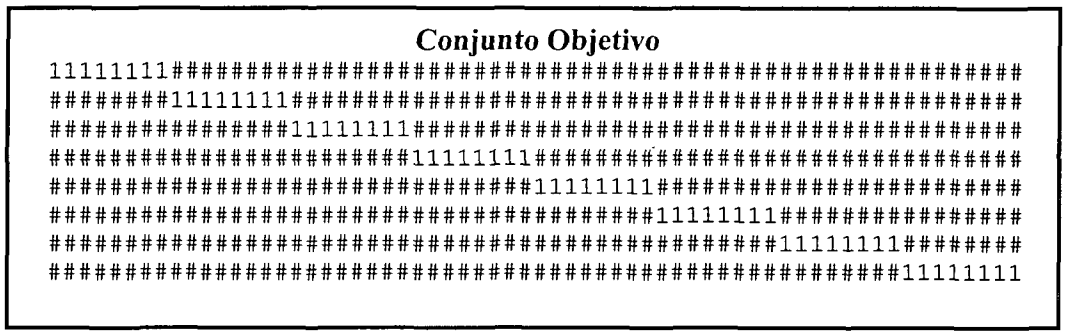

Figura 17 – Conjunto objetivo utilizado nos estudos preliminares (POTTER;DE JONG, 2000)

No conjunto objetivo utilizado, cada string de 64 bits é formada por uma parte binária fixa representada pelo número 1 e uma parte binária variável representada pelo símbolo #. O algoritmo pára a execução após ter coberto todos os padrões do conjunto objetivo ou ao atingir o número máximo de gerações.

A primeira etapa do trabalho foi divida em duas fases. Na primeira fase, os algoritmos foram implementados utilizando verificação de similaridade somente para indivíduos idênticos. Neste caso, se um indivíduo similar fosse encontrado, seria tratado como um indivíduo diferente. Na segunda fase, foram utilizados somente o AG Restrito e o algoritmo Sharing, onde foi aplicada medida de similaridade, tratando indivíduos idênticos e similares.

Função de Aptidão (*Fitness*)

Na função de aptidão utilizada os indivíduos são comparados a cada padrão calculando-se a distância de Hamming. O valor de aptidão do indivíduo será a menor distancia, isto é, o valor associado ao padrão que o indivíduo se assemelha mais.

A seguir será mostrada uma visão geral dos programas implementados e dos experimentos realizados na primeira etapa do trabalho. Na execução dos testes, o objetivo é encontrar indivíduos que cubram cada um dos 8 padrões do conjunto objetivo mostrado na Figura 17.

5.1. Algoritmos Implementados

Para comparação entre as técnicas, foram desenvolvidos três programas em linguagem C++ com interface DOS. Em cada programa é possível acompanhar os resultados de cada execução através de arquivos textos que são gerados em tempo de execução. Através destes arquivos, os dados podem ser exportados para planilhas, o que torna mais fácil a comparação dos resultados através de gráficos. Um exemplo da entrada de parâmetros para execução de um dos programas pode ser visto na figura 18:

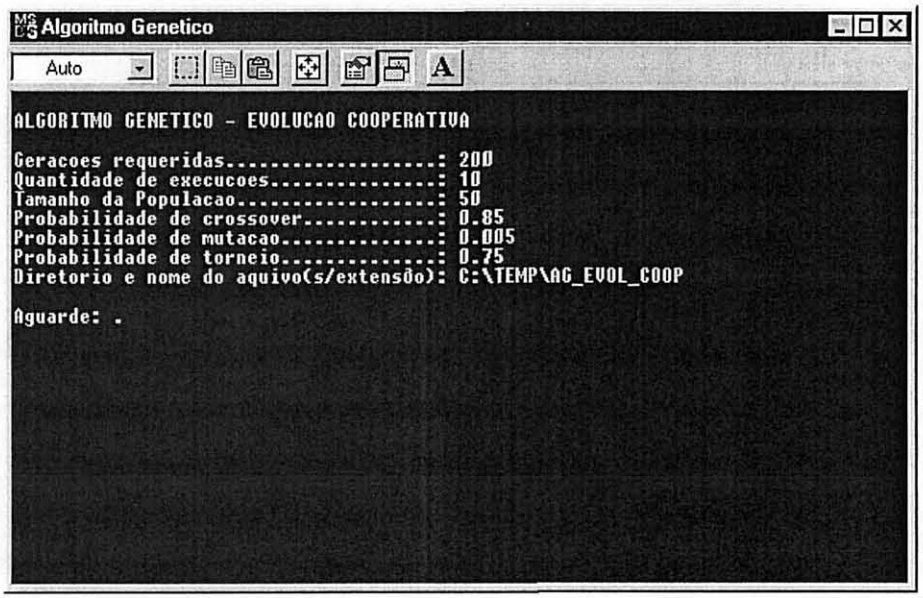

Figura 18 – Interface de entrada do Algoritmo Evolução Cooperativa

Após o término de todas as execuções, um dos arquivos gerados armazenará o tempo que o algoritmo levou para encontrar os 8 padrões, ou para atingir o número máximo de gerações solicitadas:

```
Execução: 1 - Hora Inicial: 19:42:33          Hora Final: 19:42:38
Execução: 2 - Hora Inicial: 19:42:38          Hora Final: 19:42:41
Execução: 3 - Hora Inicial: 19:42:41          Hora Final: 19:42:58
Execução: 4 - Hora Inicial: 19:42:58          Hora Final: 19:43:05
Execução: 5 - Hora Inicial: 19:43:05          Hora Final: 19:43:09
```

Figura 19 – Arquivo com os tempos de execução

Para cada execução, um arquivo é criado contendo todos os padrões encontrados em cada geração. Nesse arquivo é possível visualizar a quantidade de indivíduos encontrados para cada um dos padrões que devem ser atingidos. No exemplo abaixo, são mostradas as três últimas gerações em uma das execuções do algoritmo, quando é encontrado pelo menos um indivíduo que cobre cada um dos padrões que forma o conjunto objetivo.

```
Conjunto Objetivo
11111111000000000000000000000000000000000000000000000000000000000000   N° de Padrões  = 4
00000000111111110000000000000000000000000000000000000000000000000000   N° de Padrões  = 7
00000000000000001111111100000000000000000000000000000000000000000000   N° de Padrões  = 3
00000000000000000000000011111111000000000000000000000000000000000000   N° de Padrões  = 0
00000000000000000000000000000000111111110000000000000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000001111111100000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000000000000011111111000000000000   N° de Padrões  = 0
00000000000000000000000000000000000000000000000000000000111111110000   N° de Padrões  = 5


Conjunto Objetivo
11111111000000000000000000000000000000000000000000000000000000000000   N° de Padrões  = 4
00000000111111110000000000000000000000000000000000000000000000000000   N° de Padrões  = 11
00000000000000001111111100000000000000000000000000000000000000000000   N° de Padrões  = 1
00000000000000000000000011111111000000000000000000000000000000000000   N° de Padrões  = 0
00000000000000000000000000000000111111110000000000000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000001111111100000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000000000000011111111000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000000000000000000000111111110000   N° de Padrões  = 3


Conjunto Objetivo
11111111000000000000000000000000000000000000000000000000000000000000   N° de Padrões  = 10
00000000111111110000000000000000000000000000000000000000000000000000   N° de Padrões  = 6
00000000000000001111111100000000000000000000000000000000000000000000   N° de Padrões  = 1
00000000000000000000000011111111000000000000000000000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000111111110000000000000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000001111111100000000000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000000000000011111111000000000000   N° de Padrões  = 1
00000000000000000000000000000000000000000000000000000000111111110000   N° de Padrões  = 6
```

Figura 20 – Três ultimas gerações de uma das execuções

Outro arquivo criado para cada execução, mostra a quantidade total de padrões encontrados em cada geração. Com estes resultados verifica-se em quantas gerações o algoritmo consegue atingir o conjunto objetivo. Um exemplo deste arquivo pode ser visto na figura 21:

```
Geracoes;Padroes;
1;3;
2;4;
3;5;
4;4;
5;6;
6;6;
7;5;
8;6;
9;7;
```

Figura 21 – Padrões encontrados em cada geração de uma execução

Através destes arquivos, foram criados alguns gráficos para melhor visualização dos resultados obtidos com estas três técnicas, os quais serão mostrados na próxima seção.

5.2. Experimentos Realizados

Em todos os experimentos a população inicial foi criada aleatoriamente. Para seleção dos indivíduos, foi utilizado o método de seleção por torneio, cruzamento em dois pontos e mutação simples. Para cada conjunto de parâmetros, foram efetuados várias execuções com o objetivo de encontrar parâmetros que permitissem encontrar os oito padrões. Após estas execuções, os seguintes parâmetros foram escolhidos:

- Evolução Cooperativa: 8 espécies e população com tamanho 50 para cada espécie;
- Sharing: população com tamanho 300;
- AG Restrito: população e lista longa com tamanho 300 e lista curta com tamanho 30;
- Máximo de 200 gerações para os três métodos.

Na primeira fase do trabalho, verificando a existência de indivíduos idênticos somente, foram utilizados dois conjuntos de parâmetros mostrados na tabela 1, e efetuadas 10 execuções para cada uma das técnicas.

Tabela 1 – Probabilidades utilizadas na primeira fase

	Taxa		
	Cruzamento	**Mutação**	**Torneio**
#1	0.65	0.02	0.65
#2	0.85	0.05	0.75

Na segunda fase do trabalho, usando medida de similaridade, verificando a existência de indivíduos idênticos e similares, foi utilizado apenas o segundo conjunto de parâmetros e efetuadas 10 execuções para o algoritmo Sharing e 10 execuções para o AG Restrito. Nesta fase, devido ao fato do algoritmo de evolução cooperativa não necessitar de medida de similaridade para trabalhar melhor, não foi utilizado para comparações.

Para calcular a semelhança entre dois indivíduos no algoritmo Sharing, foi utilizada a função Sharing Triangular, mostrada na seção 4.1 e para o AG Restrito foi utilizada a medida de distancia Hamming (igual a 6), mostrada na seção 4.3. A seguir serão mostrados os resultados obtidos em cada uma destas fases.

5.2.1. Medida de Similaridade verificando Indivíduos Idênticos

Com o objetivo de comparar as três técnicas descritas anteriormente, os algoritmos Sharing e AG Restrito foram executados verificando somente a existência de indivíduos idênticos na população.

Nesta primeira fase, foi calculada a média entre as 10 execuções e o tempo de processamento de cada algoritmo. Com o primeiro conjunto de parâmetros da tabela 1, chegou-se aos seguintes resultados:

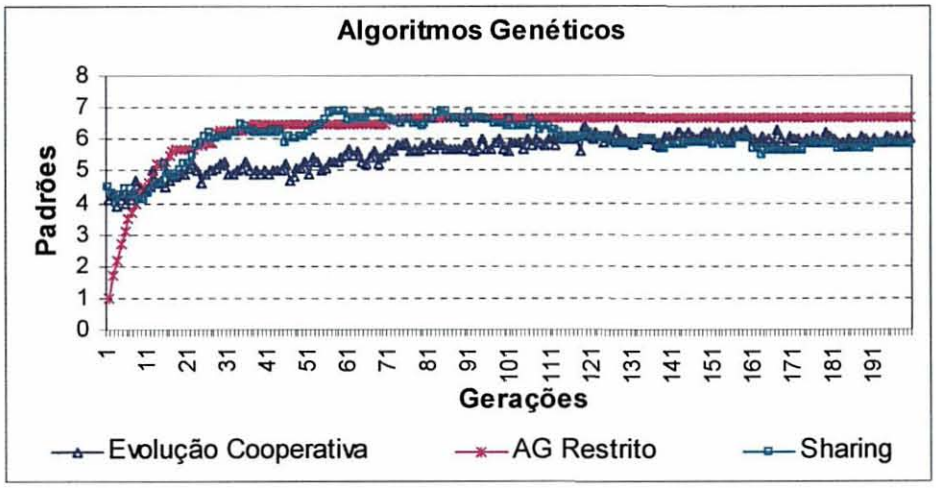

Gráfico 1 - Média de 10 execuções na primeira fase (#1)

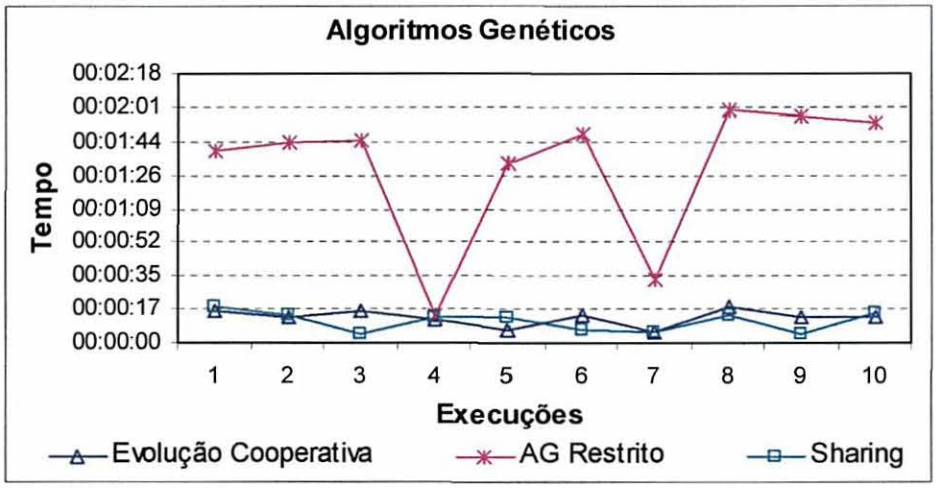

Gráfico 2 – Tempo de processamento em 10 execuções na primeira fase (#1)

Pode-se observar que nenhum dos três algoritmos conseguiu encontrar os oito padrões. Ao invés disto o AG Restrito chegou em sete padrões e permaneceu com esses padrões até o fim das gerações. O algoritmo Sharing também alcançou sete dos oito padrões, mas nas gerações posteriores, acabou perdendo alguns dos padrões encontrados. O algoritmo evolução cooperativa foi aos poucos alcançando seis padrões. Com estes experimentos verifica-se que diferentemente do algoritmo Sharing, quando o AG Restrito

encontra uma boa solução, ele não a perde, pois os melhores padrões permanecem armazenados na lista longa.

Quanto ao tempo de processamento, o qual é mostrado no gráfico 2, pode-se notar que o algoritmo de Evolução Cooperativa e o Algoritmo Sharing têm um tempo de processamento bastante similar em cada execução. O tempo de execução do AG Restrito se mostrou bastante diferente, devido ao fato de que nesse algoritmo a cada geração um dos indivíduos selecionados é comparado com os indivíduos pertencentes a lista longa, o que veio a alternar bastante o tempo de execução.

Ao utilizar os parâmetros do segundo conjunto (#2) da tabela 1, foram obtidos os seguintes resultados:

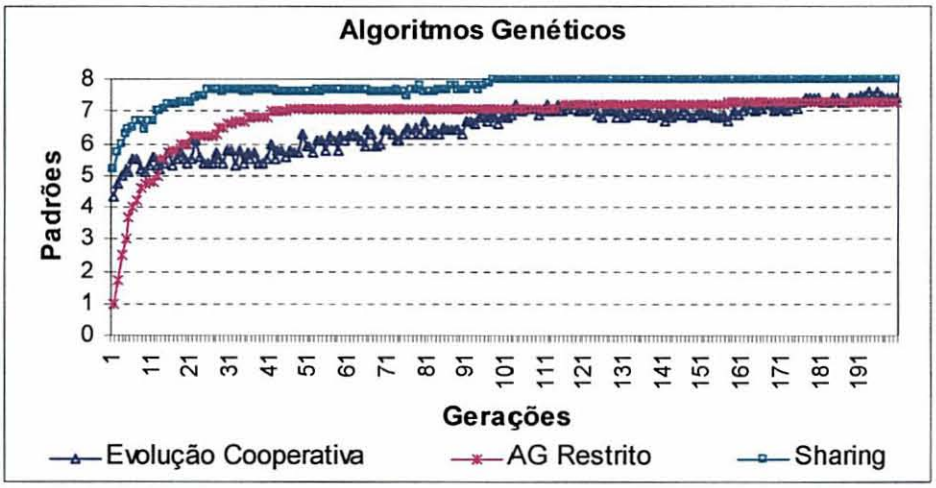

Gráfico 3 - Média de 10 execuções na primeira fase (#2)

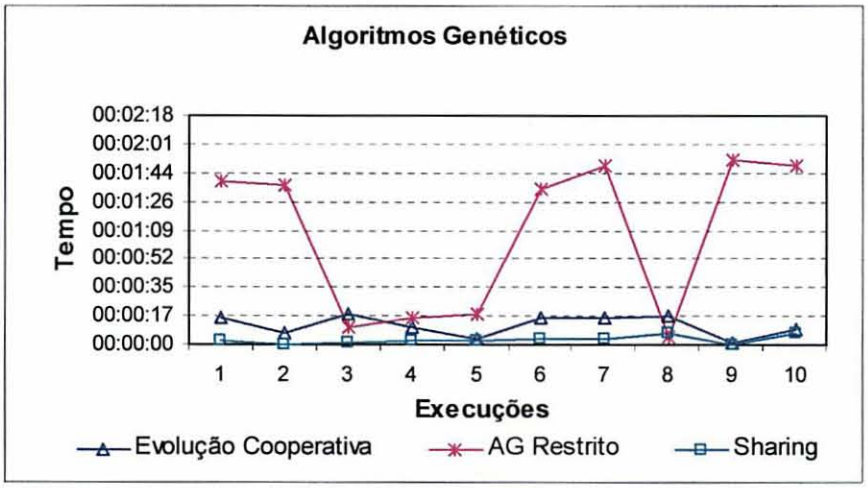

Gráfico 4 – Tempo de processamento em 10 execuções na primeira fase (#2)

É possível observar através das execuções que em média, o algoritmo sharing consegue um melhor comportamento do que os outros dois algoritmos, alcançando os oito padrões. O tempo de processamento de ambos algoritmos mostrou-se bastante similar ao caso anterior.

5.2.2. Medida de Similaridade verificando Indivíduos Idênticos e Similares

Na segunda fase de testes o algoritmo sharing e o AG Restrito foram alterados para trabalhar com medida de similaridade. Para esta fase, foram aplicados os parâmetros do segundo conjunto da tabela 1, obtendo-se os seguintes resultados:

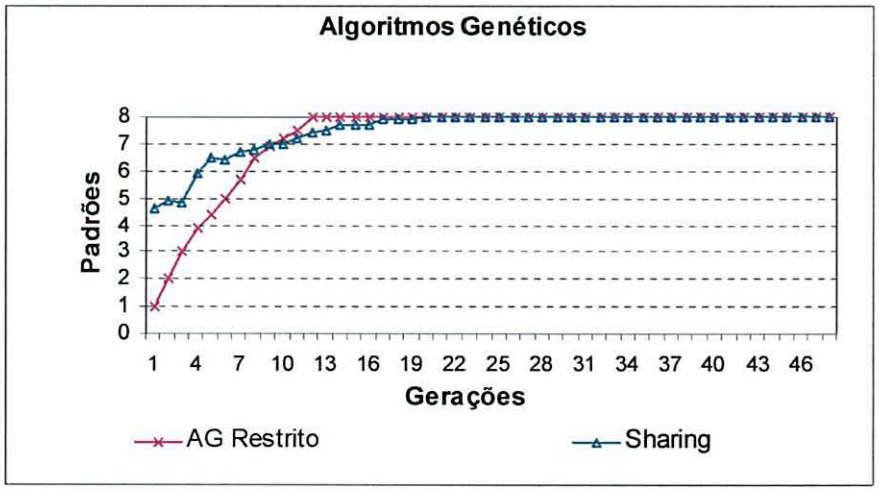

Gráfico 5 - Média de 10 execuções na segunda fase (#2)

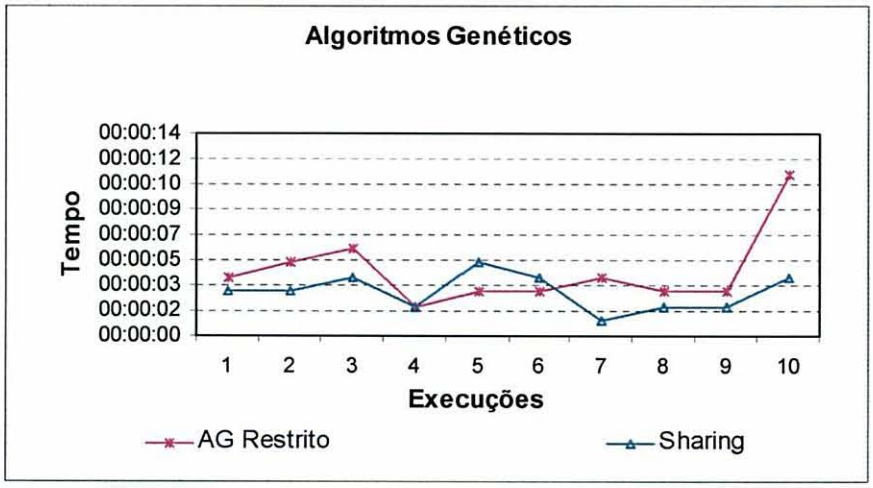

Gráfico 6 – Tempo de processamento em 10 execuções na segunda fase (#2)

Nestes experimentos, pode-se observar que os dois algoritmos tiveram um comportamento muito melhor aos casos anteriores, como por exemplo, na quantidade de gerações que foram necessárias para atingir o conjunto objetivo. Com a aplicação de medida de similaridade verificando a existência de indivíduos idênticos e similares, o AG Restrito conseguiu obter um tempo de processamento bastante semelhante ao algoritmo Sharing. Com estes resultados, conclui-se que a aplicação de medida de similaridade nestas duas técnicas é fundamental para o problema definido no início deste capítulo (Figura 17).

Com os resultados obtidos, foi possível mostrar a eficácia destes algoritmos no problema proposto. Embora a quantidade de testes efetuados não tenha sido grande, pode-se verificar que estas técnicas são úteis de diferentes maneiras. A Evolução Cooperativa é uma abordagem que pode levar vantagem da distribuição de espécies em diferentes computadores, mas isto pode adicionar complexidade na administração, como por exemplo, uma condição de estagnação da evolução. Neste caso, pode ser que existam poucas espécies no ecossistema de qual construir uma boa solução; então uma espécie nova será criada e sua população aleatoriamente inicializada. Esta aproximação quando usada de forma consecutiva não parece trazer grandes vantagens. E por último a necessidade de verificação de similaridade utilizando uma medida de distância para o algoritmo Sharing e o AG Restrito pode vir a ser bastante complexa quando aplicado em diferentes domínios.

Baseado nestes prévios estudos (KURAHASHI; TERANO, 2000; LOPES; POZO, 2001) e nos resultados obtidos (CAVALHEIRO, POZO, 2002), foi implementado um classificador que utiliza um algoritmo genético restrito por listas tabu, o qual será descrito no próximo capítulo.

6. Classificador

Neste capítulo, será apresentado GADBMS, um sistema classificador para a indução de regras baseado em um algoritmo genético restrito por *x* listas tabu. Para criação deste sistema, levou-se como base o trabalho feito por LOPES e POZO (2001), no qual foi implementado um classificador que utiliza um algoritmo genético restrito por listas Tabu para efetuar a busca das regras.

Na abordagem apresentada por LOPES e POZO (2001), são utilizadas uma lista longa e uma lista curta para armazenar as regras. Cada classe é evoluída separadamente. Na abordagem proposta, são utilizadas *x* listas Tabu, onde *x* corresponde à quantidade de classes do problema proposto. Nesta abordagem, todas as classes são evoluídas em uma mesma execução e cada indivíduo tem seu valor de aptidão calculado diretamente na base de dados, utilizando a si mesmo como um comando SQL.

A ferramenta foi implementada em C++ com interface DOS e para os testes foi utilizado banco de dados ORACLE para armazenar todos os conjuntos de dados avaliados. A arquitetura implementada pode ser vista na figura 22 abaixo:

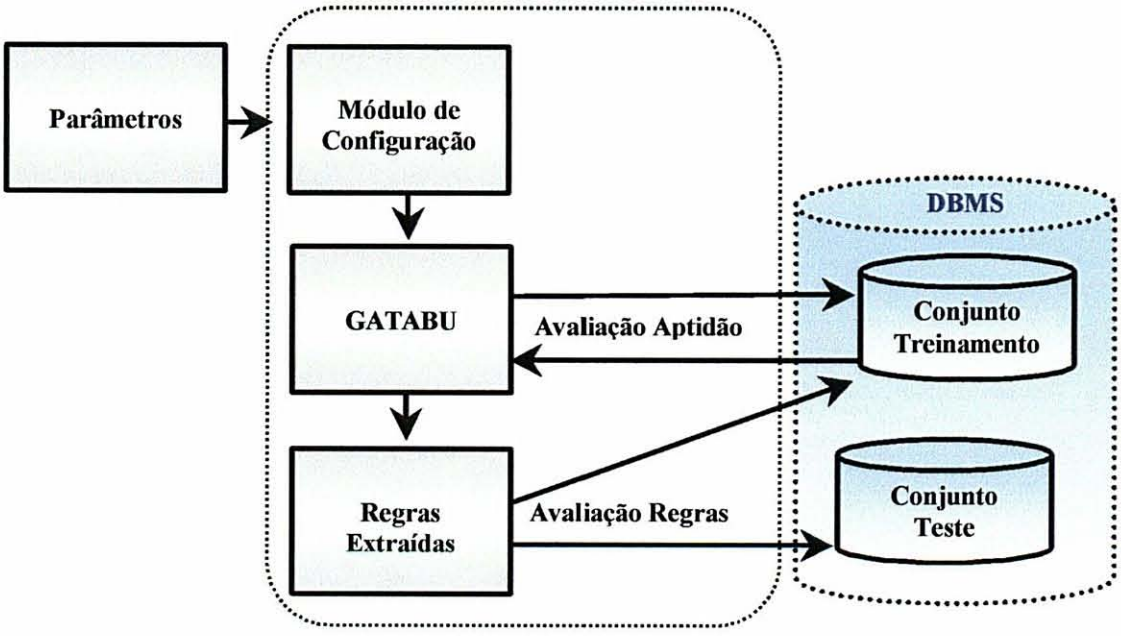

Figura 22 - GADBMS

Como mostra a figura 22, o classificador é formado por três módulos principais. No Módulo de Configuração é feita a leitura dos parâmetros de entrada. GATABU é o núcleo do sistema, onde é utilizado um algoritmo genético restrito para efetuar a busca das regras que formarão o conjunto de soluções. No terceiro módulo, as regras extraídas são organizadas em ordem para posterior verificação da precisão do classificador. Estes módulos são detalhados na próxima seção.

6.1. Módulo de Configuração

Para execução da ferramenta é necessário que os conjuntos de dados estejam armazenados em tabelas na base de dados. Além disso, é necessária a existência de um arquivo que contenha todos os atributos que serão avaliados, assim como os valores possíveis de cada atributo. A primeira linha desse arquivo deverá ser obrigatoriamente as possíveis classes desse conjunto de dados, utilizando formato numérico de 0 a *n*. Nas próximas linhas devem ficar os nomes das colunas, as quais deverão possuir a mesma nomenclatura utilizada na criação das tabelas na base de dados, e os valores possíveis para cada atributo. No caso de atributos discretos, os valores deverão estar todos separados por vírgula e finalizados por ponto final e para atributos contínuos, representados pela palavra "continuous.". Quando um atributo for continuo, a ferramenta irá buscar diretamente na base de dados quais são os valores mínimo e máximo para aquele atributo, ficando armazenados durante a execução do programa. Um exemplo desse arquivo é mostrado na figura a seguir:

```
CLASSES    0,1,2.
v1         0,1.
v2         1,2,3.
v3         0,1.
v4         1,2,3,4,5.
v5         continuous.
v6         0,1.
v7         continuous.
v8         continuous
```

Figura 23 – Arquivo com os atributos do conjunto a ser avaliado

Outros dados utilizados neste módulo são os parâmetros como: probabilidade de cruzamento, mutação, probabilidade de criação de restrições vazias, que regula a quantidade de restrições que em média serão deixadas vazias a cada nova regra criada, número de gerações e execuções, tamanho da população, tamanho da lista longa e da lista curta, medida de similaridade que poderá ser em genótipo ou fenótipo, além de uma medida de distância.

Após informar estes parâmetros, a parte mais importante do processo é a busca das regras, onde é usado um algoritmo genético restrito, explicado na próxima seção.

6.2. GATABU

A ferramenta utiliza um algoritmo genético restrito por listas tabu para evoluir a população de regras. Depois de finalizada a execução, as regras armazenadas em x listas longas serão usadas para criar o classificador. A seguir serão explicadas as principais características deste algoritmo.

6.2.1. População Inicial

Baseado na entrada de parâmetros e após leitura do arquivo com todos os atributos da base de dados a ser minerada, a população inicial é criada aleatoriamente. Quando os indivíduos são criados, um número aproximado de atributos irrestritos é usado. A proporção de atributos irrestritos é definida através da probabilidade de criação de restrições vazias. Os atributos que serão alterados no indivíduo são escolhidos aleatoriamente. Com este mecanismo, é possível diminuir a probabilidade de criação de regras invalidas na população inicial, ou seja, regras que não cobrem exemplos dentro do conjunto de dados.

Um exemplo de regras da população criadas aleatoriamente, utilizando a probabilidade de criação de restrições vazias, é mostrado na figura 24.

```
IF classes = 0                    IF  classes = 1
     v1    = *                        v1    = *
     v2    = *                        v2    = *
     v3    = *                        v3    = *
     v4    = 1                        v4    = 0
     v5    = *                        v5    = *
     v6    = *                        v6    = 1
     v7    = 3                        v7    = *
     v8    = *                        v8    = *
     v9    = *                        v9    = 0
     v10   = *                        v10   = *
     v11   = 3                        v11   = 1
     v12   = *                        v12   = *
     v13   = *                        v13   = *
     v14   = *                        v14   = *
     v15   = *                        v15   = 0
     v16   = *                        v16   = 1
     FITNESS = 0.40000000             FITNESS = 0.38676845
```

Figura 24 – Regras criadas aleatoriamente

Quando o valor de aptidão é calculado, os atributos que possuem o símbolo asterisco (*) não são utilizados no comando SQL executado.

6.2.2. Função de Aptidão

Após criar a população é necessário calcular o valor de aptidão de cada indivíduo na população. A função de aptidão para este problema deve poder qualificar as regras como classificadores parciais, sendo assim, a precisão de uma regra é mais importante que sua habilidade para cobrir todos os exemplos de treinamento. Através da ferramenta desenvolvida, é possível calcular o valor de aptidão escolhendo uma das quatro funções disponíveis:

a) VP+VN / VT (LANGLEY, 1996)

b) VP

c) VP/VT

d) (VP + 1) / (VP + FP + K) função de Laplace (NIBLETT, 1987)

Onde,

VP (Verdadeiros Positivos), exemplos positivos corretamente classificados pela regra.

VN (Verdadeiros Negativos), exemplos negativos corretamente classificados pela regra no sentido que eles não satisfazem o antecedente da regra.

FP (Falsos Positivos), exemplos negativos incorretamente classificados como positivos.

VT, total de exemplos no conjunto.

K, é o número de classes no conjunto.

Um exemplo é denominado "positivo" se ele possui a classe prevista pelo conseqüente da regra.

O valor de aptidão de cada indivíduo é calculado diretamente no conjunto de treinamento, utilizando-se o próprio indivíduo como um comando SQL. Depois disto, um algoritmo genético restrito por listas tabu é aplicado na população que foi criada aleatoriamente para efetuar a busca das regras.

6.2.3. Algoritmo Genético Restrito por Listas Tabu

O algoritmo trabalha com x listas longas com tamanho m e x listas curtas com tamanho n, sendo x a quantidade de classes no conjunto a ser avaliado, m e n o número máximo de indivíduos que serão armazenados nas listas.

Nas listas longas são armazenados apenas os indivíduos que não possuem genótipo/ fenótipo idêntico ou similar. Os melhores indivíduos das gerações anteriores permanecem armazenados nestas listas.

Nas listas curtas estarão armazenados somente os indivíduos das gerações mais recentes. Quando a lista estiver completamente cheia, um novo indivíduo substituirá o mais antigo da lista. Nesta lista os indivíduos poderão ter o mesmo genótipo ou fenótipo.

A quantidade máxima de gerações deverá ser igual ao numero máximo de indivíduos que podem ser armazenados na lista longa. Assim, ao fim de *t* gerações, um conjunto de regras formado pelos indivíduos das listas longas será avaliado através do módulo de Regras Extraídas. Na figura 25 uma idéia básica deste algoritmo é apresentada:

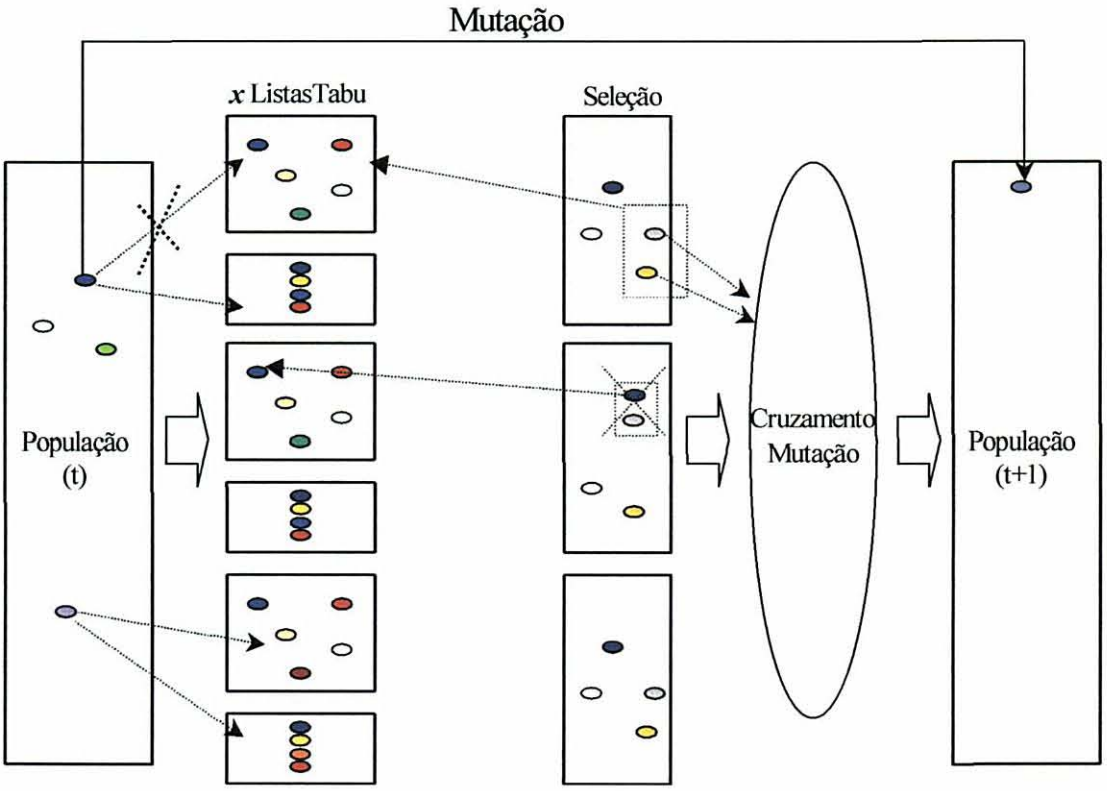

Figura 25 – Algoritmo Genético Restrito por Listas Tabu

Em cada geração é verificado se o melhor indivíduo da população está armazenado em sua lista tabu correspondente, de acordo com a classe a que o indivíduo pertence. O indivíduo somente será armazenado na lista longa se não existir um indivíduo idêntico ou similar nela. Se um indivíduo com genótipo/fenótipo idêntico ou similar precisa ser adicionado a lista longa, ele somente será adicionado se possuir um valor de aptidão maior

que o indivíduo da lista. O indivíduo será removido da lista, sofrerá mutação e será colocado na nova população para participar das próximas gerações.

Os indivíduos são escolhidos pelo método de seleção por torneio, e a lista Tabu é aplicada a somente um dos indivíduos selecionados para cruzamento (KURAHASHI; TERANO, 2000). Se o indivíduo "pai" possuir um indivíduo idêntico ou similar nas listas, os dois indivíduos serão descartados e uma nova seleção por torneio será efetuada, até que seja selecionado um indivíduo diferente em fenótipo/genótipo. Para evitar que este procedimento entre em "loop", o algoritmo pára a execução, se após 50 seleções não for encontrado um novo indivíduo.

6.2.4. Similaridade

De acordo com testes efetuados e dependendo da base de dados que estava sendo utilizada nos experimentos, foi necessário calcular a similaridade de maneiras diferentes. Em alguns casos, utilizando medida de similaridade em fenótipo, a precisão do classificador foi melhor, em outros casos, ao utilizar medida de similaridade em genótipo o resultado não pareceu trazer grande diferença. O que se pôde observar claramente é que quanto ao tempo de execução a medida de similaridade em fenótipo é bem maior do que quando medido em genótipo, principalmente pelo fato de se estar acessando a base de dados mais vezes.

A medida de similaridade em genótipo corresponde à quantidade de atributos similares ou idênticos na regra. Se dois atributos de diferentes regras são idênticos, é adicionado 1.0 a uma variável, e se dois atributos de diferentes regras são similares, é adicionado 0.5 a essa variável. Após comparar todos os atributos entre duas regras, e acumular os valores, essa variável é comparada a uma distancia previamente definida.

A similaridade em fenótipo é medida baseada na quantidade de registros corretamente classificados pela regra. Essa similaridade é calculada da seguinte maneira: os registros cobertos pela regra que está sendo avaliada são separados em uma tabela temporária, resultando em uma quantidade x. Cada regra que compõe a lista longa, é executada nessa tabela temporária, resultando num valor y de registros cobertos. Com estes

dois valores (x e y), é calculado o percentual de indivíduos cobertos pela regra que está sendo avaliada, através da fórmula y/x e comparada com a distância previamente definida.

6.2.5. Cruzamento

A ferramenta desenvolvida trabalha somente com cruzamento em dois pontos. Dois pontos são aleatoriamente escolhidos e trocados entre os indivíduos, resultando em dois novos descendentes. Um exemplo de cruzamento entre duas regras selecionadas, pode ser visto na figura 26:

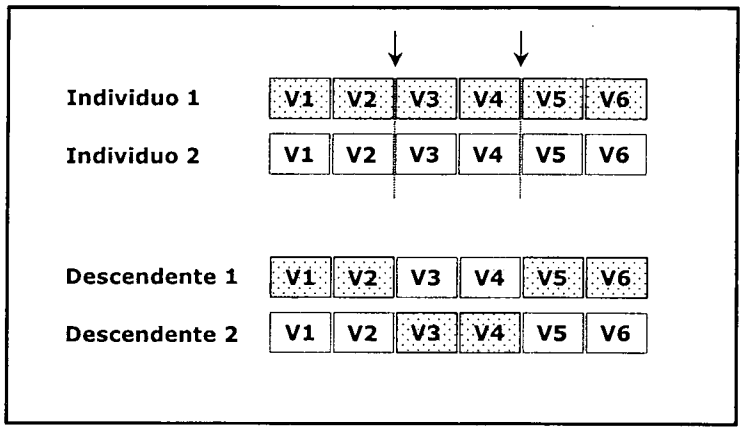

Figura 26 – Dois pontos de cruzamento

6.2.6. Mutação

O operador de mutação busca manter a diversidade da população. Quando aplicado em uma regra, escolhe aleatoriamente um de seus atributos e altera seu valor. Quando aplicado em atributos discretos, a mutação consiste em escolher um dos valores que foram pré-definidos através do arquivo utilizado no módulo de configuração, na seção 6.1. Para atributos contínuos, ao aplicar mutação, serão criados aleatoriamente um novo valor mínimo e um novo valor máximo, baseado nos valores previamente armazenados no módulo de configuração.

6.3. Regras Extraídas

Depois de ter finalizado o processo de busca e ter obtido um conjunto de soluções, neste módulo as regras pertencentes às listas longas serão organizadas de uma maneira apropriada, buscando obter uma boa precisão do classificador. Nesta abordagem foi utilizado conjunto de regras ordenado. No conjunto de regras ordenado, todas as regras são agrupadas em uma ordem particular. A ordenação é feita com base no valor de aptidão de cada regra. Se existirem várias regras com um mesmo valor de aptidão, é levada em consideração a quantidade de atributos da regra. A regra com menos atributos será utilizada antes de uma regra com mais atributos, no caso de valores de aptidão iguais.

Para classificar um novo exemplo, cada regra é testada em ordem até que alguma regra que cobre algum exemplo é encontrada. As regras que não cobrem nenhum exemplo no conjunto de treinamento são eliminadas. Para evitar que uma regra cubra os mesmos exemplos já cobertos por outra regra, a cada avaliação, são retirados os indivíduos que já foram cobertos. A classe que possuir mais registros que não foram cobertos pelas regras será a classe padrão, a qual será atribuída a um exemplo que não é coberto por nenhuma regra.

6.4. Interface de Entrada

Para a execução da ferramenta, alguns parâmetros precisam ser informados. Para cada execução um arquivo é criado contendo os parâmetros, o tempo de execução, taxa de erro e regras encontradas durante a execução do algoritmo. Um exemplo dos parâmetros informados antes da execução do algoritmo é mostrado na figura 27.

Figura 27 – Interface de entrada do algoritmo

Nas figuras 28 e 29, é possível visualizar um exemplo dos dados armazenados no arquivo criado a cada execução da ferramenta:

```
Hora Inicial.....: 15:23:13
Hora Final.....: 14:31:18
Entrou em Loop na Geração.....: 20
Quantidade de execuções.....: 3
Número da execução.....: 1
Gerações requeridas.....: 200
Probabilidade de crossover.....: 0.85
Probabilidade de mutação.....: 0.200
Probabilidade de torneio.....: 1.00
Tipo similaridade.....: 2
Medida de similaridade.....: 4
Quantidade de classes.....: 3
Quantidade de atributos.....: 9

Tamanho da População.....: 200
Tamanho da Lista Longa.....: 200
Tamanho da Lista Curta .....: 20
Percentual população boa.....: 0.40
Geração do Fitness.....: 4
Tabela para avaliação.....: smoke
Tabela para teste.....: smoke_test
Arquivo NAMES (sem extensão)...: smoke.names
Probabilidade Restrições Vazias: 0.75
Taxa de erro.....: 0.3050
Regras Encontradas.....: 11
Regras Executadas.....: 10
```

Figura 28 – Parâmetros gravados

```
* * REGRAS GERADAS NA BASE: smoke * *
 IF   v2 =  1
      v5 <= 7.76
      v7 >= -0.68
 THEN CLASSES     = 2
 FITNESS          = 65
ELSE
 IF   v1 =  0
      v2 =  1
      v3 =  0
      v4 =  2
      v5 <= 12.03
      v6 =  0
      v8>= -0.02 AND <= 2.92
 THEN CLASSES     = 1
 FITNESS          = 7
ELSE
 (DEFAULT) CLASSES =    2
```

```
* REGRAS EXECUTADAS NA BASE: smoke_test *
 IF   v2 =  1
      v5 <=  7.76
      v7 >= -0.68
 THEN CLASSES     = 2
 FITNESS          = 30
ELSE
 IF   v1 =  0
      v2 =  1
      v3 =  0
      v4 =  2
      v5 <=  12.03
      v6 =  0
      v8 >= -0.02 AND <= 2.92
 THEN CLASSES     = 1
 FITNESS          = 1
ELSE
(DEFAULT)CLASSES  = 2
 FITNESS          = 661
```

Figura 29 – Regras Classificadas

Neste arquivo, as regras são mostradas junto com o valor de aptidão correspondente. As regras geradas correspondem às regras originadas na base de treinamento e que após terem sido ordenadas e re-executadas, continuaram cobrindo pelo menos um indivíduo. As regras executadas correspondem às regras geradas na base de treinamento, e que são utilizadas para calcular a precisão do classificador no conjunto de teste.

No próximo capítulo, são detalhadas as bases de dados utilizadas nos testes, assim como uma comparação dos resultados obtidos com o sistema implementado frente a outros classificadores.

7. Comparação do Classificador

Com o objetivo de fazer uma comparação com outros trabalhos, foram avaliados quatro conjuntos de dados usados por LOPES e POZO (2001), os quais foram comparados com a metodologia utilizada por LIM, LOH e SHIH (1999), onde 33 algoritmos de classificação foram comparados em diferentes conjuntos de dados em termos de exatidão do classificador e muitas outras características. Para comparação do classificador implementado, foram utilizados como base de comparação apenas 22 desses algoritmos, todos baseados em árvores de decisão e regras (Anexo 1). Na tabela 2, estão descritos os conjuntos de dados utilizados nos experimentos.

Tabela 2 – Conjuntos de dados analisados (LIM;LOH;SHIH,1999;LOPES;POZO,2001)

Nome	Tam.	Classes	Atributos		Descrição dos Conjuntos Analisados
			(*D*)	(*C*)	
Bld	345	2	6		Diagnóstico de doenças de fígado em homens via
Bld+	345	2	15		testes de sangue e consumo de álcool.
Pid	532	2	7		Diagnóstico de diabetes de pacientes da Índia
Pid+	532	2	15		Diagnóstico pelo estado psicológico + testes.
Smo	1855	3	3	5	Previsão da atitude de pessoas diante de restrições a
Smo+	1855	3	10	5	fumantes no local de trabalho.
Vot	435	2		16	Classificação de um político como republicano ou
Vot+	435	2		30	democrata de acordo com suas votações

Na tabela acima, *D* é o número de atributos discretos, *C* é o número de atributos contínuos e o símbolo + representa os conjuntos de dados com ruídos. Para cada conjunto de dados, uma variação foi criada pela adição de ruído, o qual é simplesmente a adição de mais atributos com valores aleatórios.

Nos experimentos executados foram utilizados dois métodos para estimar a taxa de erro:

- Para conjunto de dados com mais de 1000 exemplos, foi utilizado o conjunto de testes para estimar a taxa de erro.

- Para conjuntos pequenos, com menos de 1000 exemplos, foi utilizado o procedimento chamado *Cross-Validation* (LIM;LOH;SHIH,1999; LOPES; POZO, 2001). Neste procedimento o conjunto de dados é dividido em 10 subconjuntos distintos, contendo aproximadamente a mesma proporção de registros de cada classe do conjunto original. O classificador é executado 10 vezes, cada vez usando um subconjunto diferente para teste e usando os demais 9 subconjuntos como treinamento. Cada execução resulta em uma taxa de erro parcial. A taxa de erro total é obtida através da média de todas as 10 estimativas dos subconjuntos.

7.1. Resultados obtidos

Para todas as execuções, foi definido um número máximo de 200 gerações, todavia, na maioria das vezes, as gerações não ultrapassaram de 50. Nesta ferramenta a execução é interrompida automaticamente se, após 50 novas tentativas de cruzamento, o indivíduo pai possuir um indivíduo idêntico ou semelhante nas listas de restrições (seção 6.2.3).

Foi utilizada população e lista longa com tamanho 200 e lista curta com tamanho 20. A probabilidade de cruzamento, probabilidade de mutação e medida de similaridade foram ajustadas de acordo com o conjunto de dados a ser avaliado.

Alguns testes foram executados para chegar aos melhores parâmetros de cada conjunto analisado. Um ponto importante a ser comentado é que em todos os conjuntos avaliados, o GADBMS foi executado com uma população de 200 indivíduos obtendo a mesma precisão de quando foi executado com uma população de 2000 indivíduos.

Através dos testes, foi possível verificar que quanto maior a quantidade de atributos no domínio, maior é a necessidade de se aplicar uma probabilidade de criação de restrições vazias acima de 80%. Para domínios com menos de 10 atributos, esta taxa oscilou entre 50 e 60%.

Em um dos primeiros testes com os melhores parâmetros, mostrado na Tabela 3, foi utilizada similaridade em genótipo. Trabalhando com esta similaridade, pode-se notar que o tempo de execução e a quantidade de regras extraídas foram menores do que quando foi

utilizada similaridade em fenótipo, porém estas exceções não foram significantes para que o classificador encontrasse a melhor precisão.

Tabela 3 – Similaridade aplicada no conjunto de dados Smo

	Genótipo	Fenótipo
Regras encontradas	3	10
Taxa de erro	0,3050	0,3050
População	200	200
Distancia	4	4
Tempo (hh:mm:ss)	00:01:56	00:16:58

Para verificação dos resultados obtidos quando uma nova função de aptidão é aplicada, foram executados quatro testes sobre a base de dados Vot, mostrados na tabela 4:

Tabela 4 – Resultados obtidos com aplicação de funções de aptidão diferentes na base Vot

	(1)	(2)	(3)	(4)
Média	**VP+VN / VT**	**VP**	**VP/VT**	**(VP + 1) / (VP + FP + K)**
Taxa de erro	0,1051	0,0901	0,0685	0,0597
Gerações (Convergência)	182	18	50	39
Regras Encontradas	6	3	3	6
Regras Executadas	5	3	3	5

Os parâmetros utilizados para estes testes foram todos iguais, com população de tamanho 60, medida de similaridade em fenótipo com limiar (*threshold*) de distância igual a 6, cruzamento em dois pontos, com probabilidade de 0.85 e mutação 0.20, apenas alterando a função para calculo do valor de aptidão dos indivíduos. Como pode ser visto, o melhor resultado foi encontrado utilizando a quarta função. A função número 1, não obteve um resultado satisfatório em tempo de processamento, principalmente pela quantidade de gerações utilizadas. As regras encontradas utilizando a função 2 e 3 foram bastante semelhantes.

Outro teste efetuado, foi referente a utilização de tamanhos de populações diferentes, mostrado na tabela a seguir. Para comparação foram utilizadas as bases SMO, VOT e PID.

Tabela 5 – Resultados obtidos com aplicação de tamanhos de populações diferentes

			Tamanho da População	
			200	60
MÉDIA	Smo	Taxa de erro	0,3050	0,3050
		Gerações	40	36
		Regras Encontradas	11	13
		Regras Executadas	11	13
	Vot	Taxa de erro	0,0503	0,0597
		Gerações	20	39
		Regras Encontradas	6	6
		Regras Executadas	6	5
	Pid	Taxa de erro	0,2576	0,2610
		Gerações	48	37
		Regras Encontradas	15	13
		Regras Executadas	13	12

Como pode ser visto na tabela 5, o tamanho da população não influiu notoriamente na precisão do classificador. Utilizando a base Smo que possui atributos contínuos e discretos e conjunto de dados que possui 1855 exemplos, ao comparar os resultados obtidos com a média de três execuções, a precisão alcançada com 200 indivíduos não foi diferente de quando foram utilizados menos indivíduos. Ao executar com a base Vot, onde todos os atributos são discretos, e o conjunto de dados possui 435 exemplos, a precisão média utilizando uma população de 200 indivíduos foi um pouco melhor, se comparada a utilização de 60 indivíduos, o mesmo acontecendo com a base Pid, que possui todos os atributos contínuos.

Outra importante medida é a margem de erro, dada por $(p(1-p)/n)^{1/2}$, onde p é a menor taxa de erro para o conjunto de dados e n é o número de exemplos de treinamento. O resultado do classificador é considerado próximo ao melhor se a diferença entre o resultado e o melhor dos conjuntos de dados não for maior que esta margem de erro (LIM;LOH;SHIH, 1999). A tabela 6, mostra os resultados obtidos pelo GADBMS, frente aos classificadores baseados em árvores e regras, testados no trabalho de LIM;LOH;SHIH.

Tabela 6 – Bases de dados em que o GADBMS esteve próximo aos melhores resultados

BASE	Taxa Erro Min.	Tamanho Treinam.	Margem de Erro	Min.+ME	GADBMS	Próximo Melhor
BLD	0,2790	310	0,0255	0,3045	0,3862	
BLD+	0,3130	310	0,0263	0,3393	0,4219	
PID	0,2210	478	0,0190	0,2400	0,2576	
PID+	0,2210	478	0,0190	0,2400	0,2705	
SMO	0,3040	1855	0,0107	0,3147	0,3050	x
SMO+	0,3050	1855	0,0107	0,3157	0,3050	x
VOT	0,0364	391	0,0095	0,0459	0,0503	
VOT+	0,0412	391	0,0101	0,0513	0,0455	x
Media	0,2151		0,0163	0,2314	0,2553	

GADBMS obteve resultados próximos ao melhor dos classificadores, somente em três bases de dados (SMO, SMO+,VOT+). Segundo a análise de LIM;LOH e SHIH, das bases analisadas as mais fáceis de classificar são: VOT e VOT+. Na base de dados SMO, somente um dos algoritmos (T1) conseguiu atingir a taxa de erro de 0,3040, ficando todos os outros com taxa igual ou maior que 0,3050. No caso da base de dados SMO+, o classificador GADBMS conseguiu atingir a taxa de erro mínima dos outros classificadores.

Os resultados mínimo e máximo, obtidos pelos classificadores baseados em árvores e regras (LIM;LOH;SHIH, 1999), a mediana encontrada pelo algoritmo de LOPES e POZO e pelo GADBMS nas bases analisadas, são mostrados na tabela 7.

Tabela 7 – Comparação entre os classificadores baseados em regras

	LIM;LOH;SHIH			LOPES;POZO	GADBMS
Nome	Min.	Max.	Mediana	Média	Média
Bld	0,2790	0,4320	0,3220	0,3775	0,3862
Bld+	0,3130	0,4410	0,3490	0,3475	0,4219
Pid	0,2210	0,3100	0,2380	0,2593	0,2576
Pid+	0,2210	0,3180	0,2500	0,2778	0,2705
Smo	0,3040	0,4240	0,3050	0,3008	0,3050
Smo+	0,3050	0,4110	0,3050	0,3068	0,3050
Vot	0,0364	0,0580	0,0457	0,0582	0,0503
Vot+	0,0412	0,0662	0,0469	0,0611	0,0455
Média	0,2151	0,3075	0,2327	0,2486	0,2553

GADBMS obteve uma taxa de erro de 0,2553. A taxa de erro do algoritmo esteve em todas as execuções, dentro dos resultados obtidos no trabalho feito por LIM, LOH e SHIH (1999). Comparado à taxa de erro dos 23 algoritmos baseados em arvores de decisão e regras, apresentados por (LIM; LOH; SHIH, 1999; LOPES; POZO, 2001), GADBMS não é estatisticamente diferente desses algoritmos. De acordo com o método Tukey (RUPPERT, 1981), uma diferença entre a média da taxa de erro de dois algoritmos é estatisticamente diferente ao nível de 10% se eles diferem mais do que 0,058 (LIM;LOH;SHIH, 1999).

Outro item testado no trabalho corresponde a análise de *rank* (LIM;LOH; SHIH, 1999). Nesta análise cada algoritmo é classificado de acordo com a taxa de erro. O algoritmo que possuir a menor taxa de erro é o primeiro do rank, o próximo com a menor taxa de erro é o segundo do rank e assim por diante. Caso vários algoritmos tenham a mesma taxa de erro, o rank desses algoritmos deverá ser a média. Baseado nesta análise, a tabela 8 foi construída para verificação do rank obtido pelo GADBMS, utilizando as bases de dados analisadas.

Tabela 8 – Análise por Rank entre os algoritmos baseados em árvores e regras

	BLD		BLD+		PID		PID+		SMOKE		SMOKE+		VOTE		VOTE+		Rank
QL1	0,3200	9,5	0,3690	15,0	0,2250	3,0	0,2210	1,5	0,3050	8	0,3050	7,5	0,0364	1,0	0,0503	16	7,6
QU1	0,3990	19,0	0,3760	16,0	0,2260	4,5	0,2300	4,5	0,3050	8	0,3050	7,5	0,0412	4,0	0,0412	1	8,1
QL0	0,3060	4,0	0,3800	17,0	0,2230	2,0	0,2210	1,5	0,3050	8	0,3050	7,5	0,0503	18,0	0,0480	13	8,9
IC1	0,3190	8,0	0,3130	1,0	0,2370	10,0	0,2330	6,5	0,3050	8	0,3050	7,5	0,0480	13,5	0,0548	19	9,2
ST1	0,3260	13,0	0,3510	12,0	0,2500	18,5	0,2440	8,0	0,3050	8	0,3050	7,5	0,0432	5,0	0,0432	2	9,3
QU0	0,3890	18,0	0,4030	20,0	0,2300	7,0	0,2300	4,5	0,3050	8	0,3050	7,5	0,0435	8,0	0,0435	4,5	9,7
FTL	0,4130	21,0	0,4190	21,0	0,2210	1,0	0,2250	3,0	0,3050	8	0,3050	7,5	0,0457	11,5	0,0457	10	10,4
IC0	0,3270	14,0	0,3270	4,0	0,2390	12,0	0,2480	10,0	0,3190	19,0	0,3050	7,5	0,0435	8,0	0,0457	10	10,6
STO	0,3110	6,0	0,3260	3,0	0,2370	10,0	0,2690	20,0	0,3050	8	0,3050	7,5	0,0500	16,0	0,0500	14	10,6
OCU	0,3500	16,0	0,3470	11,0	0,2370	10,0	0,2560	15,0	0,3120	16,0	0,3050	7,5	0,0435	8,0	0,0435	4,5	11,0
C4R	0,2920	2,0	0,3200	2,0	0,2270	6,0	0,2330	6,5	0,3530	21,0	0,3490	20,0	0,0526	20,5	0,0457	10	11,0
IM0	0,3120	7,0	0,3360	8,0	0,2460	14,0	0,2650	19,0	0,3110	15,0	0,3390	19,0	0,0367	2,0	0,0457	10	11,8
C4T	0,3080	5,0	0,3290	6,0	0,2420	13,0	0,2520	13,0	0,3050	8	0,4050	22,0	0,0480	13,5	0,0503	16	12,0
FTU	0,4010	20,0	0,4020	19,0	0,2470	15,5	0,2580	16,0	0,3050	8	0,3050	7,5	0,0435	8,0	0,0435	4,5	12,3
IM	0,3200	9,5	0,3400	9,5	0,2330	8,0	0,2460	9,0	0,3050	8	0,3080	16,0	0,0503	18,0	0,0594	21	12,4
T1	0,4320	23,0	0,4410	23,0	0,2500	18,5	0,2500	11,5	0,3040	1,0	0,3170	18,0	0,0435	8,0	0,0435	4,5	13,4
OCM	0,2790	1,0	0,3670	14,0	0,2470	15,5	0,2610	18,0	0,3050	8	0,3050	7,5	0,0580	23,0	0,0662	23	13,8
CAL	0,4200	22,0	0,3980	18,0	0,2260	4,5	0,2500	11,5	0,3160	17,0	0,3100	17,0	0,0457	11,5	0,0457	10	13,9
IB0	0,3220	11,5	0,3330	7,0	0,2580	22,0	0,2590	17,0	0,3930	22,0	0,3870	21,0	0,0386	3,0	0,0506	17	15,1

GADBMS	0,3862	17,0	0,4219	22,0	0,2576	21,0	0,2705	21,0	0,3050	8	0,3050	7,5	0,0503	18,0	0,0455	7	15,2
LMT	0,3220	11,5	0,3620	13,0	0,2490	17,0	0,2530	14,0	0,3500	20,0	0,3060	15,0	0,0483	15,0	0,0529	18	15,4
OCL	0,2960	3,0	0,3280	5,0	0,3100	23,0	0,3180	23,0	0,3170	18,0	0,3050	7,5	0,0574	22,0	0,0651	22	15,4
IB	0,3280	15,0	0,3400	9,5	0,2520	20,0	0,2780	22,0	0,4240	23,0	0,4110	23,0	0,0526	20,5	0,0554	20	19,1

Através da tabela acima, é possível verificar que o GADBMS obteve a posição 15,2 no rank. Segundo HOLLANDER (1999), a diferença na média dos ranks maior que 8.7 é estatisticamente significante ao nível de 10%, o que não ocorre para o GADBMS em relação aos outros algoritmos.

Durante os experimentos, as execuções não tiveram uma variação significante entre as execuções quando os mesmos parâmetros genéticos foram executados em um mesmo conjunto de dados. A adição de atributos como ruídos nos conjuntos de dados não aumentou significantemente a taxa de erro.

Outra comparação efetuada se refere à quantidade de regras encontradas nas execuções do classificador sobre as bases de dados:

Tabela 9 – Comparação entre os classificadores em quantidade de regras

	LIM;LOH;SHIH (1999)			LOPES;POZO	GADBMS
Nome	Min.	Max.	Mediana	Média	Média
Bld	2	31247	10	10	13
Bld+	2	25177	6	11	7
Pid	2	30375	7	20	13
Pid+	2	29249	7	19	12
Smo	1	9884	2	42	3
Smo+	1	9718	2	45	3
Vot	2	46695	2	13	6
Vot+	2	31837	2	16	8
Média			5	22	8

A quantidade de regras encontradas através do classificador ficou abaixo da média encontrada no trabalho de LOPES e POZO (2001). Com certeza, é possível melhorar ainda mais estes resultados, principalmente pelo fato de ter sido efetuado uma quantidade de testes muito pequenas e poucas bases de dados terem sido testadas.

8. Conclusões

Este trabalho introduziu o GADBMS, uma ferramenta de Mineração de Dados para a tarefa de classificação que utiliza um algoritmo genético restrito por x listas Tabu para efetuar a busca das regras. Nesta abordagem, as regras pertencentes a qualquer uma das classes podem ser encontradas em apenas uma execução, diferente da abordagem apresentada por LOPES (2001), onde a cada execução são encontradas as regras pertencentes a uma determinada classe.

Anterior ao desenvolvimento da ferramenta, três técnicas que têm o objetivo comum de melhorar algoritmos genéticos foram estudadas, implementadas e comparadas: Compartilhamento de Recursos ou Sharing (GOLDBERG; RICHARDSON, 1987), Evolução Cooperativa (POTTER; DE JONG, 2000) e a integração de algoritmos genéticos com Busca Tabu (KURAHASHI; TERANO, 2000), o qual foi chamado de AG Restrito.

A Evolução Cooperativa é uma abordagem que pode levar vantagem da distribuição de espécies em diferentes computadores, mas isto pode adicionar complexidade na administração, como por exemplo, quando a evolução estagna. Neste caso, pode ser que existam poucas espécies no ecossistema com as quais construir uma boa solução; então, uma espécie nova será criada e sua população inicializada aleatoriamente. Esta abordagem, quando usada em forma consecutiva não parece trazer grandes vantagens.

O Algoritmo Sharing e o AG Restrito precisam do uso de medida de similaridade para trabalhar melhor, o que pode ser complexo em domínios de aplicações diferentes. Quando a medida de similaridade é aplicada nestas técnicas, há uma grande melhoria, tanto em tempo de execução, como para chegar ao problema objetivo.

Apesar destas três técnicas mostrarem serem úteis de diferentes maneiras, optou-se pela utilização do AG Restrito para implementação da ferramenta de Mineração de Dados. No GADBMS, todas as regras possíveis para cada uma das classes a serem previstas, podem ser encontradas em apenas uma execução. O sistema demonstrou ser eficiente mesmo trabalhando com uma população pequena, reduzindo o tempo de processamento e obtendo o mesmo resultado de quando foi utilizada uma população maior.

O uso de comandos SQL mostrou ser bastante interessante, principalmente devido ao fato de ser possível avaliar as regras diretamente na base de dados. Uma vez que o tamanho da população a ser utilizado pode ser pequena, isto pode trazer uma grande vantagem quando a ferramenta for executada em uma grande quantidade de dados.

Com os experimentos executados, foi possível verificar a real necessidade de se utilizar uma função de aptidão adequada. Apesar do sistema possuir quatro funções de aptidão possíveis de serem escolhidas, somente com a função de aptidão de Laplace (NIBLETT, 1987) é que o sistema obteve os melhores resultados.

A medida de similaridade aplicada também mostra ser um parâmetro importante na busca das regras. Quando utilizada em genótipo, como no conjunto de dados SMO teve um tempo de processamento bem menor, e encontrou poucas regras no espaço de busca. Em outros casos, como na analise sobre os conjuntos de dados VOT e PID, a medida de similaridade em fenótipo mostrou ser mais eficaz, melhorando a precisão do classificador. A introdução de ruídos nas bases de dados não afetou os resultados obtidos pela ferramenta nos conjuntos de dados analisados.

A ferramenta foi comparada com 23 outros algoritmos baseados em árvores e regras (LIM; LOH; SHIH, 1999; LOPES; POZO, 2001), e obteve resultados promissores. Poucas combinações de parâmetros foram tentadas, o que significa que a precisão do classificador em muitas das bases de dados pode ser melhorada se outros valores de parâmetros forem utilizados.

8.1. Trabalhos Futuros

Como proposta para trabalhos futuros, sugere-se:

- Executar a ferramenta em conjuntos maiores de dados, analisando também o tempo de execução e as regras encontradas;

- Utilizar novos conjuntos de parâmetros e funções de aptidão que possam vir a aumentar a precisão do classificador, pois uma função de aptidão adequada a um determinado tipo de problema pode ajudar a melhorar a precisão do algoritmo;
- Encontrar uma medida de distância adequada que possa ser aplicada indiferente aos tipos de atributos que estão sendo avaliados;
- Utilizar novas heurísticas na busca das regras, como a utilização da Evolução Cooperativa;

Referências

BANZHAF, W.; NORDIN, P. et al. **Genetic programming – An introduction: On the automatic evolution of computer programs and its applications**. Morgan Kaufmann Publishers, 1998.

BECKER, R. A.; CHAMBERS, J. M.; WILKS, A.R. **The new S language.** Wadsworth, 1988.

BREIMAN, L.; FRIEDMAN, J.; OLSHEN, R.; STONE, C. **Classification and regression trees**. New York:Chapman and Hall, 1984.

BRODLEY, C. E.; UTGOFF, P. E. Multivariate decision trees. **Machine Learning**, Boston, v.19, p.45-77, 1995.

BUNTINE, W. Learning classification trees. **Statistics and Computing**, London, v.2, p.63-73, 1992.

CANTU-PAZ, E. Topologies, migration rates, and multi-population parallel genetic algorithms, In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, 1, 1999, p.91-98. **Proceedings....** 1999.

CARUANA, R. A.; SCHAFFER, J. D. Representation and hidden bias: Gry vs. binary coding for genetic algorithms. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 5, 1988. **Proceedings...** Morgan Kaufmann, 1988.

CAVALHEIRO, A. F.; POZO, A. R. Genetic algorithm and complex problems: An empirical comparative study. In: INTERNATIONAL CONFERENCE ON APPLIED INFORMATICS , **Proceedings...** . 2002.

CAVICCHIO, JR., D. J. **Adaptive search using simulated evolution**. Doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microlms No. 25-199), 1970.

CLARK, L. A.; PREGIBON, D. Tree-based models. In: CHAMBERS, J. M.; HASTIE, T. J. (Eds.), **Statistical models in S**. New York: Chapman & Hall, 1993. p.377-419.

COELHO, A. An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. CONGRESS ON EVOLUTIONARY COMPUTATION, **Proceedings...** .1999, p.1-11.

COSTA, D. An evolutionary tabu search algorithm and the NHL scheduling problem. **Information Systems and Operational Research**, 33 (3), p.161-178, 1995.

DE JONG, K. A. **An Analysis of the Behavior of a Class of Genetic Adaptative Systems**. PhD thesis, University of Michigan, 1975.

DE JONG, K.; SPEARS, W. Using genetic algorithms to solve NP-complete problems. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 3, 1989. **Proceedings...** .1989

ESHELMAN, L.; SHAFFER, J. Preventing premature convergence in genetic algorithm by preventing incest. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 4, p.115-122, 1991. **Proceedings...** .1991

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P.; UTHURUSAMY, R. **Advances in knowledge Discovery and data mining**. Menlo Park: AAAAI, 1996.

FONSECA, C.; FLEMING, P. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 5, p.416-423, 1993. **Proceedings...** .1993.

FRAWLEY, W. J.; PIATETSKY-SHAPIRO, G.; MATHEUS, C. J. **Knowledge discovery in databases: an overview**, Piatsky-Shapiro, G. and Frawley, W.J. Editores, In: Knowledge Discovery in Databases, MIT Press, Cambridge, MA, 1991.

FREITAS, A.; LAVINGTON, S. H. **Mining very large databases with parallel processing**. Boston: Kluwer Academic, p.208, 1998.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers and Operations Research**, v. 13, p.533-549, 1986.

GLOVER, F. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). **Discrete Applied Mathematics**, v. 49, p.231-255, 1994.

GLOVER, F.; KELLY, J.; LAGUNA, M. Genetic algorithms and tabu search: hybrid for optimization. **Computers and Operations Research**, v. 22(1), p.111-134, 1995.

GOLDBERG, D. E. **Genetic algorithms in search, optimization and machine learning**. Alabama: Addison Wesley, 1989. 413p.

GOLDBERG, D. E. A note on Boltzman tournament selection for genetic algorithms and population oriented simulated annealing. **Complex Systems**, v. 4, p.445-460, 1990.

GOLDBERG, D. E.; DEB, K. A comparative analysis of selection schemes used in genetic algorithms. In: RAWLINS, G. (Ed.), **Foundations of Genetic Algorithms**. San Francisco, CA: Morgan Kaufmann. 1991.

GOLDBERG, D. E.; RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 2, **Proceedings...** . 1987.

HAND, D. J. **Discrimination and Classification**, Chichester, U.K.: John Wiley and Sons, 1981.

HAYKIN, S. **Neural Network, Neural Networks - Comprehensive Foundation**, Macmillan, College Publishing Company, 1994.

HIROYASU, T.; MIKI, M.; WATANABE, S. Divided range genetic algorithms in multiobjective optimization problems. In: INTERNATIONAL WORKSHOP ON EMERGENT SYNTHESIS, p.57-66, 1999. **Proceedings...** .1999

HOLLAND, J. H. **Adaptation in natural and artificial systems**. Michigan: University of Michigan, 1975.

HOLTE, R. C. Very simple classification rules perform well on most commonly used datasets. **Machine learning**, Boston, v.11, p.63-90, 1993.

HORN, J.; NAFPLIOTIS, N. Multiobjective optimization using the niched pareto genetic algorithm. **Technical Report IlliGAl Report 93005**, University of Illinois at Urabama-Champaign, 1993

KITANO, H. Empirical studies on the speed of convergence of neural network training using genetic algorithms, In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 8, 1990, **Proceedings....** 1990.

KITANO, H. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. **Physica D**, Amsterdam, v. 75, p. 225-238, 1994.

KOZA, J. R. **Genetic programming: on the programming of computers by means of natural selection**. Cambridge, MA: MIT Press, 1992.

KOZA, J. R. **Genetic programming II: Automatic discovery of reusable programs**. Cambridge, MA: MIT Press, 1994.

KRISHNAMACHARI, B. **Global optimization in the design of mobile communication systems**. Ithaca, 1999. 175p. Master of Science in Electrical Engineering, Cornell University.

KURAHASHI, S.; TERANO, T. A genetic algorithm with Tabu search for multimodal and multiobjective function optimization. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 2, **Proceedings....** 2000, p.291-298.

LANGLEY, P. **Elements of Machine Learning,** ISBN 1-55860-301-8, San Francisco, California, Morgan Kaufmann, 1996.

LIM, T. S.; LOH, W. Y.; SHIH, Y. S. A comparison of prediction accuracy, complexity and training time of 33 old and new classification algorithms. **Machine Learning Journal**, Boston, 1999.

LOH, W. Y.; SHIH, Y. S. Split selection methods for classification trees. **Statistica Sinica**, v.7, p.815-840, 1997.

LOH, W. Y.; VANICHSETAKUL, N. Tree-structured classification via generalized discriminate analysis (with discussion). **Journal of the American Statistical Association**, Boston, v.83,p.715-728, 1988.

LOPES, F. **Algoritmo genético restrito por listas Tabu no contexto de mineração de dados**. Curitiba, 2001. 77p. Dissertação (Mestrado em Informática), Universidade Federal do Paraná.

LOPES, F.; POZO, A. R. Genetic algorithm restricted by tabu lists in data mining. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, **Proceedings...** . 2001, p.178.

MALEK, M.; GURUSWAMY M., et al. Serial and parallel simulated annealing and tabu search algorithms for the travelling salesman problem. **Annals of Operations Research**, v. 21, p.59-84, 1989.

MANTAWY, A.; ABDEL-MAGID, Y.; SELIM, S. A new genetic based tabu search algorithm for unit commitment problem. **Eletric Power Systems Research**, v. 49, p.71-78, 1999.

McLACHLAN, G. **Discriminant Analysis and Statistical Pattern Recognitions**, New York: Wiley, 1992.

MEYER, T. P.; PACKARD, N. H. Local forecasting of high-dimensional chaotic dynamics. In: CASDAGLI, N.; EUBANK, S. (Eds.), **Nonlinear Modeling and Forecasting**. Addison-Wesley. 1992.

MITCHELL, M. **An introduction to genetic algorithms**. Cambridge: Mit Press. 1997. 207 p.

MUHLENBEIN, H. Parallel genetic algorithm in Combinatorial Optimization. **Computer Science and Operation Research**, Pergamon Press.: p.441-456, 1992.

MUHLENBEIN, H.; GORGES-SCHELEUTER, M.; KRAMER, O. Evolution algorithms in combinatorial optimization, **Parallel Computing**, v. 7, p.65-85, 1998.

MULLER W.; WYSOTZKI, F. Automatic construction of decision trees for classification. **Annals of Operations Research**, Amsterdam, v. 52, p.231-247, 1994.

MULLER W.; WYSOTZKI, F. The decision tree algorithm CAL5 based on a statistical approach to its splitting algorithm. In: NAKHAEZADEH, G.; TAYLOR, C. C. (Eds.), **Machine learning and statistics**: the interface, New York: J. Wiley, 1997, p.45-65.

MURTHY, S. K.; KASIF, S.; SALZBERG, S. A system for induction of oblique decision trees. **Journal of Artificial Intelligence Research**, Charlottesville, v. 2, p. 1-33, 1994.

NIBLETT, T. Constructing decision trees in noisy domains. In: EUROPEAN WORKING SESSION ON LEARNING, 2., 1987. **Proceedings...** . Wilmslow, 1987. p.67-78.

NODA, E.; FREITAS, A. A.; LOPES, H. S. Discovering interesting prediction rules with a genetic algorithm. In: CONGRESS ON EVOLUTIONARY COMPUTATION (CEC-99). **Proceedings...** . Washington D. C., 1999, p. 1322-1329.

POTTER, M.A.; DE JONG, K.- **The Design and Analysis of a Computational Model of Cooperative Coevolution**. PhD thesis, George Mason University, 1997.

POTTER, M. A.; DE JONG, K. Cooperative coevolution: an architecture for evolving coadapted subcomponents. In: EVOLUTIONARY COMPUTATION, 8(1), **Proceedings...** .2000, p.1-29.

POWELL, D.; TONG, S.; SKOLNICK, M. EnGENEous: Domain Independent machine learning for design optimization. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 3, p.151-159 1989. **Proceedings...** .1989

SCHAFFER, D. Multiple objective optimization with vector evaluated genetic algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 1, p.93-100, 1985.

SRINIVAS, N.; DEB, K. **Multiobjective optimization using nondominated sorting in genetic algorithms.** Technical reports, Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India, 1993.

TAMAKI, H.; KITA, H.; KOBAYASHI, S. Multiobjective optimization by genetic algorithms. In: INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION , p. 517-522, 1996.

TSUTSUI, S.; FUJIMOTO, Y. Forking genetic algorithms with blocking and shrinking modes. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 5, p.206-213, 1993. **Proceedings...** .1993.

TSUTSUI, S.; FUJIMOTO, Y. Extended forking genetic algorithms for order representation. CONFERENCE ON EVOLUTIONARY COMPUTATION, **Proceedings....**1994, p.170-175.

ULDER, N.; AARTS, E.; BANDELT, H.; LAARHOVEN, P.; PASCH, E. Genetic local search algorithms for the traveling salesman problem. Parallel Problem-solving from Nature Lecture Notes, **Computer Science** 496, p.109-116, 1991.

WEISS, S. I.; KULIKOWSKI, C. **Computer Systems that Learn: Classification e Prediction Methods from Statistics, Neural Networks, Machine Learning, and Expert Systems**, San Francisco, California, Morgan Kaufmann, 1991.

ANEXOS

ANEXO 1 – ALGORITMOS BASEADOS EM ARVORES DE DECISÃO E REGRAS, UTILIZADOS NAS COMPARAÇÕES (LIM;LOH;SHIH,1999)

CART – Versão do algoritmo CART (BREIMAN et al., 1984), tendo um critério especial de escolha de atributos. Dois métodos de poda foram testados.

Árvore S-Plus – variante do método CART escrito na linguagem S (BECKER; CHAMBERS; WILKS, 1988). É descrita em detalhes em CLARK e PREGIBON (1993).

C4.5 – foi usada a release 8 (QUINLAN, 1993;QUINLAN, 1996), com valores padrão, gerando árvores de decisão e usando o programa do pacote para a geração de conjuntos de regras.

FACT – é um algoritmo rápido de classificação, detalhado em LOH e VANICHSETAKUL (1988). Duas variantes foram testadas conforme o critério de escolha dos atributos.

QUEST – este algoritmo é descrito em LOH e SHIH (1997). Quatro variações diferentes foram testadas, conforme o critério de poda e escolha dos atributos. A versão usada foi a 1.7.10.

IND – este algoritmo é detalhado em BUNTINE (1992). A versão usada é a 2.1, com valores padrão. Quatro variações são analisadas.

OC1 – este algoritmo é detalhado em MURTHY, KASIF E SALZBERG (1994).

LMDT – detalhado em BRODLEY e UTGOFF (1995), usando os valores padrão do programa.

CAL5 – criado principalmente para valores numéricos, é detalhado em MULLER e WYSOTZKI (1994) e MULLER e WYSOTZKI (1997). Usou-se uma ferramenta do pacote para encontrar os valores ótimos.

TI – árvore de decisão baseada na escolha de apenas 1 atributo (HOLTE,1993).