OTAVIO LARSEN

CONSTRUÇÃO DE ATRIBUTOS X-OF-N USANDO ALGORITMOS GENÉTICOS

CURITIBA
2002

OTAVIO LARSEN

CONSTRUÇÃO DE ATRIBUTOS X-OF-N USANDO ALGORITMOS GENÉTICOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: Sistemas Inteligentes

Orientador: Júlio César Nievola
Co-Orientador: Alex Alves Freitas

CURITIBA
2002

Larsen, Otavio
Construção de Atributos X-of-N utilizando Algoritmos Genéticos. Curitiba, 2002. 50.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. Construção de Atributos 2. Atributos X-of-N 3. Mineração de Dados 4. Algoritmos Genéticos. I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t

“Um homem nunca sabe aquilo de que é capaz até que o tenta fazer”.

(Charles Dickens)

Agradecimentos

À minha mãe e irmão, pelos dois últimos anos, apertados, mas superados. Aos familiares que apoiaram e estiveram próximos nesses dois anos. À minha namorada, que esteve sempre ao meu lado. Aos professores, pelo apoio e auxílio no último ano, pela oportunidade e empenho na conquista da bolsa que permitiu que a conclusão deste desafio fosse possível. À Motorola, pelo apoio financeiro, e a oportunidade de realizar este estudo. E aos amigos, que tornaram essa experiência não só construtiva, como divertida.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AG ..... **Algoritmos Genéticos**
MER ..... **Modificação do Espaço de Representação**
PG ..... **Programação Genética**
Atr ..... **Atributo**
Val ..... **Valor**

Lista de Símbolos

$P(X|Y)$ ..... Probabilidade de X dado Y

A_i ..... Atributo i

MaxAtr ..... Quantidade de Atributos Existentes

V_{ij} ..... Valor j do Atributo i

$MaxAtrVal_i$ .....Quantidade de Valores Possíveis para o Atributo i

P(0) ..... População Inicial

GI(A) ..... Ganho de Informação do Atributo A

C .....Atributo Meta

I(C) .....Quantidade de Informação do Atributo Meta

I(C|A) .Quantidade de Informação do Atributo Meta dado o Valor do Atributo Previsor

TGI(A) .....Taxa de Ganho de Informação de A

Resumo

Este trabalho propõe um novo Algoritmo Genético (AG) para construção indutiva. O objetivo do AG é construir novos atributos X-of-N a partir dos atributos originais de uma dada base de dados, com o objetivo de melhorar a eficiência de um algoritmo de mineração de dados (ou aprendizado de máquina) aplicado àquela base de dados. O AG segue a abordagem de pré-processamento para construção indutiva, de forma que os novos atributos construídos pelo AG (juntamente com os atributos originais) possam ser repassados para qualquer algoritmo de mineração de dados. Neste projeto utilizou-se como algoritmo de mineração de dados o C4.5, que é um algoritmo de árvore de decisão bem conhecido. Para avaliar a qualidade dos novos atributos construídos pelo AG, comparamos a performance do C4.5 com e sem os novos atributos construídos pelo AG para várias bases de dados. Essa comparação é realizada com varias versões do AG. Além disso, o AG é comparado com outro método de construção de atributos proposto na literatura.

Palavras-Chave: Construção de Atributos, Atributos X-of-N, Mineração de Dados, Algoritmos Genéticos.

Abstract

We propose a new Genetic Algorithm (GA) for constructive induction. The goal of the GA is to construct new X-of-N attributes out of the original attributes of a given data set, in order to improve the effectiveness of a data mining (or machine learning) algorithm to be applied to that data set. The GA follows the preprocessing approach for constructive induction, so that the new attributes created by the GA can be given (together with the original attributes) to any data mining algorithm. In this project we use C4.5 as the data mining algorithm, which is a very well-known decision-tree induction algorithm. In order to evaluate the effectiveness of the new attributes constructed by the GA, we compare the performance of C4.5 with and without the new attributes constructed by the GA across several data sets. This comparison is performed with several versions of the GA. In addition, we compare the GA with another attribute construction method proposed in the literature.

Keywords: Constructive Induction, X-of-N Attribute, Data Mining, Genetic Algorithms.

1. Introdução

Mineração de dados auxilia usuários finais a obter informações úteis para seus negócios a partir de bancos de dados do mundo real. Um sistema de mineração de dados utiliza dados históricos para aprender, mas para isso é necessário que seja informado qual problema deve ser resolvido. A partir desse ponto, ele procura por padrões e relacionamentos nos dados que sejam úteis para resolver o problema do usuário.

A maior parte da tecnologia de mineração de dados existente hoje é relativamente nova à comunidade comercial. A pergunta que se segue à tradicional "O que é mineração de dados" é "Que tipo de tarefa deve ser realizada?". A resposta não é simples, pois existem vantagens e desvantagens entre as diferentes técnicas para cada tipo de tarefa. Uma das tarefas existente é a classificação.

Uma das mais estudadas, a classificação tem como objetivo classificar um item em uma determinada classe dentre várias classes pré-definidas. Ou seja, encontrar uma relação entre atributos e classes [Weiss & Kulikowski 1991] [Michie et Al. 1994] [Carvalho 1999].

A qualidade dos resultados obtidos por um algoritmo de classificação depende fortemente da qualidade dos atributos sendo minerados. Quando os atributos são de baixa qualidade, significando serem de baixa relevância para discriminar as classes, a precisão da classificação tende a ser baixa.

A fim de evitar tal problema, uma abordagem prática consiste em criar novos atributos além dos atributos originais. A abordagem aqui proposta consiste em um algoritmo genético para construção de atributos do tipo X-of-N. Esse tipo de atributo foi escolhido graças ao seu alto poder expressivo, como será visto posteriormente. A escolha do paradigma de algoritmos genéticos foi motivada pelo fato desses algoritmos lidarem bem com interação entre atributos [Freitas 2001b].

Não foi encontrando, na literatura disponível hoje, outro trabalho a desenvolver um AG para construção de atributos X-of-N.

Este trabalho se divide da seguinte forma. A revisão bibliográfica, na seção 2, abrange os principais aspectos de mineração de dados, construção de atributos e algoritmos genéticos. Na seção 3, é apresentado o projeto em si, descrevendo o algoritmo proposto. Na seção 4 encontram-se os resultados computacionais obtidos. Na seção 5, por fim, as conclusões e os trabalhos futuros.

2. Revisão Bibliográfica

2.1 Mineração de Dados

Mineração de dados tem como objetivo extrair informações úteis de bases de dados do mundo real. A partir de dados armazenados, tem-se como objetivo encontrar informações essenciais e úteis [Berson & Smith, 1997].

Métodos de mineração de dados procuram encontrar descrições lógicas ou matemáticas, idealmente precisas e simples, desenvolvendo basicamente operações de generalização e especialização a partir dos dados [Carvalho 1999].

Mineração de dados tem como metas principais a predição e descrição [Fayyad et al. 1996]. A predição utiliza-se dos dados contidos na base de dados para prever valores futuros de algumas variáveis de interesse, ou seja, consiste em criar modelos que permitam identificar acontecimentos futuros, não se atendo necessariamente à compreensão humana. A descrição, em contrapartida, tem como objetivo a descoberta de uma descrição dos padrões dos dados, descrições estas que sejam preferencialmente de fácil interpretação.

Pode-se alcançar as metas da mineração de dados através de diferentes tarefas, como estão a classificação, que tem como objetivo descobrir um relacionamento entre os atributos previsores e o atributo meta (classe), usando registros cuja classe é conhecida; o agrupamento, onde como o próprio nome demonstra, visa agrupar registros semelhantes em um mesmo grupo; e a associação, que tenta encontrar relação entre os atributos. A ênfase desse projeto é a classificação, vista com mais detalhes.

2.2 As tarefas de Classificação

Entre as tarefas de mineração de dados, uma das mais estudadas é classificação, conforme mencionado na introdução. Esta tarefa tem como objetivo classificar um objeto (ou registro) em uma determinada classe dentre várias classes pré-definidas, com base nos valores dos atributos daquele objeto. Um conjunto de dados de treinamento é utilizado para gerar um modelo de classificação. Após o treinamento, o modelo gerado é utilizado para classificar outros dados, ainda não vistos pelo algoritmo [Hand 1997], [Carvalho 1999]. Isso permite avaliar a capacidade de generalização do modelo, ou seja, sua capacidade para classificar corretamente dados de teste não acessados durante treinamento.

Como exemplo de classificação, considere a pequena base de dados mostrado na Tabela 2.1.

Tabela 2.1 – Dados de entrada para um sistema de classificação [Freitas 2001a]

Sexo	**País**	**Idade**	**Comprar (Meta)**
M	França	25	Sim
M	Inglaterra	21	Sim
F	França	23	Sim
F	Inglaterra	34	Sim
F	França	30	Não
M	Alemanha	21	Não
M	Alemanha	20	Não
F	Alemanha	18	Não
F	França	34	Não
M	França	55	Não

Verificando os dados, temos como objetivo identificar ou prever quem compraria e quem não compraria um determinado produto. Deve-se encontrar um subconjunto de atributos que possa ser usado para realizar essa previsão com um alto grau de acerto. Por exemplo, utilizando-se o atributo “sexo”, não encontraríamos regras úteis, pois os dois sexos têm a mesma probabilidade de comprar o produto. Mais precisamente, p(Comprar=sim|Sexo=M) = p(Comprar=sim|Sexo=F) = 2/5 = 0.4, onde p(X|Y) indica a probabilidade condicional de X dado Y. Ao avaliarmos o atributo “país”, já conseguimos encontrar relacionamentos úteis para prever o valor do atributo comprar, pois todos aqueles que eram ingleses adquiriram o produto, e aqueles

que eram alemães não. Ainda assim entre os franceses, houve alguns que compraram e outros que não compraram o produto. Dentre o grupo de franceses considerando-se um atributo adicional, a "idade", identificou-se que aqueles que eram maiores de 25 anos não compraram, e os que eram menores de ou tem 25 anos compraram o produto.

Assim, é possível gerar as seguintes regras de classificação:

- SE (País = Alemanha) ENTÃO (Comprar = Não)
- SE (País = Inglaterra) ENTÃO (Comprar = Sim)
- SE (País = França E Idade ≤ 25) ENTÃO (Comprar = Sim)
- SE (País = França E Idade > 25) ENTÃO (Comprar = Não)

Esse conjunto de regras auxilia então na tomada de decisões, sendo que através delas conclui-se que se encontraria um retorno mais rentável investindo na Inglaterra do que na Alemanha. Vale relembrar que as regras são avaliadas em dados de testes não vistos durante o treinamento.

Na classificação temos um problema assimétrico, pois um único atributo meta será previsto a partir dos demais atributos. É também difícil avaliar a qualidade de uma regra, dificultando a definição de quais regras um algoritmo de classificação deveria descobrir.

2.3 Construção Indutiva

Antes de se aplicar um algoritmo de classificação a uma base de dados, freqüentemente é necessário realizar algumas operações de pré-processamento ou preparação dos dados. Dentre algumas operações possíveis, encontramos a seleção e a construção de atributos. O enfoque deste projeto consiste na construção de atributos, também chamada de construção indutiva.

Para realizar com eficácia a tarefa de classificação, é necessário que os dados sejam descritos usando uma representação de dados (atributos) de boa qualidade. Quando os atributos existentes são de baixa qualidade, o resultado obtido será de baixa qualidade. A maioria dos algoritmos tradicionais de mineração de dados e aprendizado de máquina utilizam somente atributos já existentes na base de dados, o que torna esses métodos bastante vulneráveis a representações de dados de baixa qualidade.

Na maioria dos problemas reais, a relação entre o conjunto inicial de atributos e a classificação desejada é complexa. Ou seja, existem vários atributos disponíveis, mas pouco

conhecimento de qual será mais relevante, e não se sabe se um conjunto adequado de atributos está disponível.

Com o objetivo de resolver os problemas relacionados com a falta de qualidade dos atributos originais, foi proposta a idéia de construção indutiva por Michalski, em 1978, conforme citado em [Zheng 1995]. A idéia original consistia em adicionar à base atributos mais relevantes ao problema de classificação do que os originalmente fornecidos, visando uma melhora no processo de aprendizagem.

Várias pesquisas foram feitas, com o objetivo de reconstruir o espaço de representação inicial, tornando-o menos complexo. Como resultado, várias técnicas foram criadas, modificando a representação pela adição de novos atributos. Inicialmente, duas categorias foram criadas, de acordo com [Vafaie & De Jong 1998]. Na primeira, um especialista desenvolvia técnicas para transformar os atributos originais em um novo conjunto de atributos mais relevantes.

Como geralmente os especialistas não estão disponíveis, a pessoa responsável pelo desenvolvimento aplicava técnicas de tentativa-e-erro para encontrar um conjunto de atributos de boa qualidade, caracterizando uma segunda categoria de técnicas. Mas essas duas categorias de técnicas são baseadas em processos manuais, e portanto tendem a não ser muito eficientes.

Pesquisas estão sendo realizadas para a construção automática de atributos, sendo que os trabalhos nessa área podem ser categorizados de acordo com o tipo de atributos que eles constroem. Métodos de construção indutiva podem ser classificados de acordo com vários aspectos [Hu e Kibbler 1996]:

- Entrelaçamento vs Pré-processamento:

Em entrelaçamento, o processo de aprendizagem e o processo de construção de atributos estão interligados em um único algoritmo. A maioria dos algoritmos recaem nesta categoria, limitando a aplicabilidade do método construtivo. Separando-os, o método de construção pode ser usado como pré-processamento para qualquer algoritmo de aprendizagem. Por meio do pré-processamento, pode-se também avaliar a qualidade dos atributos aprendidos em uma gama maior de algoritmos de aprendizagem, pois é possível fornecer os atributos construídos para outros algoritmos de classificação.

- Guiado por Hipóteses (*Hypothesis-driven*) vs Guiado por Dados (*Data-driven*):

Um método guiado por hipóteses constrói novos atributos baseados nas hipóteses geradas anteriormente. Isso tem a vantagem de utilizar conhecimento previamente gerado pelo algoritmo, e a desvantagem de depender intensamente da qualidade desse conhecimento. Métodos guiados por dados não possuem o beneficio de utilizar conhecimento prévio, mas evitam uma forte dependência daquele conhecimento.

- Medidas Absolutas vs Medidas Relativas:

Medidas absolutas validam a qualidade de um atributo construído nos dados de treinamento sem se importar com como o atributo foi construído. Um exemplo deste tipo de medida é o Ganho de Informação. Medidas relativas validam um atributo construído com relação à qualidade dos atributos originais usados na construção. Ou seja, quando se utiliza uma medida relativa, um novo atributo construído é considerado de alta qualidade quando ele tem uma qualidade consideravelmente melhor que a qualidade dos atributos originais utilizados para a sua construção.

A maioria dos sistemas de construção indutiva constroem atributos que são conjunções ou disjunções dos atributos originais. Poucos sistemas constroem atributos usando outros tipos de representação, como por exemplo, M-of-N [Zheng 1998]. Representações M-of-N são normalmente interpretadas como "pelo menos M-de-Ns". Ou seja, pelo menos M condições das N disponíveis são verdadeiras. Por exemplo, considere o seguinte atributo 2-of-3: 2-of-{"Sexo = M", "Idade < 21", "Salário = alto"}. O valor desse atributo M-of-N (onde M=2 e N=3) e' verdadeiro para uma instancia (objeto) se ela tem pelo menos 2 dos 3 valores de atributo-valor acima.

Uma representação mais poderosa que a M-of-N é a X-of-N, que representa quantas das N condições são verdadeiras.

O valor de um atributo X-of-N para uma dada instância é o número de pares de atributo-valor da instância que são iguais a algum dos N pares atributo-valor do atributo X-of-N. Por exemplo, considere o seguinte atributo X-of-N: X-of-{"Sexo = M", "Idade < 21", "Salário = alto"}. Suponha que uma dada instância tenha os seguintes pares atributo-valor: {"Sexo = M",

"Idade = 51", "Salário = alto"}. Essa instância tem 2 dos 3 pares atributo-valor do atributo X-of-N, de modo que o valor do atributo X-of-N para esse exemplo é 2.

Note que um atributo X-of-N pode assumir qualquer número inteiro na faixa [0,N], enquanto um atributo M-of-N pode assumir apenas valores verdadeiros ou falsos. Portanto, a representação M-of-N é um caso particular da representação X-of-N.

As definições formais destas duas representações são descritas a seguir [Zheng 1998].

Seja $\{A_i \mid 1 \leq i \leq MaxAtr\}$ o conjunto de atributos, e para cada A_i, $\{V_{ij} \mid 1 \leq j \leq MaxAtrVal_i\}$ é o valor do atributo, onde MaxAtr é o número de atributos, e $MaxAtrVal_i$ é o número de diferentes valores de A_i.

Um atributo M-of-N é composto de um conjunto de pares atributo-valor e um número M, definido como: M-of-$\{AV_k \mid AV_k$ é um par atributo-valor representado por "$A_i = V_{ij}$", $1 \leq k \leq N_+$, $N \leq N_+$, $1 \leq N \leq MaxAtr$, $1 \leq M \leq N\}$. N_+ é o número de pares atributo-valor em uma representação, chamado de tamanho da representação, e N é o número de atributos diferentes que aparecem na representação. Um par atributo-valor AV_k ($A_i = V_{ij}$) é verdadeiro para um objeto se o atributo A_i do objeto tem o valor V_{ij}. O valor de uma representação M-of-N é verdadeiro ou falso. Dada um objeto, o valor do atributo M-of-N para aquele objeto é verdadeiro se pelo menos M das AV_k são verdadeiras, caso contrário é falso.

Um atributo X-of-N é composto por um conjunto de pares atributo-valor, definidos como: X-of-$\{AV_k \mid AV_k$ é um par atributo-valor representado como "$A_i = V_{ij}$", $1 \leq k \leq N_+$, $N \leq N_+$, $1 \leq N \leq MaxAtr\}$. O valor de uma representação X-of-N pode ser qualquer número entre 0 e N. Dado um objeto, seu valor para o atributo X-of-N é X se X das AV_k são verdadeiras.

A seguir descreve-se resumidamente alguns métodos de construção de atributos propostos na literatura.

2.3.1 Construção de atributos X-of-N

No método proposto neste trabalho, utiliza-se construção de atributos do tipo X-of-N como parte do pré-processamento para um algoritmo de mineração de dados. Um método proposto para construir esse tipo de atributo é descrito nessa seção.

[Zheng 1995] propôs o algoritmo XofN[1], onde uma árvore de decisão é gerada para construir um novo atributo utilizando atributos primitivos (originais) e os atributos previamente construídos. A construção do atributo X-of-N é feita através de uma busca gulosa ("*greedy search"*) no espaço de instâncias (objetos) definido pelos atributos primitivos.

O ponto inicial da busca é um atributo X-of-N vazio, ou seja, sem nenhum par atributo-valor. A cada passo da busca, aplica-se um entre dois operadores: adição de um possível par atributo-valor ou eliminação de um par atributo-valor.

Durante a busca, o algoritmo cria e mantém o melhor atributo X-of-N (que possua a maior taxa de ganho de informação) para cada tamanho de atributo X-of-N possível (isto é, cada número de pares atributo-valor possível), e finalmente, retorna o melhor encontrado.

A taxa de ganho de informação é utilizada para avaliar os atributos construídos. Para evitar atributos complexos que podem gerar um sobreajuste nos dados, na avaliação dos atributos construídos foi adicionado um custo de codificação. Esse custo considera a quantidade de pares atributo-valor existente no atributo X-of-N, ou seja, quanto maior o número de pares, maior o custo de codificação, e portanto menor a qualidade do atributo.

Há dois critérios de parada para o algoritmo X-of-N. Um critério se baseia em um tamanho máximo para o atributo construído. O algoritmo também termina se durante cinco passos seguidos não houver melhora na qualidade do melhor atributo X-of-N construído até o momento.

De acordo com a definição de atributo X-of-N, os atributos só podem ser construídos a partir de atributos binários ou nominais (categóricos). Para lidar com atributos primitivos contínuos, deve-se discretizá-los em uma etapa de pré-processamento. Esse pré-processamento também foi adotado para este projeto.

Zheng também apresenta um trabalho que verifica os efeitos em uma árvore de decisão ao acrescentar novos atributos construídos de quatro formas possíveis (conjuntiva, disjuntiva, M-of-N e X-of-N). Com certas opções pré-determinadas para os valores de alguns parâmetros do método de construção de atributos XofN, pode-se construir qualquer um desses quatro tipos possíveis de atributos [Zheng 1998]. Novamente, o método de busca aplicado para construir novos atributos é o método de busca guloso, que apesar de simples é amplamente utilizado.

[1] XofN é o nome do algoritmo proposto por Zheng, que constrói atributos do tipo X-of-N.

A estrutura de controle, o método de construção da árvore de decisão, a estratégia para construção de novos atributos e a função de avaliação são os mesmos do algoritmo XofN descritos anteriormente. Opta-se por qual tipo de atributo deseja-se construir. Se a opção foi o tipo X-of-N, então o funcionamento é exatamente o mesmo do algoritmo XofN original.

2.3.2 Construção de Atributos Entrelaçada com Algoritmo AQ.

Na metodologia apresentada por [Bloedorn & Michalski 1998] para a construção dos atributos, é usado um método indutivo de aprendizagem conhecido como AQ, com o objetivo de encontrar uma descrição generalizada dos exemplos.

Conforme mencionado anteriormente, o foco principal da construção indutiva é desenvolver métodos capazes de gerar hipóteses simples e precisas para tarefas de aprendizagem nas quais o espaço de representação é de baixa qualidade.

A seqüência de eventos do método se inicia com a entrada de uma base de dados de treinamento, com a caracterização do espaço de representação inicial, com a descrição dos atributos, os seus tipos de dados e domínios, sendo que estes são fornecidos pelo usuário.

A base de dados de treinamento é então dividida em uma sub-base de dados de treinamento propriamente dito e uma de validação. A sub-base de treinamento é repassada para o módulo de Geração de Regras de Decisão (regras de classificação), que gera descrições abrangentes na forma de regras para cobrir todo o espaço de busca. As regras são validadas avaliando-se sua precisão preditiva e sua complexidade nos dados de validação.

Com base nos resultados, o algoritmo opta ou pela parada, ou por repassar as regras para o módulo de Modificação do Espaço de Representação. Se as regras possuem qualidade suficiente, ou todas as modificações possíveis já foram feitas, as regras finais geradas são validadas nos dados de testes, a fim de determinar a precisão preditiva em dados separados dos dados de treinamento, medindo a capacidade de generalização das regras descobertas.

O módulo de Modificação do Espaço de Representação (MER) é responsável por determinar qual operador de modificação é aplicado em um dado estado da busca. O usuário pode escolher quais operadores de MER devem ser usados (para fim de geração de atributos, seleção de atributos e/ou eliminação de atributos), ou aceitar o *default* que usa todos os operadores. Quando todos os operadores estão presentes, eles são aplicados na seguinte ordem: seleção,

abstração e por fim geração de atributos. A seleção de atributos é realizada usando o ganho de informação [Quinlan, 1993] como um filtro para os atributos. Atributos com um ganho de informação menor que um limiar (*threshold*) são retirados do espaço de representação.

A abstração de atributos utiliza o algoritmo *ChiMerge* [Kerber 1992] para discretizar um atributo, dividindo os valores originais em vários intervalos (ou faixas) de valores, onde cada intervalo é então tratado como um valor discreto. Uma abordagem exaustiva de geração e teste de operadores de seleção, aplicando esses operadores aos dados, é executada pelo módulo de MER durante a geração de atributos, visto a variedade de operadores existentes.

O objetivo é construir novos atributos usando operadores que sejam fáceis de ser aplicados e tenham interpretação simples para o usuário. Os novos atributos devem ter uma capacidade de discriminação de classes maior que um limiar discriminatório mínimo e devem ter um custo computacional menor que um limiar de incremento de custo. O usuário pode limitar o número de novos atributos construídos que serão adicionados ao espaço de representação.

2.4 Algoritmos Genéticos

Algoritmos Genéticos (AGs) são métodos computacionais de busca baseados nos mecanismos de evolução natural e na genética. Em AGs uma população de indivíduos (possíveis soluções para o problema em questão) evolui de acordo com operadores probabilísticos concebidos a partir de metáforas biológicas, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua.

Em AGs, considera-se a Teoria da Evolução Darwiniana como um processo adaptativo de otimização, usando populações de estruturas computacionais que evoluem. Busca-se uma melhora na "qualidade" da população com respeito ao "ambiente", o qual representa um problema a ser resolvido.

Uma população inicial P(0) de indivíduos é gerada (na maioria das vezes, aleatoriamente), e calcula-se a adequabilidade (uma medida de qualidade) de cada um dos indivíduos. A função de adequabilidade consiste em determinar a eficácia com que um indivíduo (solução candidata) resolve um problema alvo, quando aplicado aos dados. Em seguida, indivíduos são selecionados para reprodução com probabilidade proporcional às suas adequabilidades. Uma vez que um conjunto de indivíduos foi selecionado, eles são usados para criar novos indivíduos através da

aplicação de operadores genéticos, geralmente de mutação e cruzamento. Esses operadores geralmente são aplicados de acordo com taxas de mutação e cruzamento especificadas pelo usuário. Um exemplo de pseudocódigo para um algoritmo genético pode ser como segue [Eshelman 2000]:

```
inicio
t=0;
inicialize P(t);
avalie indivíduos em P(t);
enquanto a condição de parada não for satisfeita
inicio
        t = t+1;
        selecione indivíduos de P(t-1);
                recombine (ou seja, realize cruzamento) e realize mutação nos indivíduos
                selecionados, e  adicione-os em P(t), substituindo os indivíduos
                anteriormente mantidos naquela população
        avalie indivíduos em P(t)
fim
fim
```

2.4.1 Métodos de Seleção

Para a seleção dos indivíduos que serão submetidos ao cruzamento, é possível optar entre alguns métodos, sendo que os dois que mais se destacam são o método da roleta e o torneio, por sua simplicidade de implementação. O método da roleta consiste em selecionar um indivíduo com uma probabilidade baseada na proporção de sua adequabilidade em relação ao total da soma das adequabilidades dos indivíduos [Goldberg 1989] [Grefenstette 2000]. Um exemplo do funcionamento da roleta é apresentado na Tabela 2.2 [Goldberg 1989]:

Tabela 2.2 – Dados para a execução da roleta [Goldberg 1989]

No.	**Indivíduo**	**Adequabilidade**	**% do total**
1	01101	169	14,4
2	11000	576	49,2
3	01000	64	5,5
4	10011	361	30,9

A roleta para os indivíduos da Tabela 2.2 é representada da forma ilustrada na Figura 2.2:

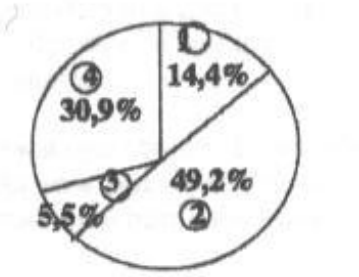


Figura 2.1 – Roleta [Goldberg 1989]

Gerando-se números aleatórios entre 0 e 100, selecionam-se os indivíduos que serão reproduzidos. Digamos, por exemplo, que os indivíduos selecionados foram:

- indivíduo número 1
- indivíduo número 2
- indivíduo número 2
- indivíduo número 4

Note que esse exemplo está consistente com o fato de o indivíduo 2 ter a maior probabilidade de ser selecionado (ocupando 49,2% da área da roleta na Figura 2.2).

Outro método é a seleção por torneio. Aplicando-se o torneio, um grupo de q indivíduos são obtidos aleatoriamente da população, com reposição (o indivíduo retorna à população para possivelmente participar em outros torneios) ou sem reposição (o indivíduo selecionado não pode participar novamente em outros torneios). Estes q indivíduos participam do torneio, onde o indivíduo com a melhor adequabilidade é escolhido para a reprodução, e o processo se repete até uma nova população ser gerada [Blickle 2000].

2.4.2 Cruzamento

Cruzamento consiste basicamente em misturar o material genético de dois indivíduos "pais" (obtidos através da aplicação de algum método de seleção), produzindo dois novos indivíduos "filhos" que herdam características dos pais.

Nesta seção, continuar-se-á usando o exemplo de execução de um algoritmo genético mencionado na seção anterior. Assim sendo, agora será realizado o cruzamento entre os indivíduos 1 e 2 e entre os indivíduos 2 e 4. Para realizar o cruzamento, pode-se optar por alguns métodos, como cruzamento de um ponto ou cruzamento uniforme. Segundo [Eshelman, Caruana e Schaffer 1989], o tradicional cruzamento de um ponto possui um forte *bias* posicional. Isso porque este método escolhe um ponto aleatório nos indivíduos selecionados para o cruzamento e troca todo o material genético à direita do ponto escolhido, como no exemplo da figura 2.3:

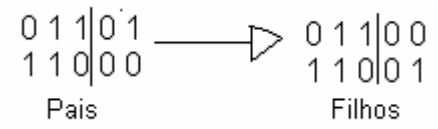


Figura 2.2 – Cruzamento de um ponto

Dessa forma, bits mais próximos tendem a ser trocados juntos com maior freqüência, mesmo que dois bits que se relacionem (do ponto de vista de solução do problema) estejam separados por uma grande quantidade de bits [Eshelman, Caruana e Schaffer 1989].

Para tentar eliminar essa limitação, [Syswerda 1989] desenvolveu o cruzamento uniforme. Este consiste na geração de uma máscara que define quando o bit será trocado e quando permanecerá o mesmo. Como exemplo, segue a figura abaixo:

Figura 2.3 – Cruzamento uniforme

Na figura acima, a máscara 1 0 0 1 1 representa quais bits devem ser trocados entre os pais e quais bits devem permanecer inalterados. Caso ocorra 1 na máscara, deve ocorrer a troca na posição correspondente, ou seja, o filho 1 recebe o bit do pai 2 e o filho 2 recebe o bit do pai 1. Caso ocorra 0 na máscara, o filho 1 recebe o bit do pai 1 e o filho 2 recebe o bit do pai 2 [Syswerda 1989]. Dessa forma, evita-se o problema do *bias* posicional, assumindo-se que a máscara é gerada aleatoriamente.

2.4.3 Mutação

A mutação é usada geralmente para manipular vetores de tamanho fixo [Bäck et al. 2000]. Seu funcionamento se dá da seguinte forma:

- determinar as posições em que se realizarão mutações, onde todas as posições possuem a mesma probabilidade de terem seus valores alterados, independente das alterações realizadas em outras posições;
- cria-se um novo vetor, substituindo-se os valores das posições desejadas por um novo valor, gerado aleatoriamente, dentro dos intervalos permitido para aquelas posições. No caso de indivíduos representados por *strings* binários, a mutação consiste simplesmente em inverter o valor de um bit.

2.4.4 *Niching*

Apesar dos algoritmos genéticos serem baseados nos princípios da evolução, diferentemente desta, em que se encontra uma diversidade de espécies, os algoritmos genéticos tendem a convergir rapidamente para uma única solução ótima ou aproximadamente ótima [Carvalho 2001]. Para tratar esta questão foram desenvolvidos métodos de *niching*. Alguns desses métodos são brevemente descritos a seguir, enfatizando-se o método de "*fitness sharing*", método utilizado neste trabalho.

Na pré-seleção, se a adequabilidade do filho for maior que a adequabilidade do pai com pior adequabilidade, o indivíduo gerado deve substituir o pai. É um método de implementação simples e custo computacional reduzido, mas implica em uma seleção antes da seleção real de indivíduos [Carvalho 2001].

No método de "*crowding*", nichos são criados quando indivíduos existentes são substituídos por outros similares. Seleciona-se uma certa quantidade de indivíduos e aquele que for mais semelhante ao novo indivíduo é eliminado, e o novo indivíduo incluído na população.

Já a "*fitness sharing*" é um mecanismo para escalonamento da adequabilidade, alterando somente o estágio de atribuição da adequabilidade em um algoritmo genético. "*Sharing*" pode ser aplicado em conjunto com outros mecanismos de escalonamento, mas deve ser o último, exatamente anterior à seleção [Mahfoud 2000].

Esse método verifica a quantidade de indivíduos em uma mesma região de adequabilidades próximas. Essa quantidade de indivíduos é utilizada para realizar uma depreciação na adequabilidade do indivíduo, uma vez que a adequabilidade de um indivíduo é dividida pela contagem do nicho. Esta contagem de nicho é feita através da soma da função de "*sharing*" entre ele e os outros indivíduos, incluindo o próprio.

A função de "*sharing*" calcula a distância entre dois indivíduos. Enquanto que na natureza os nichos são distinguidos entre si pelo fenótipo, em algoritmos genéticos o genótipo pode ser utilizado. A escolha depende do problema a ser resolvido. No "*sharing*" genotípico, calcula-se o número de bits (ou genes) que não são semelhantes entre os dois indivíduos, enquanto que na fenotípica a computação da distância utiliza o conhecimento específico do fenótipo, sendo mais comum a utilização da distância Euclidiana.

A fórmula para o cálculo da "*fitness sharing*" para um indivíduo *i* de uma população é a seguinte:

$$F'(i) = \frac{F(i)}{\sum_{j=1}^{\upsilon} sh(d(i,j))} \tag{2.1}$$

onde

$$sh(d) = \begin{cases} 1 - (d/\sigma_{share})^{\alpha} & \text{se } d < \sigma_{share} \\ 0 & \text{caso contrário} \end{cases} \tag{2.2}$$

sendo σ_{share} o limiar de dissimilaridade e *d* a distância (genotípica ou fenotípica) entre os indivíduos.

2.4.5 Elitismo

Enquanto a aplicação dos métodos de *niching* visam ampliar a diversidade de indivíduos, a prática do elitismo visa evitar a perda de uma boa solução. Uma vez que a população é recombinada geração após geração, uma boa solução encontrada pode ser perdida durante um cruzamento. Para contornar esse problema, aplica-se o procedimento do elitismo. Após a avaliação da adequabilidade dos indivíduos, o melhor entre eles é copiado sem alteração para a próxima geração. Assim garante-se que uma boa solução não seja perdida durante a evolução.

2.5 Programação Genética

Programação genética é um tipo de algoritmo evolucionário que se distingue em vários aspectos, como a representação do indivíduo, o estilo dos operadores genéticos e a função de adequabilidade. É considerada por alguns como uma especialização de algoritmos genéticos, quando esses aspectos são considerados separadamente, mas quando considerados conjuntamente, eles podem definir uma abordagem diferenciada [Kinnear, Jr 2000].

Um algoritmo de programação genética possui um conjunto de terminais e um conjunto de funções ou operadores. Terminais são geralmente entradas ou variáveis para o programa, enquanto que o conjunto de funções pode possuir operadores de vários tipos, tais como aritméticos ou lógicos. A figura 2.5 apresenta um indivíduo com operadores aritméticos e a figura 2.6 com operadores lógicos.

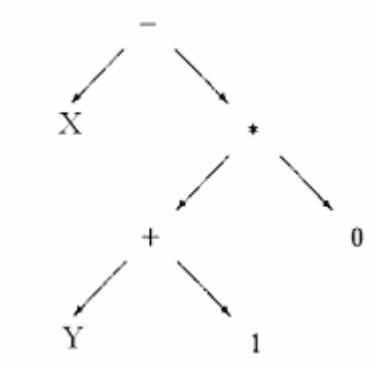


Figura 2.4 – Árvore da PG com Operadores Aritméticos

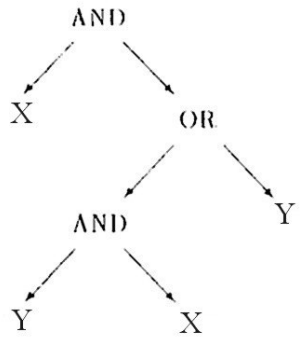


Figura 2.5 – Árvore da PG com Operadores Booleanos

2.6 AGs/ PG para Construção Indutiva

Seções anteriores descreveram construção de atributos e algoritmos evolucionários separadamente. Nesta seção esses dois tópicos são integrados, discutindo-se algoritmos evolucionários propostos para resolver o problema de construção de atributos.

A maioria dos métodos de construção de atributos utilizando abordagens evolucionárias trabalham com programação genética. [Kuscu 1999] utiliza a programação genética para gerar novos atributos utilizando os atributos originais e algumas funções primitivas, com o objetivo de descobrir um novo conjunto de atributos que facilite a solução do problema por ele proposto. Trabalhando em conjunto com uma rede neural do tipo *back-propagation*, o objetivo desses novos atributos é facilitar a tarefa de aprendizado da rede. A rede, por sua fez, define a qualidade dos subconjuntos de atributos gerados, computada através da performance da rede com esses atributos.

Em [Hu 1998], é proposto um algoritmo de programação genética que constrói novos atributos booleanos. Inicialmente o algoritmo transforma atributos originais em booleanos, gerando assim o conjunto terminal. O espaço de busca de um algoritmo de programação genética é definido pelo conjunto de terminais e pelo conjunto de funções. O conjunto de terminais possui os atributos primitivos, e o conjunto de funções podem ser funções aritméticas ou operadores relacionais. No algoritmo de programação genética utilizado por [Hu 1998], somente as operações AND e NOT foram utilizadas. Os indivíduos são formados por um atributo primitivo ou novos atributos, e a adequabilidade dos indivíduos é calculada levando em conta dois critérios.

Primeiro, uma medida de qualidade absoluta do atributo construído, independente da qualidade dos atributos originais usados nessa construção. Segundo, uma medida de qualidade relativa do atributo construído, a qual mede o quanto o atributo construído é melhor do que os atributos originais usados na sua construção. O autor aplica esse método como um pré-processamento dos dados, a fim de em seguida aplicar um algoritmo de mineração de dados.

[Vafaie e De Jong 1998] descreveram um método que automatiza o processo de construção e seleção de conjuntos de atributos úteis. Um ponto chave é a utilização de técnicas de computação evolucionária para realizar o objetivo proposto. Nesta metodologia, define-se que um conjunto inicial de atributos é fornecido como entrada. Apesar de se propor a construir e selecionar atributos, segundo os autores tentar executar essas tarefas em paralelo traz a tona problemas complexos de sincronização e representação. Permitindo que os processos atuem separadamente, evita-se esses problemas e consegue-se grande flexibilidade no controle do direcionamento da seleção ou construção dos atributos. O sistema é geral o suficiente para permitir uma seleção dinâmica dos módulos, dependendo do problema.

Tanto a construção quanto a seleção são baseados em algoritmos genéticos que utilizam uma função de avaliação como uma realimentação para guiar a busca. Neste caso, essa função consiste da performance de cada subconjunto de atributos selecionado no espaço de atributos modificado com os dados de treinamento. O melhor subconjunto é retornado para ser utilizado no estágio final da tarefa de classificação.

Para a Representação da Seleção de Atributos, o principal interesse é representar todo o espaço dos possíveis subconjuntos do conjunto de atributos dado. A representação dos atributos é feita através de um gene binário para cada atributo, representando a presença ou ausência deste. Logo, cada indivíduo consiste de uma string binária de tamanho fixo. A vantagem desta abordagem é a facilidade de implementação, bem como o fato de que os operadores dos algoritmos genéticos tradicionais podem ser aplicados.

Para a Representação da Construção de Atributos, os novos atributos podem ser descritos como expressões combinando atributos e operadores. O interesse é evoluir conjuntos de atributos que funcionam bem juntos, ou seja, um indivíduo neste caso possui um tamanho variável representando conjuntos de atributos, sendo que alguns podem ser do conjunto original de atributos e outros podem ser expressões. Relacionado à escolha da representação está a seleção de

formas úteis para operadores genéticos de cruzamento e mutação. Deve-se levar em consideração que existem dois níveis de representação: o conjunto de atributos construídos para solucionar os problemas e os próprios atributos originais. Logo, é necessário dois tipos de operadores de recombinação para melhor acomodar esta representação.

Ao nível de conjunto de atributos originais, o foco principal é avaliar a eficácia das diferentes combinações de atributos e ainda uma forma dos operadores tradicionais acomodarem tanto o aproveitamento (*exploitation*) como a exploração (*exploration*). Neste caso, o cruzamento pode simplesmente trocar segmentos dos pais limitados por genes (atributos), para criar os filhos. A mutação permanece inalterada, trocando um gene (atributo) por outro do conjunto inicial de atributos.

Ao nível dos atributos construídos, uma árvore representa cada atributo (ou gene). Precisa-se, então, de um novo operador genético para acomodar esta representação. O operador utilizado no sistema é o proposto por Koza para o paradigma de programação genética, que trabalha bem com indivíduos de tamanho variável. Nele, porções (sub-árvores) completas dos pais são trocadas. A aplicação de mutação em estruturas do tipo árvore é direta, e envolve selecionar aleatoriamente um nó da árvore (um operador ou atributo) para ser modificado.

A função de avaliação é dividida em duas partes. Após a seleção de um subconjunto de atributos, os dados de treinamento iniciais são alterados, adicionando-se os valores dos novos atributos que não estavam presentes nos vetores iniciais, e removendo aqueles não selecionados. O segundo passo é construir um classificador usando os dados de treinamento modificados.

A tabela 2.3 apresenta uma síntese dos principais pontos dos métodos comentados em mais detalhes nas Seções 2.3 e 2.6.

Tabela 2.3 – Comparativo dos métodos de Construção de Atributos

Método	Tarefa	Método de Busca	Abordagem	Observação
Construção de Atributos X-of-N [Zheng 1995]	Construção	Busca Gulosa (greedy search)	Entrelaçada	Trabalha em conjunto com árvore de decisão.
Construção Dirigida por Dados [Bloedorn e Michalski 1998]	Construção	Algoritmo de Indução de Regras	Entrelaçada	
Transformação do Espaço de Atributos [Vafaie e De Jong 1998]	Seleção Construção	Algoritmos Genéticos	Filtro Entrelaçada	Não trabalha com as duas tarefas simultaneamente.
Modelo Genético para Construção de Atributos [Kuscu 1999]	Construção	Programação Genética	Entrelaçada	Trabalha em conjunto com uma rede neural.
Abordagem de Programação Genética para Construção de Atributos [Hu 1998]	Construção	Programação Genética	Pré-processamento	Utiliza somente os operadores AND e NOT

Na tabela 2.3, a primeira coluna identifica qual método está representado com sua respectiva referência bibliográfica. A segunda coluna define o tipo de tarefa que o método realiza, entre construção, seleção ou ambos. A terceira coluna especifica o método de busca aplicado. A quarta indica a abordagem usada: *wrapper* ou filtro, para seleção de atributos; entrelaçada ou pré-processamento para construção de atributos. Por fim, a última coluna representa características específicas em cada um dos métodos. Percebe-se, por meio desta tabela, a predominância de métodos que seguem a abordagem *wrapper* ou entrelaçada, utilizando o algoritmo de classificação como método de avaliação, mesmo eles sendo considerados computacionalmente caros.

3. Método Proposto

Como já foi mencionado, este trabalho propõe um algoritmo genético (AG) para construção de atributos X-of-N. A motivação para construir esse tipo de atributo é que espera-se que ele possua um alto poder expressivo, e nos artigos encontrados na literatura apresentou um bom desempenho, gerando soluções de boa qualidade. Atributos X-of-N possuem também a capacidade de representar interação entre diferentes atributos, aspecto também abordado pelos algoritmos genéticos, influenciando assim a decisão de unir as duas abordagens. O AG segue uma abordagem de pré-processamento para construção de atributos. A figura 3.1 apresenta um pseudocódigo do algoritmo proposto:

```
popIncial = CriarPopulaçãoInicial();
//cada indivíduo tem o mínimo 2 e no máximo 7 pares atributo valor ativos
Execute(popInicial)
 Para k de 0 até número de gerações desejadas faça
   computeFitness(pop);
   adjustIndividuals(pop);
   doCrossOver = random.nextInt(100);
   doMutacao = random.nextInt(100);
FimPara

  computeFinalFitness(pop);
  adjustFinalIndividuals(pop);
   selectIndividuals;
}
```

Figura 3.1 – Pseudo-código

No pseudocódigo apresentado, temos a representação da criação do indivíduo, em um primeiro momento. Em seguida temos a idéia do AG em si, em que, para a quantidade de

gerações pré-definidas, calcula-se a adequabilidade, e após o cálculo os indivíduos passam por um ajuste, eliminado genes caso represente melhora na adequabilidade do indivíduo. No próximo passo, aplica-se os operadores genéticos de cruzamento e mutação. Ao fim da evolução, repete-se o processo de ajuste, e identifica-se os atributos construídos a serem adicionados na base.

3.1 Representação de um Indivíduo

Para esse algoritmo genético, ficou determinado que um indivíduo é um conjunto de pares <Atr, Val>, onde Val é um valor do domínio do atributo. Os atributos contínuos são discretizados em uma etapa de pré-processamento, antes da execução do AG. Essa discretização é realizada usando-se o C4.5-Disc [Kohavi e Sahami 1996]. Os atributos X-of-N construídos pelo AG possuem um tamanho variando de no mínimo 2 (dois) pares de <Atr, Val> e no máximo 7 (sete) pares de <Atr, Val>.

O valor mínimo de 2 pares é justificado pelo fato de que, se um individuo pudesse representar apenas um par de <Atr, Val>, ele não estaria representando um atributo construído, mas sim apenas uma condição envolvendo um dos atributos originais.

O valor máximo de 7 foi escolhido com base em experimentos psicológicos constituindo evidencia que esse é aproximadamente um limite na capacidade da memória de curto prazo em seres humanos [Miller 1956]. Assim, esse limite ajuda o atributo construído a ser mais compreensível para o usuário, o que é um ponto importante em mineração de dados.

A representação do genoma possui todos os atributos originais (isto é, cada gene corresponde a um dos atributos originais) e utiliza-se um *flag* ativo para indicar quais são os pares <Atr, Val> usados, como na figura 3.2. Se o *flag* ativo tem valor 1, o correspondente par <Atr,Val> é incluído no atributo X-of-N construído. Caso contrário (ou seja, se o *flag* ativo tem valor 0), o par <Atr, Val> não é incluído no atributo X-of-N.

Sempre que um individuo é gerado, seja na criação da população inicial, ou seja como resultado de um cruzamento ou mutação, a condição que o numero de genes ativos esteja entre 2 e 7 (inclusive) é forçada. Se o individuo possuir menos que 2 ou mais que 7 genes com valor do *flag* ativo igual a 1, alguns (tantos quanto necessário) *flags* ativos sofrem uma inversão em seus valores.

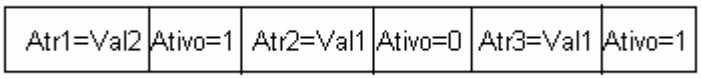


figura 3.2 – Representação do Indivíduo

3.2 Adequabilidade

Lembre-se que a função de adequabilidade avalia a qualidade da solução candidata representada pelo indivíduo. Neste algoritmo, a função de adequabilidade é definida como a taxa de ganho de informação do atributo construído, calculada como a seguir.

O ganho de informação de um atributo *A*, definido como GI(*A*), representa a diferença entre a quantidade de informação do atributo meta (classe) C, definido como I(*C*), e a quantidade de informação de C dados os valores do atributo A, representada por I(*C*|*A*). Matematicamente,

$$GI(A) = I(C) - I(C|A) , \tag{3.1}$$

onde

$$I(C) = -\sum_{j=1}^{n} p(C_j) \bullet \log p(C_j) \tag{3.2}$$

e

$$I(C \mid A) = \sum_{j=1}^{m} p(A_i) \bullet \left(-\sum_{j=1}^{n} p(C_j \mid A_j) \bullet \log p(C_j \mid A_j) \right) \tag{3.3}$$

onde p(C_j) é a probabilidade empírica da observação da *j*-ésima classe (isto é, o *j*-ésimo valor do atributo *C*), *n* é o número de classes, p(A_i) é a probabilidade empírica de observar o *i*-ésimo valor do atributo *A*, *m* é o número de valores que o atributo *A* pode assumir, e p(C_j|A_i) é a probabilidade empírica de observar a *j*-ésima classe com a condição de ter observado o *i*-ésimo valor do atributo *A*.

A taxa de ganho de informação do atributo *A*, representada por TGI(*A*), é calculada dividindo o ganho de informação de *A*, representado por GI(*A*), pela quantidade de informação do atributo *A*, representada por I(*A*), ou seja:

$$\mathrm{TGI}(A) = \mathrm{GI}(A) / \mathrm{I}(A)\ , \tag{3.4}$$

onde

$$I(A) = -\sum_{i=1}^{m} p(A_i) \bullet \log p(A_i) \tag{3.5}$$

TGI é a medida padrão para seleção de atributos no algoritmo C4.5 [Quinlan 1993]. Optou-se pela TGI(*A*) como medida de adequabilidade, ao invés do GI(*A*), porque o último possui um indesejado *bias* favorecendo atributos com uma grande quantidade de valores.

No C4.5 o critério de taxa de ganho de informação seleciona o atributo A que maximiza TGI(A), com a restrição que o ganho de informação GI(A) deve ser maior ou igual ao GI médio dos atributos examinados. Isso evita que um atributo com um valor muito pequeno de I(A) seja selecionado apesar de seu valor de GI(A) não ser muito alto (já que na formula de TGI(A) o termo I(A) ocorre no denominador), o que naturalmente seria indesejável. Essa restrição foi também incorporada na seleção por torneio e no elitismo do AG, como será discutido a seguir.

3.3 Seleção por Torneio

Após a validação da adequabilidade dos indivíduos, iniciam-se os procedimentos de evolução de um AG padrão. Um método de seleção por torneio foi criado, onde se deve definir a quantidade k de indivíduos que serão comparados para a seleção dos pais. Estes k escolhidos passam por uma pré-seleção, realizada da seguinte forma: através do seu ganho de informação (e não da taxa de ganho de informação), gera-se a média de ganho de informação entre os k indivíduos. Desses k indivíduos, aqueles que tiverem ganho de informação maior que a média passam para um vetor l. Entre os l escolhidos a partir da restrição, o que tiver maior *fitness sharing* vai ser o pai selecionado. No atual momento, k está definido como 5 (cinco), depois de testes realizados em bases selecionadas somente para definição de parâmetros (esses testes serão detalhados posteriormente).

3.4 *Niching: Fitness Sharing*

Como já foi levantado na seção 2.4.4, existe um problema de convergência prematura dos indivíduos, de forma divergente do que ocorre na natureza. Para evitar esse problema, foram desenvolvidos métodos de *niching*, para manter a diversidade da população.

Para a escolha do indivíduo selecionado para a reprodução, a adequabilidade de um indivíduo é modificada usando-se o método de *niching fitness sharing*, descrito anteriormente. Para cada indivíduo, avalia-se a semelhança genotípica entre o indivíduo selecionado e todos os demais. Calcula-se a função de *sharing*, a partir da soma das distâncias entre o indivíduo selecionado e cada um dos indivíduos da população, normaliza-se o valor encontrado (uma vez que os indivíduos possuem um tamanho fixo), e verifica-se se ele está dentro do limite definido para a semelhança (limiar de dissimilaridade). Em caso afirmativo, divide-se a adequabilidade normal do indivíduo pela função de *sharing* encontrada, retornando esse novo valor (adequabilidade*/sharing*) para o torneio.

Para o cálculo da similaridade genotípica, há vários casos a serem considerados. Primeiro suponha que os dois pares atributos-valor (um de cada indivíduo, e ambos referindo-se ao mesmo atributo) estejam ativos nos indivíduos cuja distância está sendo medida. Nesse caso, se os valores dos atributos forem iguais nos dois indivíduos, a distância é definida como 0 (zero). Caso os dois pares de atributo-valor estejam ativos, mas com valores diferentes, a distância é definida como 0,5 (meio). Em todos os outros casos, a distância é 1 (um). Como exemplo da computação de distância genotípica entre dois indivíduos considere os dois indivíduos mostrados na figura 3.3:

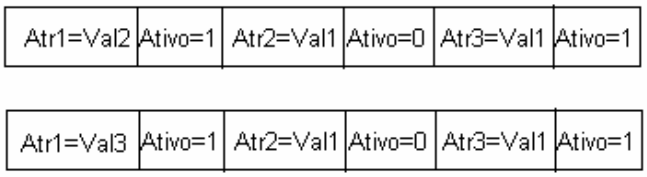


Figura 3.3 – Indivíduos para Cálculo da Distância Genotípica.

No exemplo acima, o atributo 1 está ativo nos dois indivíduos, mas os indivíduos possuem valores diferentes para aquele atributo. Logo, sua distância, com relação a esse atributo,

é de 0,5 (meio). O atributo 2 não está ativo em nenhum dos dois, então a distância entre os indivíduos, com relação a esse atributo, é de 1 (um). Já o atributo 3 está presente nos dois indivíduos, os quais possuem o mesmo valor para o atributo. Logo, a distância entre os atributos, com relação a esse atributo, é 0 (zero). Temos então uma distância total de 1,5 (um e meio), a qual normalizada (dividida pela quantidade de genes), seria 0,5 (meio). Esse seria o valor de *d*, empregado nas fórmulas da seção 2.4.

3.5 Elitismo

Durante a evolução das populações, a realização de um cruzamento ou mutação pode ocasionar a perda de um bom indivíduo, como descrito na seção 2.4.5. Para evitar que isso ocorra, aplica-se o elitismo.

O critério para seleção de atributos explicado na seção 3.3 também foi utilizado para definir o elitismo presente no AG, ou seja, o indivíduo (atributo construído) selecionado para o elitismo precisa ter seu ganho de informação GI(A) maior do que o GI médio dos indivíduos da população.

3.6 Cruzamento

O procedimento de cruzamento utilizado no AG proposto no trabalho foi o cruzamento uniforme, descrito na seção 2.4.

Essa escolha teve como objetivo evitar um forte *bias* posicional existente em outros tipos de cruzamento, tal como o cruzamento em um ponto, conforme explicado naquela seção. A probabilidade de cruzamento utilizada foi de 80%, um valor comum na literatura.

3.7 Mutação

Inicialmente o algoritmo define se deve ocorrer a mutação para cada um dos filhos gerados no cruzamento, utilizando-se uma probabilidade de mutação de 1%. Em caso positivo, escolhe-se aleatoriamente um gene e realiza-se a mutação, substituindo o valor do atributo em questão com um novo valor gerado aleatoriamente, dentre os valores pertencentes ao domínio do atributo.

3.8 Ajuste

Com o objetivo de fornecer ao algoritmo um conhecimento do problema a ser resolvido, desenvolveu-se um procedimento de ajuste de indivíduos representando atributos X-of-N.

Esse procedimento é iterativo. Em cada iteração, desativa-se um dos genes atualmente ativos por vez. Caso essa desativação melhore a adequabilidade do indivíduo, guarda-se a posição do gene cuja desativação resultou em melhora, e reativa-o. Parte-se para o próximo gene ativo, desativa-o e verifica sua adequabilidade. Dentre todos os genes que foram temporariamente desativados, o gene cuja desativação resultou em maior melhora em relação a adequabilidade original, será efetivamente desativo. Isso completa uma iteração. Repete-se esse procedimento na próxima iteração, e assim sucessivamente, parando-se o procedimento quando não houver melhora na adequabilidade ou quando for alcançado o limite de pelos menos dois genes ativos (já que esse deve ser o numero mínimo de genes ativos em um individuo para que ele represente efetivamente um novo atributo X-of-N, como mencionado anteriormente). O exemplo abaixo representa o funcionamento desse método. A figura 3.4 a seguir representa o indivíduo que está sendo analisado.

Atr1 = Val2	Ativo = 1	Atr2 = Val1	Ativo = 0	Atr3 = Val1	Ativo = 1	Atr4 = Val2	Ativo = 1	Adeq.	0,64

Figura 3.4 – Indivíduo

O indivíduo possui 3 atributos ativos, com uma adequabilidade de 0,64. Seguindo o método desenvolvido, desativa-se o primeiro atributo e reavalia-se sua adequabilidade, encontrando uma nova adequabilidade de 0,7 (figura 3.5).

Atr1 = Val2	**Ativo = 0**	Atr2 = Val1	Ativo = 0	Atr3 = Val1	Ativo = 1	Atr4 = Val2	Ativo = 1	Adeq.	0,7
Posição:	1	Adeq.	0,7						

Figura 3.5 – Desativação do primeiro atributo

Como a nova adequabilidade é maior que a do atributo original, guarda-se a posição do atributo e a adequabilidade com o atributo desativado. Reativa-se então o atributo, e verifica-se o próximo atributo (Figura 3.6).

Atr1 = Val2	Ativo = 1	**Atr2 = Val1**	**Ativo = 0**	Atr3 = Val1	Ativo = 1	Atr4 = Val2	Ativo = 1	Adeq.	0,64
Posição:	1	Adeq.	0,7						

Figura 3.6 – Análise segundo atributo

Como o atributo já está desativado, verifca-se o próximo. Sendo um atributo ativo, ele é desativado e a adequabilidade do indivíduo é recalculada, sendo de 0,53 (Figura 3.7).

Atr1 = Val2	Ativo = 1	Atr2 = Val1	Ativo = 0	**Atr3 = Val1**	**Ativo = 0**	Atr4 = Val2	Ativo = 1	Adeq.	0,53
Posição:	1	Adeq.	0,7						

Figura 3.7 – Desativação do terceiro atributo

Como a nova adequabilidade é menor que a adequabilidade do indivíduo original, avalia-se o próximo atributo. É um atributo ativo, que quando desativado, o indivíduo retorna uma adequabilidade de 0,77 (Figura 3.8).

Atr1 = Val2	Ativo = 1	Atr2 = Val1	Ativo = 0	Atr3 = Val1	Ativo = 1	**Atr4 = Val2**	**Ativo = 0**	Adeq.	0,77
Posição:	4	Adeq.	0,77						

Figura 3.8 – Desativação do quarto atributo

A nova adequabilidade é maior que a adequabilidade original, e também maior que a adequabilidade encontrada com o primeiro atributo desativado. Como não existem mais atributos, e ocorre uma melhora com algum atributo desativado, efetua-se a desativação gerando o novo indivíduo (Figura 3.9).

Atr1 = Val2	Ativo = 1	Atr2 = Val1	Ativo = 0	Atr3 = Val1	Ativo = 1	Atr4 = Val2	Ativo = 0	Adeq.	0,77

Figura 3.9 – Indivíduo após procedimento

Para esse exemplo, alcançou-se a restrição de no mínimo dois atributos ativos. Caso essa restrição não tivesse ocorrido, o processo se repetiria, até a ocorrência da restrição, ou a desativação dos atributos não apresentar melhora na adequabilidade.

3.9 Seleção de Atributos Construídos a serem Adicionados na Base de Dados

Na fase inicial da pesquisa, somente o melhor atributo construído durante toda a execução do AG era adicionado à base de dados, a qual é então passada para o algoritmo de mineração de dados alvo. Depois de estudos realizados durante essa pesquisa, definiu-se um procedimento para a inclusão, na base de dados, de mais de um atributo construído durante a execução do AG. Esse procedimento, desenvolvido a partir das informações obtidas pela pesquisa de uma forma mais justa de selecionar mais de um indivíduo, funciona da seguinte forma.

Para cada indivíduo, computa-se uma matriz de contingência com duas dimensões, onde a primeira dimensão consiste nos possíveis valores de "X" do atributo X-of-N, e a outra dimensão consiste nas possíveis classes (valores do atributo meta). Cada célula (i,j) dessa matriz contem o numero de exemplos de treinamento satisfazendo o i-ésimo valor de X e a j-ésima classe. Essa matriz, que é também utilizada para o cálculo da adequabilidade do indivíduo, auxilia na determinação da cobertura dos registros da base de treinamento. A tabela 3.1 mostra um exemplo da matriz de contingência.

Tabela 3.1 – Exemplo de Matriz de Contingência

X	Classe A	Classe B	Total
0	50	39	89
1	25	13	38
2	47	21	68
3	4	11	15

O processo de seleção de atributos construídos a serem adicionados na base de dados começa considerando o individuo de maior adequabilidade na população final, e prossegue considerando os indivíduos da população final em ordem decrescente de adequabilidade. Suponha que o individuo de melhor adequabilidade da população final tenha a matriz de contingência mostrada na Tabela 3.1.

Com base nessa matriz, podemos dizer que o indivíduo cobre 89 exemplos dos dados de treinamentos com 0 (zero) de N atributos (o valor do X, no indivíduo X-of-N, é zero). Desses 89 exemplos, 50 tem classe A e 39 classe B. Assim, atribuindo-se a classe da maioria aos exemplos

cobertos pela condição X-of-N = 0, há 50 exemplos corretamente classificados por essa condição. Marca-se, então, esses cinqüenta exemplos como corretamente cobertos, e passamos para o valor seguinte do atributo X-of-N. Verificamos que a condição X-of-N = 1(um) cobre 25 exemplos da classe A e 13 da classe B. Marcamos, então, os 25 da Classe A como corretamente cobertos, e assim sucessivamente, sempre marcando como corretamente cobertos os exemplos pertencentes à classe da maioria.

Se o indivíduo não cobriu corretamente todos os exemplos da base de treinamento, realiza-se o mesmo procedimento para o próximo indivíduo (isto é, o segundo melhor indivíduo da população final). Cabe lembrar que os indivíduos selecionados para o processo são os de melhores adequabilidades, baseado na restrição que seu ganho de informação tem que ser maior que o ganho de informação médio de todos os indivíduos da população final. (Esse é o mesmo critério usado para seleção de indivíduos para fins de elitismo). Esse processo é repetido iterativamente até que todos os exemplos da base de treinamento tenham sido marcados como corretamente cobertos, ou até que todos indivíduos da população final tenham sido considerados.

Note que, como resultado desse procedimento, o número de atributos X-of-N construídos pelo AG selecionados para serem adicionados à base de treinamento é variável, adaptando-se à base de treinamento.

3.10 Parâmetros do AG

Para o AG, foram definidos os seguintes valores de parâmetros:

- Número de Indivíduos: 101[2];
- Número de Gerações: 100;
- Número de Indivíduos no Torneio: 5;
- Probabilidade de cruzamento para os Indivíduos: 80%;
- Probabilidade de troca para os genes na execução do cruzamento uniforme: 50%;
- Probabilidade de Mutação para os Indivíduos: 1%;
- *Teta-share* (*threshold* de semelhança entre indivíduos) do *fitness sharing*: 0,1;

[2] Optou-se por 101 indivíduos a fim de manter a paridade no cruzamento, visto a aplicação de elitismo.

Os valores dos parâmetros foram definidos através de uma série de testes utilizando-se bases específicas do repositório da UCI (University of California at Irvine). As bases utilizadas para teste foram: *Bridges*, *CRX*, *Pimia Indian Diabetes*, *Post Operative* e *Voting Records*.

Para estas bases, variou-se os valores do *teta-share* e do número dos indivíduos do torneio. Para o primeiro, o algoritmo genético com *niching* foi executado variando-se seu valor entre 0,1 e 07, inclusive. Para o segundo, os testes variam entre 3, 5 e 7 indivíduos para o torneio. Ao comparar os resultados, a configuração de melhor qualidade foi escolhida para os valores *default* do algoritmo, quando aplicado nas outras bases. Para os demais, definiu-se os valores padrões da literatura.

Caber ressaltar que essas bases foram usadas exclusivamente para ajuste de parâmetros do AG, tentando encontrar valores destes que fossem robustos para várias bases de dados, constituindo valores "*default*" de parâmetros para o AG – da mesma forma que C4.5 e outros algoritmos de mineração de dados tem valores pré-definidos de seus parâmetros.

Portanto, as bases acima não foram utilizadas nos experimentos relatados na seção 4, nos quais tanto o AG como C4.5 são executados com seus valores *default* de parâmetro, para a comparação entre eles ser o mais justa possível.

4. Resultados obtidos

Com o objetivo de verificar a qualidade dos novos atributos construídos pelo algoritmo genético, comparou-se a performance do C4.5 utilizando-se somente os atributos originais com a performance do C4.5 com os atributos originais acrescidos dos atributos construídos. De agora em diante refere-se à base de dados sem atributos construídos como base de dados original e a base de dados com os atributos construídos como base de dados estendida. C4.5 é um método para gerar árvores de decisão, descrito em [Quinlan 1993].

Em ambos os casos a performance do C4.5 foi medida em relação à taxa de erro de classificação. Na maioria dos experimentos[3], a taxa de erro foi medida utilizando-se validação cruzada de fator 10 [Hand 1997], que funciona como descrito a seguir. A base de dados é particionada em 10 subconjuntos mutualmente exclusivos. Então, o algoritmo de classificação é executado 10 vezes. Na i-ésima execução, i = 1,...,10, a i-ésima partição é usada como conjunto de teste, e as outras nove são agrupadas e utilizadas como o conjunto de treinamento. Após as 10 execuções, o erro relatado é o erro médio entre todos os erros nos dados de testes nas 10 execuções.

Os experimentos foram realizados utilizando-se bases de dados de domínio público do repositório da UCI (University of California at Irvine)[4], e uma base de dados real, conseguida junto a Motorola, que forneceu apoio técnico.

As 21 bases de dados utilizadas são apresentadas na tabela 4.1, que mostra o número de exemplos, número de atributos originais e o número de classes para cada bases de dados.

[3] As exceções estão mencionadas mais abaixo.
[4] Este repositório de dados pode ser encontrado na página http://www.ics.uci.edu/~mlearn/MLRepository.html

Tabela 4.1 - Características principais das bases de dados utilizados nos experimentos

Base de Dados	**No. de exemplos**	**No. De atributos**	**No. De classes**
Audiology	226	69	24
Balance-Scale	625	4	3
Car	1717	5	4
CMC	1473	9	2
Dermatology	366	33	6
Diabetes	768	8	2
Glass	214	9	7
Heart-Disease	303	13	5
Hepatitis	155	19	2
Iris	150	4	3
Liver Disorder	345	6	2
Lung-Cancer	32	56	3
Monks-1	432	7	2
Monks-2	432	7	2
Monks-3	432	7	2
Motorola	284	377	2
Promoters	106	57	2
Soybean	307	35	19
Tic-tac-toe	958	9	2
Veh	846	17	4
Wisconsin breast cancer	699	10	2

Os resultados dos experimentos estão apresentados nas tabelas 4.2 a 4.6, as quais serão mostradas nas próximas seções. Essas tabelas têm a mesma estrutura, consistindo de 4 colunas. A segunda coluna mostra a taxa de erro do C4.5 usando apenas os atributos originais. A terceira coluna mostra a taxa de erro do C4.5 usando não apenas os atributos originais, mas também os atributos construídos pelo algoritmo X-of-N proposto por Zheng, descrito na seção 2.3.1, com algumas modificações descritas a seguir. A quarta coluna mostra a taxa de erro do C4.5 usando não apenas os atributos originais, mas também os atributos X-of-N construídos pelo algoritmo genético proposto neste trabalho.

O método proposto por Zheng sofreu adaptações para a comparação com o algoritmo proposto ser considerada justa. Diferente do método descrito na seção 2.3.1, aqui ele foi utilizado como pré-processamento. Como no método proposto, onde ele constrói um indivíduo para cada

quantidade possível de pares atributo-valor, mas o método implementado de Zheng sofre a restrição de construir entre o mínimo de dois atributos e o máximo de sete, como o algoritmo proposto. Também de forma contrário ao original, o custo de codificação do indivíduo não foi levado em conta.

Os valores após o símbolo "±" indicam os desvios padrões. Para cada base de dados foi avaliado se as taxas de erro das terceira e quarta colunas são significativamente diferentes da taxa de erro da segunda coluna, ou seja, considerou-se os resultados da segunda coluna (obtidos usando apenas os atributos originais) como uma base de comparação. Uma diferença de taxa de erro é considerada significante quando os dois intervalos de taxa de erro (levando em conta os desvios padrões) não se sobrepõem. Quando a taxa de erro da terceira e da quarta coluna é significantemente melhor que a taxa de erro da segunda coluna (atributos "originais"), um sinal de "(+)" está presente na terceira ou quarta coluna, indicando que o(s) atributo(s) construído(s) pelo respectivo algoritmo melhoraram significantemente a classificação executada pelo C4.5. Caso contrário, quando a taxa de erro da terceira ou quarta coluna for significantemente pior do que a da segunda coluna, um sinal de "(-)" se encontra na terceira coluna ou quarta coluna, significando que o atributo construído piorou a performance de classificação do C4.5.

Foi também realizada uma comparação direta entre os resultados da terceira e quarta coluna. Assim, um sinal de "(*)" na quarta coluna significa que a taxa de erro naquela coluna é significantemente melhor que a taxa de erro na terceira coluna, ou seja, o uso de atributos construídos pelo AG levou a resultados significativamente melhores que o uso dos atributos construídos pelo método de Zheng. Por outro lado, um sinal ("/") na quarta coluna indica que a taxa de erro naquela coluna é significativamente pior do que a taxa de erro na terceira coluna, ou seja, o uso de atributos construídos pelo AG levou a resultados significativamente piores que o uso de atributos construídos pelo método proposto por Zheng.

Para gerar os resultados, foram realizadas uma série de experimentos com diferentes configurações do algoritmo genético. A seguir os resultados de cada um desses experimentos será relatado em seções separadas, a saber:

Seção 4.1: resultados para o algoritmo genético "puro", sem nenhuma das extensões usadas nos outros experimentos;

Seção 4.2: resultados para o algoritmo genético estendido com *fitness sharing*;

Seção 4.3: resultados para o algoritmo genético com ajuste dos atributos construídos aplicado somente aos indivíduos da população final;

Seção 4.4: resultados para o algoritmo genético com ajuste dos atributos construídos aplicado a todos os indivíduos criados durante a evolução do algoritmo;

Seção 4.5: resultados para o algoritmo genético com *fitness sharing* e retornando, como solução, vários atributos X-of-N construídos.

4.1 Resultados para Algoritmo Genético Puro:

Observando a ocorrência dos sinais "(+)" e "(-)" na quarta coluna da tabela 4.2, podemos observar que os atributos construídos pelo algoritmo genético melhoram significantemente a performance do C4.5 em quatro bases de dados (Balance-scale, Monks-2, Motorola e Tic-tac-toe), e pioram significantemente em outros cinco bases de dados (CMC, Diabetes, Liver Disorder, Monks-3 e Veh). Em relação ao método proposto por Zheng, observando a ocorrência dos sinais "(*)" e "(/)" na quarta coluna, o algoritmo genético perde significativamente em três bases de dados (Diabetes, Liver Disorder e Veh) e ganha significativamente em outros dois (Iris e Monks-2). Nas outras bases de dados, as diferenças na taxa de erro não foram significantes. A figura 4.1 representa o esquema de execução do algoritmo.

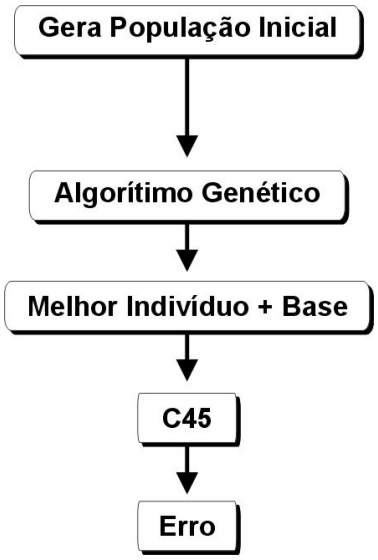


Figura 4.1 – Esquema da execução do algoritmo genético puro

Tabela 4.2: Algoritmo Genético Puro

Base de Dados	**Taxa de Erro (%)**		
	Atributos Originais	**Atributos originais + Zheng**	**Atributos originais + AG**
Audiology	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03
Balance-Scale	22,72 ± 0,81	20,75 ± 1,49	**19,48 ± 1,44 (+)**
Car	2,84 ± 0,26	2,84 ± 0,26	2,90 ± 0,31
CMC	49.12 ± 1.26	51.23 ± 1.71	53,55 ± 2,02 (-)
Dermatology	3,92 ± 0,79	4,53 ± 0,93	3,33 ± 0,85
Diabetes	27,32 ± 1,13	25,94 ± 1,86	30,60 ± 2,05 (-)(/)
Glass	2,36 ± 0.79	2,36 ± 0.79	2,36 ± 0.79
Heart-Disease	44,65 ± 2,46	44,00 ± 2,21	40,91 ± 1,86
Hepatitis	22,84 ± 1,83	20,49 ± 2,28	22,34 ± 5,26
Iris	6,01 ± 2,52	36,65 ± 1,80 (-)	**6,01 ± 2,52 (*)**
Liver Disorder	32,98 ± 3,10	36,39 ± 2,45	56,38 ± 2,46 (-)(/)
Lung Cancer	52,14 ± 10,76	64,29 ± 12,82	52,86 ± 10,92
Monks-1	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Monks-2	29,60 ± 0,04	24,50 ± 0,03	**0,00 ± 0,00 (+) (*)**
Monks-3	0,00 ± 0,00	2,80 ± 0,01 (–)	2,80 ± 0,01 (–)
Motorola	4,92±0,54	4,19±1,01	**2,79±0,66(+)**
Promoters	18,88 ± 2,19	**13,25 ± 1,49 (+)**	15,00 ± 4,28
Tic-tac-toe	14,31 ± 1,14	**5,98 ± 0,69 (+)**	**5,87 ± 0,67 (+)**
Veh	30,60 ± 1,18	32,79 ± 1,18	54,84 ± 3,29 (-)(/)
Wisconsin breast cancer	4,57 ± 0,64	4,87 ± 0,80	5,69 ± 0,88
Ganha			
C45		2	4
Zheng			2
Perde			
C45		2	5

4.2 Resultados para Algoritmo Genético com *Fitness Sharing*:

Utilizando a implementação de *fitness sharing* (Figura 4.2), descrita na seção 3.5, obteve-se os resultados mostrados na tabela 4.3. Observando a ocorrência dos sinais "(+)" e "(-)" na quarta coluna, podemos observar uma melhora significativa na performance do C4.5, com os novos atributos construídos pelo algoritmo genético, em cinco bases de dados (Balance Scale,

Monks-2, Motorola, Promoters e Tic-tac-toe). Por outro lado, o algoritmo genético perde significativamente em outros seis bases de dados (CMC, Diabetes, Hepatitis, Liver Disorder, Monks-3 e Veh). Comparando-se os resultados do algoritmo genético com os resultados do método proposto por Zheng, os atributos construídos pelo primeiro obtiveram resultados significativamente melhores que os atributos construídos pelo segundo em três bases de dados (Iris, Lung Câncer e Monks-2), enquanto o inverso ocorreu em cinco bases de dados (CMC, Diabetes, Hepatitis, Liver Disorder e Veh). Nas outras bases, as diferenças nas taxas de erro não foram significantes.

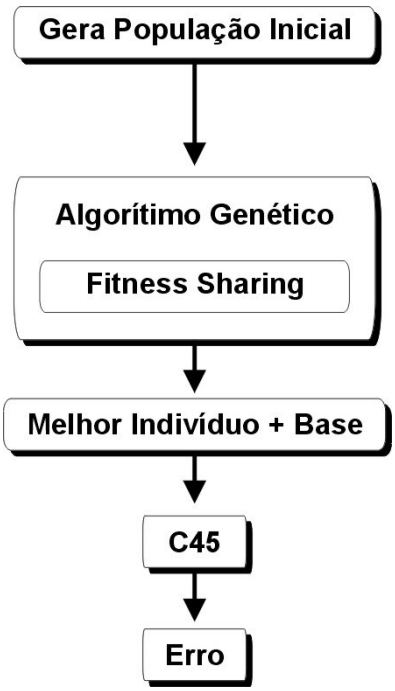


Figura 4.2 – Esquema da execução do algoritmo com *fitness sharing*

Tabela 4.3: Algoritmo Genético com *Fitness Sharing*

Base de Dados	**Taxa de Erro (%)**		
	Atributos Originais	**Atrib. Originais + Zheng**	**Atributos originais + AG**
Audiology	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03
Balance-Scale	22,72 ± 0,81	20,75 ± 1,49	**18,53 ± 1,44 (+)**
Car	2,84 ± 0,26	2,84 ± 0,26	2,90 ± 0,31
CMC	49,12 ± 1,26	51,23 ± 1,71	59,87 ± 1,71 (-)(/)
Dermatology	3,92 ± 0,79	4,53 ± 0,93	3,63 ± 0,76
Diabetes	27,32 ± 1,13	25,94 ± 1,86	32,23 ± 2,64 (-)(/)
Glass	2,36 ± 0.79	2,36 ± 0.79	2,36 ± 0.79
Heart-Disease	44,65 ± 2,46	44,00 ± 2,21	45,13 ± 2,72
Hepatitis	17,34 ± 2,04	20,49 ± 2,28	28,84 ± 4,72 (-)(/)
Iris	6,01 ± 2,52	36,65 ± 1,80 (-)	**6,01 ± 2,52 (*)**
Liver Disorder	32,98 ± 3,10	36,39 ± 2,45	51,97 ± 4,14 (-)(/)
Lung Cancer	52,14 ± 10,76	64,29 ± 12,82	**37,86 ± 11,47 (*)**
Monks-1	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Monks-2	29,60 ± 0,04	24,50 ± 0,03	**0,00 ± 0,00 (+) (*)**
Monks-3	0,00 ± 0,00	2,80 ± 0,01 (–)	2,80 ± 0,01 (–)
Motorola	4,92±0,54	4,19±1,01	**2,51±0,76(+)**
Promoters	18,88 ± 2,19	**13,25 ± 1,49 (+)**	**13,25 ± 1,98 (+)**
Tic-tac-toe	14,31 ± 1,14	**5,98 ± 0,69 (+)**	**5,34 ± 0,47 (+)**
Veh	30,60 ± 1,18	32,79 ± 1,18	53,82 ± 2,98(-)(/)
Wisconsin breast cancer	4,57 ± 0,64	4,87 ± 0,80	4,83 ± 0,78
Ganha			
C45		2	5
Zheng			3
Perde			
C45		2	6
Zheng			3

4.3 Resultados para Algoritmo Genético com *Fitness Sharing* e Ajuste dos Atributos Construídos Aplicado Somente aos Indivíduos da População Final

Nessa configuração do algoritmo genético, realizou-se um ajuste, uma forma de "poda" de pares atributo-valor, nos indivíduos, como descrito na seção 3.8. Nos experimentos cujos

resultados são relatados nesta seção, temos a aplicação da *fitness sharing* com esse ajuste aplicado somente à população final de indivíduos (Figura 4.3), obtendo-se os resultados mostrados na tabela 4.4. A motivação para aplicar esse ajuste apenas aos indivíduos da população final é uma redução no tempo computacional, em comparação com a aplicação desse ajuste a todos os indivíduos de todas as gerações.

Analisando-se os resultados da Tabela 4.4 de acordo com os critérios estabelecidos anteriormente, verifica-se um ganho na performance do C4.5, ao se utilizar os atributos construídos pelo algoritmo genético, em quatro bases de dados (Balance-Scale, Monks-2, Motorola e Tic-tac-toe) e piora em seis bases de dados (CMC, Diabetes, Hepatitis, Liver Disorder, Monks-3 e Veh). Comparando-se os resultados do algoritmo genético com os resultados do método proposto por Zheng, o primeiro apresenta melhor performance em duas bases de dados (Iris e Monks-2) e pior performance em quatro bases de dados (CMC, Diabetes, Liver Disorder e Veh).

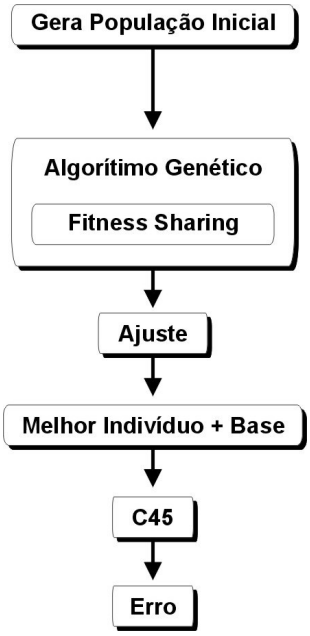


Figura 4.3 – Esquema da execução com ajuste da população final

Tabela 4.4: Algoritmo Genético com *Fitness* Sharing e Ajuste dos Atributos Construídos aplicado somente aos Indivíduos da População Final

Base de Dados	**Taxa de Erro (%)**		
	Atributos Originais	**Atrib. Originais + Zheng**	**Atributos originais + AG**
Audiology	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03
Balance-Scale	22,72 ± 0,81	20,75 ± 1,49	**18,85 ± 1,44 (+)**
Car	2,84 ± 0,26	2,84 ± 0,26	2,84 ± 0,26
CMC	49,12 ± 1,26	51,23 ± 1,71	55,10 ± 2,29 (-)(/)
Dermatology	3,92 ± 0,79	4,53 ± 0,93	3,16 ± 0,80
Diabetes	27,32 ± 1,13	25,94 ± 1,86	39,08 ± 4,96 (-)(/)
Glass	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79
Heart-Disease	44,65 ± 2,46	44,00 ± 2,21	42,43 ± 1,75
Hepatitis	17,34 ± 2,04	20,49 ± 2,28	20,67 ± 1,20 (-)
Iris	6,01 ± 2,52	36,65 ± 1,80 (-)	**9,34 ± 5,27 (*)**
Liver Disorder	32,98 ± 3,10	36,39 ± 2,45	54,49 ± 1,82 (-)(/)
Lung Cancer	52,14 ± 10,76	64,29 ± 12,82	63,57 ± 14,69
Monks-1	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Monks-2	29,60 ± 0,04	24,50 ± 0,03	**0,00 ± 0,00 (+) (*)**
Monks-3	0,00 ± 0,00	2,80 ± 0,01 (–)	2,80 ± 0,01 (–)
Motorola	4,92±0,54	4,19±1,01	**2,46±0,76 (+)**
Promoters	18,88 ± 2,19	**13,25 ± 1,49 (+)**	17,88 ± 4,16
Tic-tac-toe	14,31 ± 1,14	**5,98 ± 0,69 (+)**	**5,56 ± 0,72 (+)**
Veh	30,60 ± 1,18	32,79 ± 1,18	46,36 ± 3,47(-)(/)
Wisconsin breast cancer	4,57 ± 0,64	4,87 ± 0,80	5,45 ± 0,90
Ganha			
C45		2	4
Zheng			2
Perde			
C45		2	6
Zheng			4

4.4 Resultados para o Algoritmo Genético com *Fitness Sharing* e Ajuste dos Atributos Construídos Aplicado a Todos os Indivíduos

Nessa configuração do algoritmo genético, realizou-se o mesmo ajuste (ou "poda") nos indivíduos que foi descrito na seção 3.8. A diferença é que, na configuração cujos resultados são descritos nesta seção, este ajuste foi aplicado a todos os indivíduos de todas as gerações, mantendo a aplicação do método de *niching*, *fitness sharing* (Figura 4.4). Os resultados obtidos são mostrados na tabela 4.5. Observa-se ganho significativo de performance em relação ao C4.5, ao se utilizar os atributos construídos pelo algoritmo genético, nas mesmas bases de dados mencionados na seção anterior (Balance-Scale, Monks-2, Motorola e Tic-tac-toe). Também se observa uma perda significativa de performance nos mesmos data sets mencionados na seção anterior (CMC, Diabetes, Hepatitis, Liver Disorder, Monks-3 e Veh). De forma semelhante se apresentou os resultados do algoritmo genético em relação ao método proposto por Zheng. O primeiro ganhou do segundo nos duas bases de dados mencionados na seção anterior (Iris e Monks-2), mas perdeu somente em três dos cinco bases de dados mencionados anteriormente (Diabetes, Liver Disorder e Veh).

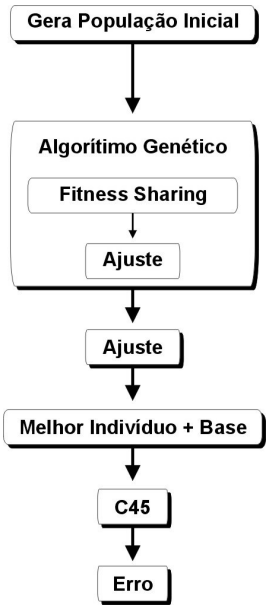


Figura 4.4 – Esquema da execução do algoritmo com o ajuste em todas as gerações

Tabela 4.5: Algoritmo Genético com *Fitness Sharing* e Ajuste dos Atributos Construídos Aplicado a Todos os Indivíduos

Base de Dados	**Taxa de Erro (%)**		
	Atributos Originais	**Atrib. Original + Zheng**	**Atributos originais + AG**
Audiology	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03
Balance-Scale	22,72 ± 0,81	20,75 ± 1,49	**18,85 ± 1,44 (+)**
Car	2,84 ± 0,26	2,84 ± 0,26	2,84 ± 0,26
CMC	49,12 ± 1,26	51,23 ± 1,71	55,05 ± 2,50 (-)
Dermatology	3,92 ± 0,79	4,53 ± 0,93	3,63 ± 0,76
Diabetes	27,32 ± 1,13	25,94 ± 1,86	49,82 ± 6,01 (-)(/)
Glass	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79
Heart-Disease	44,65 ± 2,46	44,00 ± 2,21	41,67 ± 2,03
Hepatitis	17,34 ± 2,04	20,49 ± 2,28	26,18 ± 5,54 (-)
Iris	6,01 ± 2,52	36,65 ± 1,80 (-)	**12,00 ± 5,77 (*)**
Liver Disorder	32,98 ± 3,10	36,39 ± 2,45	57,74 ± 1,63 (-)(/)
Lung Cancer	52,14 ± 10,76	64,29 ± 12,82	44,29 ± 11,31
Monks-1	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Monks-2	29,60 ± 0,04	24,50 ± 0,03	**0,00 ± 0,00 (+) (*)**
Monks-3	0,00 ± 0,00	2,80 ± 0,01 (–)	2,80 ± 0,01 (–)
Motorola	4,92±0,54	4,19±1,01	**2,87±0,71 (+)**
Promoters	18,88 ± 2,19	**13,25 ± 1,49 (+)**	17,25 ± 4,19
Tic-tac-toe	14,31 ± 1,14	**5,98 ± 0,69 (+)**	**5,34 ± 0,60 (+)**
Veh	30,60 ± 1,18	32,79 ± 1,18	47,26 ± 4,31(-)(/)
Wisconsin breast cancer	4,57 ± 0,64	4,87 ± 0,80	5,83 ± 0,84
Ganha			
C45		2	4
Zheng			2
Perde			
C45		2	6
Zheng			3

4.5 Resultados para Algoritmo Genético com *Fitness Sharing* Retornando Múltiplos Indivíduos (Atributos)

Em um último experimento realizado, estendeu-se o procedimento de designação de resultado do algoritmo genético. Mais precisamente, em vez de retornar como solução um único atributo construído pelo algoritmo genético, o algoritmo genético foi estendido para retornar, como solução, vários atributos construídos, sendo que todos os atributos retornados pelo algoritmo genético são adicionados à base de dados fornecida ao C4.5. Para definir quais atributos são retornados pelo algoritmo genético, dentre todos os atributos (indivíduos) da população final, foi utilizado o método proposto na seção 3.9. Obteve-se os resultados apresentados na tabela 4.6, mantendo a aplicação da *fitness sharing* (Figura 4.5). Uma vez mais, houve melhora na performance do C4.5, ao se utilizar os atributos construídos pelo algoritmo genético, em quatro bases de dados (Balance-Scale, Monks-2, Motorola e Tic-tac-toe), e piora em cinco bases de dados (Diabetes, Hepatitis, Liver Disorder, Monks-3 e Veh). Quando comparado ao método de Zheng, o algoritmo genético ganha em duas bases de dados (Iris e Monks-2) e perde em quatro (Diabetes, Hepatitis, Liver Disorder e Veh).

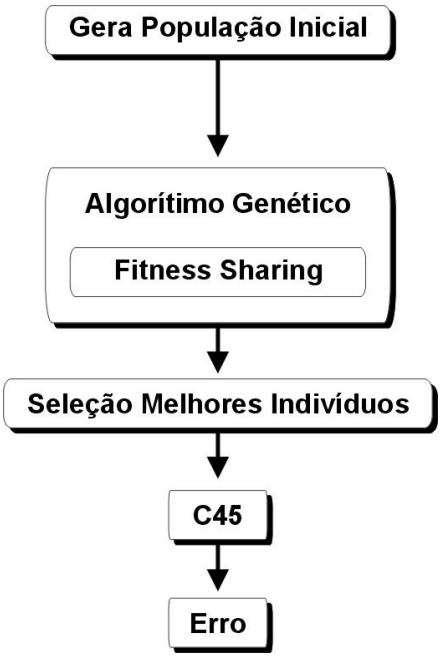


Figura 4.5 – Esquema da execução do algoritmo retornando múltiplos indivíduos.

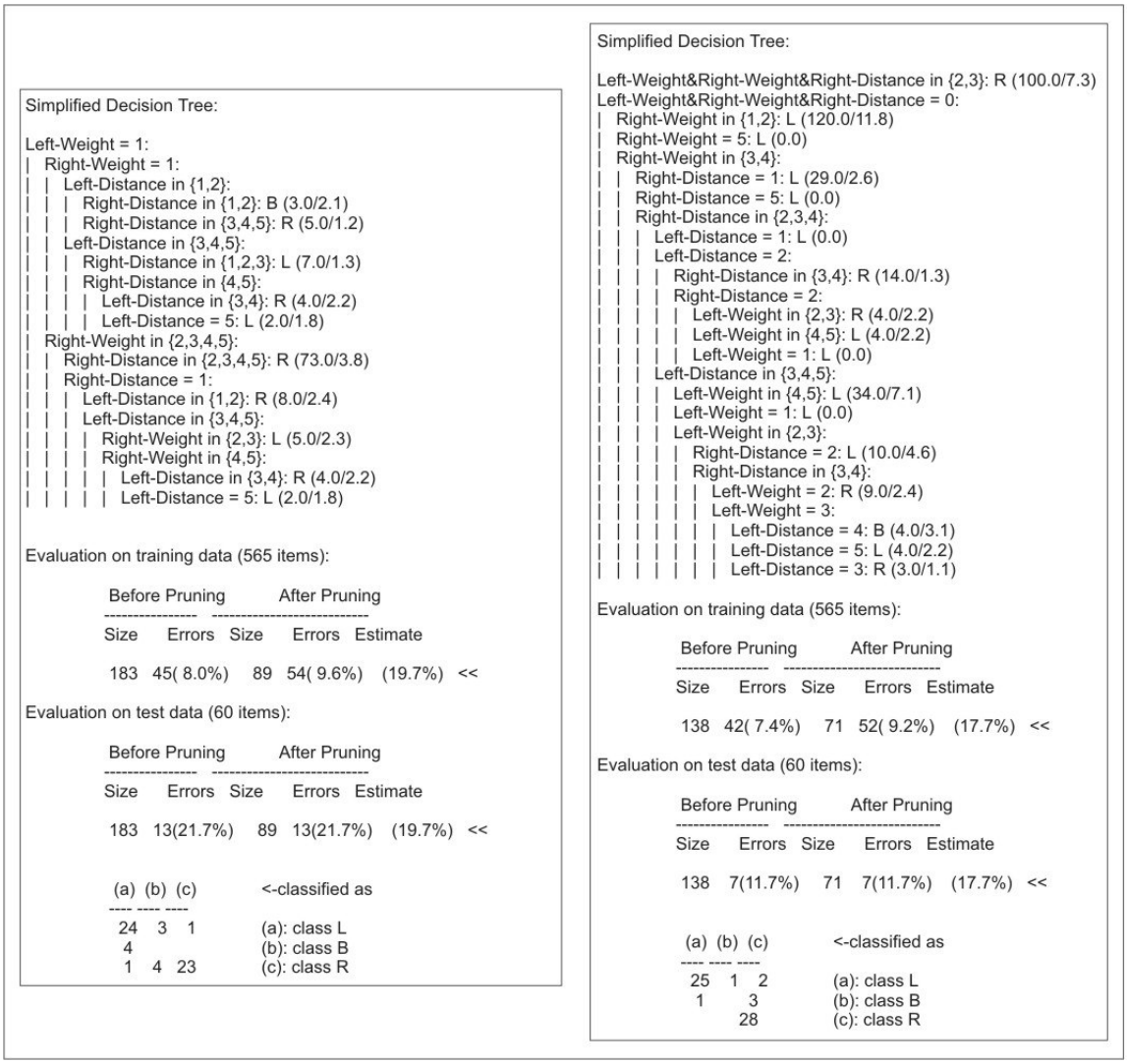
Tabela 4.6: Algoritmo Genético com *Fitness Sharing* Retornando Múltiplos Atributos

Base de Dados	**Taxa de Erro (%)**		
	Atributos Originais	**Atrib. Originais + Zheng**	**Atributos originais + AG**
Audiology	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03
Balance-Scale	22,72 ± 0,81	20,75 ± 1,49	**18,68 ± 1,78 (+)**
Car	2,84 ± 0,26	2,84 ± 0,26	2,84 ± 0,26
CMC	49,12 ± 1,26	51,23 ± 1,71	51,23 ± 2,20
Dermatology	3,92 ± 0,79	4,53 ± 0,93	4,67 ± 0,92
Diabetes	27,32 ± 1,13	25,94 ± 1,86	35,00 ± 2,28 (-)(/)
Glass	2,36 ± 0.79	2,36 ± 0.79	2,36 ± 0.79
Heart-Disease	44,65 ± 2,46	44,00 ± 2,21	44,17 ± 3,45
Hepatitis	17,34 ± 2,04	20,49 ± 2,28	30,16 ± 5,50 (-)(/)
Iris	6,01 ± 2,52	36,65 ± 1,80 (-)	**6,01 ± 2,52 (*)**
Liver Disorder	32,98 ± 3,10	36,39 ± 2,45	59,02 ± 1,78 (-)(/)
Lung Cancer	52,14 ± 10,76	64,29 ± 12,82	47,86 ± 9,59
Monks-1	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Monks-2	29,60 ± 0,04	24,50 ± 0,03	**0,00 ± 0,00 (+) (*)**
Monks-3	0,00 ± 0,00	2,80 ± 0,01 (–)	2,80 ± 0,01 (–)
Motorola	4,92±0,54	4,19±1,01	**2,82±0,71 (+)**
Promoters	18,88 ± 2,19	**13,25 ± 1,49 (+)**	16,63 ± 4,56
Tic-tac-toe	14,31 ± 1,14	**5,98 ± 0,69 (+)**	**5,34 ± 0,60 (+)**
Veh	30,60 ± 1,18	32,79 ± 1,18	58,16 ± 3,95(-)(/)
Wisconsin breast cancer	4,57 ± 0,64	4,87 ± 0,80	4,59 ± 1,04
Ganha			
C45		2	4
Zheng			2
Perde			
C45		2	5
Zheng			4

4.6 Compreensibilidade

Como citado na seção 3.1, é interessante que o conhecimento adquirido através de um algoritmo de mineração de dados seja compreensível. Com esse intuito, o indivíduo inserido na base variava entre no mínimo dois pares <Atr,Val> e sete pares <Atr, Val>. O melhor indivíduo é

inserido na base (ou melhores, para o caso do algoritmo que insere mais de um indivíduo), e pode ser utilizado para a construção da árvore de decisão pelo algoritmo genético, podendo melhorar o resultado da classificação. Na figura 4.6, temos a comparação entre a árvore gerada na aplicação do C4.5 nos dados originais da base *Balance-Scale*, e na base estendida com o atributo construído pelo algoritmo genético com *fitness sharing* (resultados obtidos sobre a décima partição).



```
Simplified Decision Tree:

Left-Weight = 1:
|  Right-Weight = 1:
|  |  Left-Distance in {1,2}:
|  |  |  Right-Distance in {1,2}: B (3.0/2.1)
|  |  |  Right-Distance in {3,4,5}: R (5.0/1.2)
|  |  Left-Distance in {3,4,5}:
|  |  |  Right-Distance in {1,2,3}: L (7.0/1.3)
|  |  |  Right-Distance in {4,5}:
|  |  |  |  Left-Distance in {3,4}: R (4.0/2.2)
|  |  |  |  Left-Distance = 5: L (2.0/1.8)
|  Right-Weight in {2,3,4,5}:
|  |  Right-Distance in {2,3,4,5}: R (73.0/3.8)
|  |  Right-Distance = 1:
|  |  |  Left-Distance in {1,2}: R (8.0/2.4)
|  |  |  Left-Distance in {3,4,5}:
|  |  |  |  Right-Weight in {2,3}: L (5.0/2.3)
|  |  |  |  Right-Weight in {4,5}:
|  |  |  |  |  Left-Distance in {3,4}: R (4.0/2.2)
|  |  |  |  |  Left-Distance = 5: L (2.0/1.8)


Evaluation on training data (565 items):

           Before Pruning           After Pruning
           ----------------   --------------------------
           Size     Errors   Size     Errors   Estimate

           183   45( 8.0%)    89   54( 9.6%)   (19.7%)   <<

Evaluation on test data (60 items):

           Before Pruning           After Pruning
           ----------------   --------------------------
           Size     Errors   Size     Errors   Estimate

           183   13(21.7%)    89   13(21.7%)   (19.7%)   <<


            (a)  (b)  (c)          <-classified as
           ----  ----  ----
             24    3    1          (a): class L
              4                    (b): class B
              1    4   23          (c): class R
```

```
Simplified Decision Tree:

Left-Weight&Right-Weight&Right-Distance in {2,3}: R (100.0/7.3)
Left-Weight&Right-Weight&Right-Distance = 0:
|  Right-Weight in {1,2}: L (120.0/11.8)
|  Right-Weight = 5: L (0.0)
|  Right-Weight in {3,4}:
|  |  Right-Distance = 1: L (29.0/2.6)
|  |  Right-Distance = 5: L (0.0)
|  |  Right-Distance in {2,3,4}:
|  |  |  Left-Distance = 1: L (0.0)
|  |  |  Left-Distance = 2:
|  |  |  |  Right-Distance in {3,4}: R (14.0/1.3)
|  |  |  |  Right-Distance = 2:
|  |  |  |  |  Left-Weight in {2,3}: R (4.0/2.2)
|  |  |  |  |  Left-Weight in {4,5}: L (4.0/2.2)
|  |  |  |  |  Left-Weight = 1: L (0.0)
|  |  |  Left-Distance in {3,4,5}:
|  |  |  |  Left-Weight in {4,5}: L (34.0/7.1)
|  |  |  |  Left-Weight = 1: L (0.0)
|  |  |  |  Left-Weight in {2,3}:
|  |  |  |  |  Right-Distance = 2: L (10.0/4.6)
|  |  |  |  |  Right-Distance in {3,4}:
|  |  |  |  |  |  Left-Weight = 2: R (9.0/2.4)
|  |  |  |  |  |  Left-Weight = 3:
|  |  |  |  |  |  |  Left-Distance = 4: B (4.0/3.1)
|  |  |  |  |  |  |  Left-Distance = 5: L (4.0/2.2)
|  |  |  |  |  |  |  Left-Distance = 3: R (3.0/1.1)

Evaluation on training data (565 items):

           Before Pruning           After Pruning
           ----------------   --------------------------
           Size     Errors   Size     Errors   Estimate

           138   42( 7.4%)    71   52( 9.2%)   (17.7%)   <<

Evaluation on test data (60 items):

           Before Pruning           After Pruning
           ----------------   --------------------------
           Size     Errors   Size     Errors   Estimate

           138    7(11.7%)    71    7(11.7%)   (17.7%)   <<


            (a)  (b)  (c)          <-classified as
           ----  ----  ----
             25    1    2          (a): class L
              1         3          (b): class B
                   28               (c): class R
```

Figura 4.6 – Comparação entre as árvores da base *balance-scale*

Essa figura representa somente a parte inicial da árvore, e destaca-se a utilização do atributo construído na raiz da árvore, na execução do C4.5 na base estendida. O atributo inserido

X-of-N inserido para esse caso foi um indivíduo de N = 3, e destaca-se também o fato do atributo representar a combinação dos três atributos apresentados nos três primeiros níveis da árvore gerada pelo C4.5 na base normal. Na figura 4.7, temos as árvores geradas para a base *diabetes* para o algoritmo genético com *fitness sharing* e ajuste dos atributos construídos aplicado a todos os indivíduos.

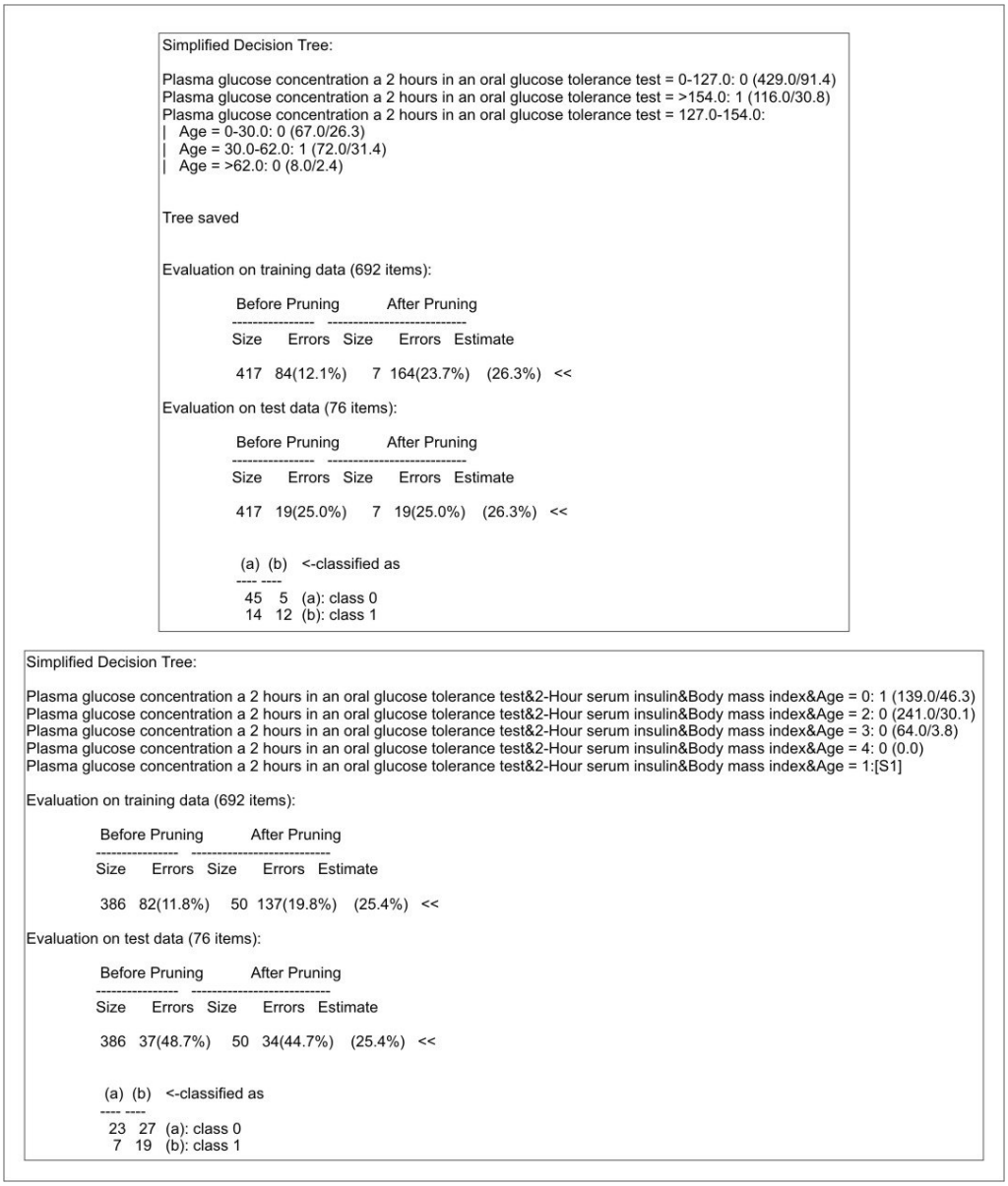
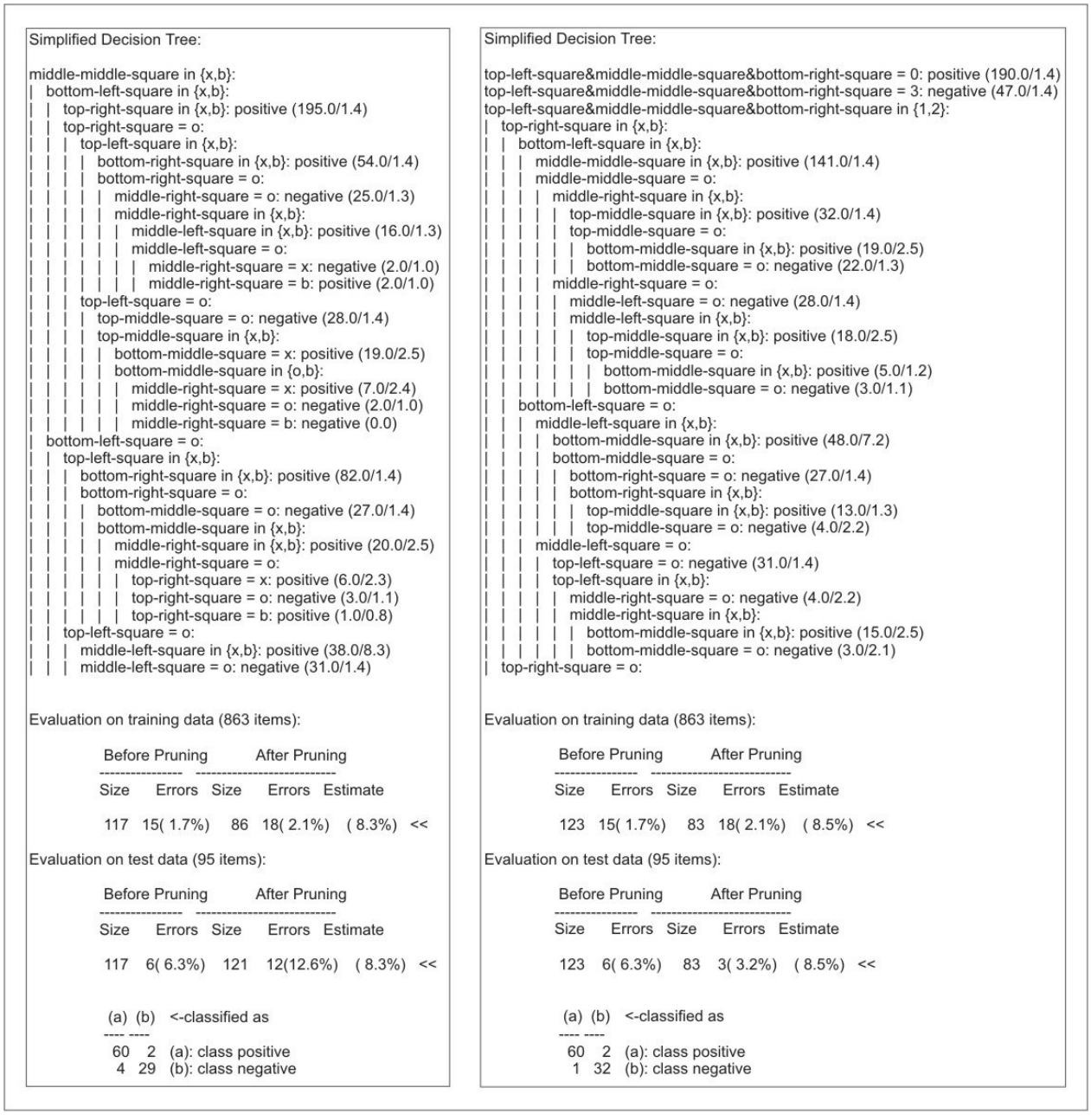


Figura 4.7 – Comparação entre as árvores da base *diabetes*

Para este caso, a árvore gerada sobre a base normal está completa, enquanto que para a base estendida temos somente a parte inicial. Observamos novamente que o atributo construído e utilizado na raiz da árvore, e possui na sua formação os atributos utilizados para gerar a árvore na base normal, mas o resultado da classificação apresentou piora. A figura 4.8 apresenta as árvores para a base *tic-tac-toe*.



```
Simplified Decision Tree:

middle-middle-square in {x,b}:
|  bottom-left-square in {x,b}:
|  |  top-right-square in {x,b}: positive (195.0/1.4)
|  |  top-right-square = o:
|  |  |  top-left-square in {x,b}:
|  |  |  |  bottom-right-square in {x,b}: positive (54.0/1.4)
|  |  |  |  bottom-right-square = o:
|  |  |  |  |  middle-right-square = o: negative (25.0/1.3)
|  |  |  |  |  middle-right-square in {x,b}:
|  |  |  |  |  |  middle-left-square in {x,b}: positive (16.0/1.3)
|  |  |  |  |  |  middle-left-square = o:
|  |  |  |  |  |  |  middle-right-square = x: negative (2.0/1.0)
|  |  |  |  |  |  |  middle-right-square = b: positive (2.0/1.0)
|  |  |  top-left-square = o:
|  |  |  |  top-middle-square = o: negative (28.0/1.4)
|  |  |  |  top-middle-square in {x,b}:
|  |  |  |  |  bottom-middle-square = x: positive (19.0/2.5)
|  |  |  |  |  bottom-middle-square in {o,b}:
|  |  |  |  |  |  middle-right-square = x: positive (7.0/2.4)
|  |  |  |  |  |  middle-right-square = o: negative (2.0/1.0)
|  |  |  |  |  |  middle-right-square = b: negative (0.0)
|  bottom-left-square = o:
|  |  top-left-square in {x,b}:
|  |  |  bottom-right-square in {x,b}: positive (82.0/1.4)
|  |  |  bottom-right-square = o:
|  |  |  |  bottom-middle-square = o: negative (27.0/1.4)
|  |  |  |  bottom-middle-square in {x,b}:
|  |  |  |  |  middle-right-square in {x,b}: positive (20.0/2.5)
|  |  |  |  |  middle-right-square = o:
|  |  |  |  |  |  top-right-square = x: positive (6.0/2.3)
|  |  |  |  |  |  top-right-square = o: negative (3.0/1.1)
|  |  |  |  |  |  top-right-square = b: positive (1.0/0.8)
|  |  top-left-square = o:
|  |  |  middle-left-square in {x,b}: positive (38.0/8.3)
|  |  |  middle-left-square = o: negative (31.0/1.4)


Evaluation on training data (863 items):

             Before Pruning          After Pruning
             ----------------   ---------------------------
             Size     Errors   Size     Errors   Estimate

             117   15( 1.7%)    86   18( 2.1%)    ( 8.3%)   <<

Evaluation on test data (95 items):

             Before Pruning          After Pruning
             ----------------   ---------------------------
             Size     Errors   Size     Errors   Estimate

             117    6( 6.3%)   121   12(12.6%)    ( 8.3%)   <<


              (a)  (b)   <-classified as
              ----  ----
                60    2   (a): class positive
                 4   29   (b): class negative
```

```
Simplified Decision Tree:

top-left-square&middle-middle-square&bottom-right-square = 0: positive (190.0/1.4)
top-left-square&middle-middle-square&bottom-right-square = 3: negative (47.0/1.4)
top-left-square&middle-middle-square&bottom-right-square in {1,2}:
|  top-right-square in {x,b}:
|  |  bottom-left-square in {x,b}:
|  |  |  middle-middle-square in {x,b}: positive (141.0/1.4)
|  |  |  middle-middle-square = o:
|  |  |  |  middle-right-square in {x,b}:
|  |  |  |  |  top-middle-square in {x,b}: positive (32.0/1.4)
|  |  |  |  |  top-middle-square = o:
|  |  |  |  |  |  bottom-middle-square in {x,b}: positive (19.0/2.5)
|  |  |  |  |  |  bottom-middle-square = o: negative (22.0/1.3)
|  |  |  |  middle-right-square = o:
|  |  |  |  |  middle-left-square = o: negative (28.0/1.4)
|  |  |  |  |  middle-left-square in {x,b}:
|  |  |  |  |  |  top-middle-square in {x,b}: positive (18.0/2.5)
|  |  |  |  |  |  top-middle-square = o:
|  |  |  |  |  |  |  bottom-middle-square in {x,b}: positive (5.0/1.2)
|  |  |  |  |  |  |  bottom-middle-square = o: negative (3.0/1.1)
|  |  bottom-left-square = o:
|  |  |  middle-left-square in {x,b}:
|  |  |  |  bottom-middle-square in {x,b}: positive (48.0/7.2)
|  |  |  |  bottom-middle-square = o:
|  |  |  |  |  bottom-right-square = o: negative (27.0/1.4)
|  |  |  |  |  bottom-right-square in {x,b}:
|  |  |  |  |  |  top-middle-square in {x,b}: positive (13.0/1.3)
|  |  |  |  |  |  top-middle-square = o: negative (4.0/2.2)
|  |  |  middle-left-square = o:
|  |  |  |  top-left-square = o: negative (31.0/1.4)
|  |  |  |  top-left-square in {x,b}:
|  |  |  |  |  middle-right-square = o: negative (4.0/2.2)
|  |  |  |  |  middle-right-square in {x,b}:
|  |  |  |  |  |  bottom-middle-square in {x,b}: positive (15.0/2.5)
|  |  |  |  |  |  bottom-middle-square = o: negative (3.0/2.1)
|  top-right-square = o:


Evaluation on training data (863 items):

             Before Pruning          After Pruning
             ----------------   ---------------------------
             Size     Errors   Size     Errors   Estimate

             123   15( 1.7%)    83   18( 2.1%)    ( 8.5%)   <<

Evaluation on test data (95 items):

             Before Pruning          After Pruning
             ----------------   ---------------------------
             Size     Errors   Size     Errors   Estimate

             123    6( 6.3%)    83    3( 3.2%)    ( 8.5%)   <<


              (a)  (b)   <-classified as
              ----  ----
                60    2   (a): class positive
                 1   32   (b): class negative
```

Figura 4.8 – Comparação entre as árvores da base *tic-tac-toe*

Novamente temos o atributo construído como raiz da árvore estendida e formado pelos atributos presentes nos primeiros níveis da árvore normal, sendo que este atributo apresenta-se de forma compreensível, uma vez que a base *tic-tac-toe* com várias partidas do "jogo da velha", e o atributo construído, formado pelos quadrados da diagonal superior esquerda, do meio e da diagonal inferior direita, quando preenchidos pelo mesmo jogador, representa condição de vitória. Finalmente, na figura 4.9 temos as árvores geradas para a base *dermatology* com o indivíduo adicionado pelo algoritmo genético puro.

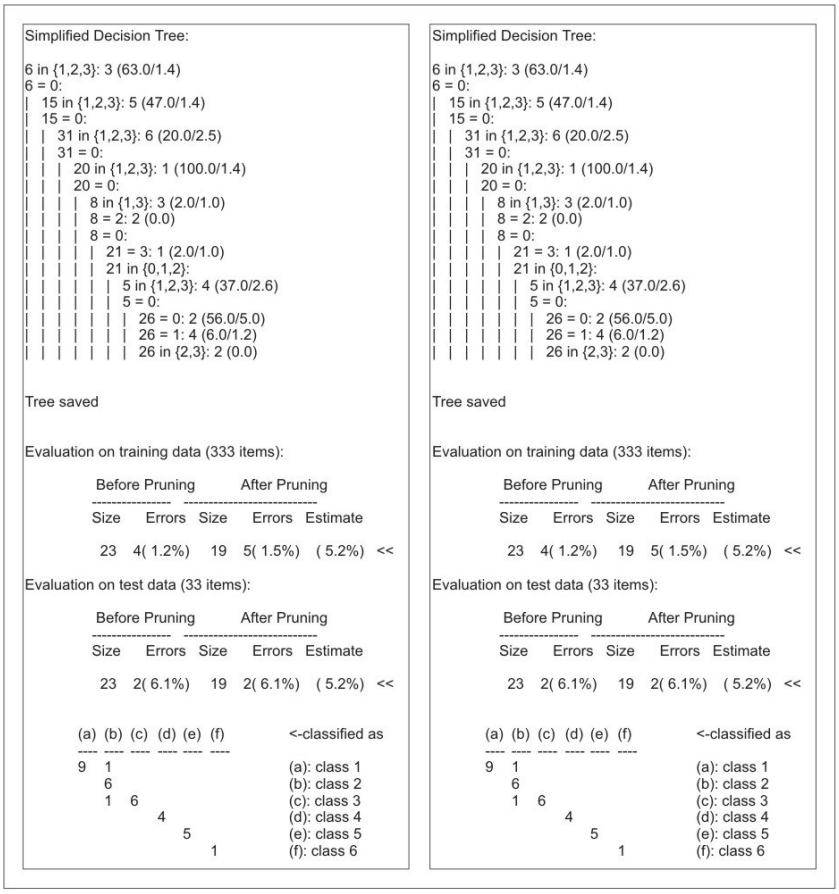


Figura 4.9 – Comparação entre as árvores da base *dermatology*

Verifica-se que o indivíduo não foi utilizado na construção da árvore, sendo as duas idênticas. Um fator que pode justificar tal fato é a quantidade de atributos já presente na base, 33, contra a quantidade de exemplos, 366, significando que esta base já está coberta pelo atributos existentes. Já para os casos anteriores, tivemos sempre uma quantidade bem inferior de atributos por uma quantidade razoável de exemplos (4, 8 e 9 atributos e 625, 768 e 958 exemplos, respectivamente).

4.7 Considerações sobre os Resultados

A partir dos resultados encontrados, percebe-se que todas as variantes do algoritmo genético desenvolvido reduziram significativamente a taxa de erro de classificação, com relação ao C4.5, em quatro bases de dados (Balance-Scale, Monks-2, Motorola e Tic-tac-toe). Além disso, todas as variantes do algoritmo genético reduziram significativamente a taxa de erro, em relação ao método proposto por Zheng, em duas bases de dados (Iris e Monks-2).

Por outro lado, todas as variantes do algoritmo genético aumentaram significativamente a taxa de erro, em relação ao C4.5, em outras três bases de dados (Liver Disorder, Monks-3 e Veh), e aumentaram significativamente a taxa de erro, quando comparado com o método de Zheng, em duas bases de dados (Liver Disorder e Veh).

Os testes foram executados em um Pentium IV 1.8, com 512 RAM, e o custo computacional variou a cada tipo de execução, sendo que para o mais simples, onde foi utilizado o algoritmo genético puro, a variação foi de no mínimo 15 minutos para o máximo de 1:00 hora de execução. Para a mais complexa, onde o ajuste dos indivíduos era realizado para toda geração, variou de 30 minutos para 3:00 horas de execução.

De modo geral observa-se, então, que os resultados obtidos pelo algoritmo genético foram ligeiramente inferiores aos resultados obtidos com o uso do C4.5 puro e com o uso do método proposto por Zheng, mais facilmente visualizável na tabela 4.7, onde encontramos um comparativo geral, com os resultados obtidos pelo C4.5, Zheng, e por cada um dos métodos propostos. É importante ressaltar que uma inspeção visual das árvores geradas pelo C4.5 mostram que o novo atributo X-of-N construído pelo algoritmo genético foi utilizado em larga escala (cerca de 90%) na árvore gerada pelo C4.5, e os resultados obtidos em uma base de produção,

com dados do mundo real, foram melhores que os resultados obtidos com o C4.5, encorajando assim as pesquisas deste projeto, a fim de encontrar um refinamento para o algoritmo para que novos e melhores resultados sejam alcançados.

Cabe ressaltar que nenhum algoritmo de classificação é superior em todas as bases de dados, o que tem sido mostrado tanto teoricamente [Schaffer et al. 1994], [Rao et al. 1995] quanto empiricamente [Michie et al. 1994]. Portanto, embora o AG tenha obtido resultados ligeiramente inferiores ao C4.5 e no método proposto por Zheng de modo geral (considerando as 20 bases de dados), o AG ainda pode ser considerado uma solução útil em alguns casos, tais como as bases de dados Balance-Scale, Monks-2 e Tic-Tac-Toe, nas quais o AG superou os outros dois métodos. Em outras palavras, o AG tem o seu "nicho" de aplicação, ainda que esse nicho seja um pouco menor que o nicho do C4.5 e do método proposto por Zheng.

Tabela 4.7 - Comparativo entre os resultados obtidos

Base de Dados	Taxa de Erro (%)						
	Atributos Originais	**Atributos originais + Zheng**	**Atributos originais + AG Puro**	**Atributos originais + AG com *Niching***	**Atributos originais + AG com Ajuste da População Final**	**Atributos originais + AG com Ajuste de Todas as Populações**	**Atributos originais + AG com adição de mais Atributos**
Audiology	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03	15,40 ± 0,03
Balance-Scale	22,72 ± 0,81	20,75 ± 1,49	19,48 ± 1,44 (+)	18,53 ± 1,44 (+)	18,85 ± 1,44 (+)	18,85 ± 1,44 (+)	18,68 ± 1,78 (+)
Car	2,84 ± 0,26	2,84 ± 0,26	2,90 ± 0,31	2,90 ± 0,31	2,84 ± 0,26	2,84 ± 0,26	2,84 ± 0,26
CMC	49.12 ± 1.26	51.23 ± 1.71	53,55 ± 2,02 (-)	59,87 ± 1,71 (-)(/)	55,10 ± 2,29 (-)(/)	55,05 ± 2,50 (-)	51,23 ± 2,20
Dermatology	3,92 ± 0,79	4,53 ± 0,93	3,33 ± 0,85	3,63 ± 0,76	3,16 ± 0,80	3,63 ± 0,76	4,67 ± 0,92
Diabetes	27,32 ± 1,13	25,94 ± 1,86	30,60 ± 2,05 (-)(/)	32,23 ± 2,64 (-)(/)	39,08 ± 4,96 (-)(/)	49,82 ± 6,01 (-)(/)	35,00 ± 2,28 (-)(/)
Glass	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79	2,36 ± 0,79
Heart-Disease	44,65 ± 2,46	44,00 ± 2,21	40,91 ± 1,86	45,13 ± 2,72	42,43 ± 1,75	41,67 ± 2,03	44,17 ± 3,45
Hepatitis	22,84 ± 1,83	20,49 ± 2,28	22,34 ± 5,26	28,84 ± 4,72 (-)(/)	20,67 ± 1,20 (-)	26,18 ± 5,54 (-)	30,16 ± 5,50 (-)(/)
Ins	6,01 ± 2,52	36,65 ± 1,80 (-)	6,01 ± 2,52 (*)	6,01 ± 2,52 (*)	9,34 ± 5,27 (*)	12,00 ± 5,77 (*)	6,01 ± 2,52 (*)
Liver Disorder	32,98 ± 3,10	36,39 ± 2,45	56,38 ± 2,46 (-)(/)	51,97 ± 4,14 (-)(/)	54,49 ± 1,82 (-)(/)	57,74 ± 1,63 (-)(/)	59,02 ± 1,78 (-)(/)
Lung Cancer	52,14 ± 10,76	64,29 ± 12,82	52,86 ± 10,92	37,86 ± 11,47 (*)	63,57 ± 14,69	44,29 ± 11,31	47,86 ± 9,59
Monks-1	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Monks-2	29,60 ± 0,04	24,50 ± 0,03	0,00 ± 0,00 (+) (*)	0,00 ± 0,00 (+) (*)	0,00 ± 0,00 (+) (*)	0,00 ± 0,00 (+) (*)	0,00 ± 0,00 (+) (*)
Monks-3	0,00 ± 0,00	2,80 ± 0,01 (-)	2,80 ± 0,01 (-)	2,80 ± 0,01 (-)	2,80 ± 0,01 (-)	2,80 ± 0,01 (-)	2,80 ± 0,01 (-)
Motorola	4,92±0,54	4,19±1,01	2,79±0,66(+)	2,51±0,76(+)	2,46±0,76 (+)	2,87±0,71 (+)	2,82±0,71 (+)
Promoters	18,88 ± 2,19	13,25 ± 1,49 (+)	15,00 ± 4,28	13,25 ± 1,98 (+)	17,88 ± 4,16	17,25 ± 4,19	16,63 ± 4,56
Tic-tac-toe	14,31 ± 1,14	5,98 ± 0,69 (+)	5,87 ± 0,67 (+)	5,34 ± 0,47 (+)	5,56 ± 0,72 (+)	5,34 ± 0,60 (+)	5,34 ± 0,60 (+)
Veh	30,60 ± 1,18	32,79 ± 1,18	54,84 ± 3,29 (-)(/)	53,82 ± 2,98(-)(/)	46,36 ± 3,47(-)(/)	47,26 ± 4,31(-)(/)	58,16 ± 3,95(-)(/)
Wisconsin breast cancer	4,57 ± 0,64	4,87 ± 0,80	5,69 ± 0,88	4,83 ± 0,78	5,45 ± 0,90	5,83 ± 0,84	4,59 ± 1,04
Ganha							
C45		2	4	5	4	4	4
Zheng			2	3	2	2	2
Perde							
C45		2	5	6	6	6	5
Zheng			3	3	4	3	4

5. Conclusões e Trabalhos Futuros

Neste projeto apresenta-se um algoritmo genético para construção de atributos, o qual constrói novos atributos do tipo X-of-N a partir dos atributos originais de um dado *data set*. Utilizamos o algoritmo C4.5 para comparar os benefícios dos atributos construídos baseado na taxa de erro de classificação, testando o procedimento em vinte base de dados de domínio público.

Os resultados obtidos mostram que o método proposto tende a piorar (isto é, aumentar) significativamente a taxa de erro do C4.5 com uma freqüência um pouco maior do que melhorar (isto é, diminuir), apesar de o atributo construído ser utilizado na árvore de decisão gerada pelo C4.5 com alta freqüência como a raiz da árvore.

Foram realizadas varias tentativas de melhorar o desempenho do AG, desenvolvendo-se versões mais sofisticadas do algoritmo, envolvendo *niching*, ajuste ("poda") de atributos construídos, e seleção de múltiplos atributos construídos para serem adicionados à base de dados. Infelizmente, essas versões mais sofisticadas não produziram um ganho significativo no desempenho do algoritmo.

Cabe ressaltar que o método proposto por Zheng obteve resultados aproximadamente equivalentes ao C4.5 puro (sem nenhum atributo construído), ao contrario dos bons resultados para aquele método relatados na literatura.

Entre os trabalhos futuros está a implementação de outros métodos de *niching*, com o objetivo de encontrar melhores diversidades genéticas, além do aprimoramento do ajuste proposto, e da rotina para a seleção de novos atributos a serem adicionados na base.

Além disso, o algoritmo genético proposto nesta dissertação constrói apenas atributos do tipo X-of-N. Em pesquisas futuras, pode ser interessante desenvolver um algoritmo genético capaz de construir outros tipos de atributos, aprimorando a autonomia e a flexibilidade do algoritmo genético.

Por fim, neste projeto, a construção foi realizada seguindo uma abordagem de pré-processamento. Em futuras pesquisas pode ser interessante construir atributos com um algoritmo genético utilizando a abordagem entrelaçada e comparar a relação custo-benefício entre ambos.

Referências Bibliográficas

[Bäck et al. 2000] T. Bäck. Mutation operators. In: Bäck, T., Fogel, D. B. e Michalewicz, T. (Eds). *Evolutionary Computation 1*, 237-255. Institute of Physics Publishing, 2000.

[Bensusan & Kuscu 1996] H. Bensaun e I. Kuscu. Constructive induction using genetic programming. *Proc. ICML´96´s Evolutionary Computing and Machine Learning Workshop*. T. Fogarty e G. Venturini (Eds.), 1996.

[Berson & Smith, 1997] A. Berson; S. J. Smith. Data Warehousing, Data Mining, and OLAP. MacGraw-Hill, 1997

[Blickle 2000] T. Blickle. Tournament selection. In: Bäck, T., Fogel, D. B. e Michalewicz, T. (Eds). *Evolutionary Computation 1*, 172-180. Institute of Physics Publishing, 2000.

[Bloedorn & Michalski 1998] E. Bloedorn and R.S. Michalski. Data driven constructive induction: methodology and applications. In: H. Liu & H. Motoda (Eds.) *Feature Extraction, Construction and Selection: a data mining perspective*, 51-68. Kluwer, 1998.

[Carvalho 1999] D. R. Carvalho. Data Mining através de indução de regras e algoritmos genéticos. Dissertação de Mestrado. Programa de Pós-Graduação em Informática Aplicada. PUC-PR, 1999.

[Carvalho 2001] D. R. Carvalho. Uma revisão de métodos de niching em algoritmos genéticos. Estudo Individual. Programa de Pós-Graduação em Informática Aplicada. PUC-PR, 2001.

[Eshelman, Caruana e Schaffer 1989] L. J. Esehlman, R. A. Caruana e J. D. Shaffer. Biases in the Crossover Ladscape. *Proc. Int. Conf. Genetic Algorithms (ICGA-89),* 10-19, 1989.

[Eshelman 2000] L. J. Eshelman. Genetic Algorithms. In: Bäck, T., Fogel, D. B. e Michalewicz, T. (Eds). *Evolutionary Computation 1*, 64-80. Institute of Physics Publishing, 2000.

[Freitas 2001a] A. A. Freitas. Apostila da Disciplina de Data Mining. Mestrado em Informática Aplicada. PUC-PR, 2001 (http://www.ppgia.pucpr.br/~alex)

[Freitas 2001b] A. A. Freitas. *Understanding the Crucial Role of Attribute Interaction in Data Mining.* Artificial Intelligence Review 16(3), pp. 177-199, 2001.

[Grefenstette 2000] J. Grefenstette. Proportional selection and sampling algorithms. In: Bäck, T., Fogel, D. B. e Michalewicz, T. (Eds). *Evolutionary Computation 1*, 172-180. Institute of Physics Publishing, 2000.

[Goldberg 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[Hand 1997] D. Hand. *Constructing and Assessing Classification Rules.* John Wiley & Sons, 1997.

[Hu e Kibbler 1996] Y. Hu e D. Kibler. Generation of Attributes for Learning Algorithms. In *Procedures of the thirteenth AAAI*, pages 806-811, 1996.

[Hu 1998] Y.-J. Hu. A genetic programming approach to constructive induction. *Genetic Programming 1998: Proc. 3rd Int. Conf.*, 146-151. Morgan Kaufmann, 1998.

[Kerber 1992] R. Kerber. ChiMerge: Discretization of numeric attributes. *Proc. 1992 Conf. American Assoc. for AI (AAAI-92),* 123-128.

[Kinnear, Jr 2000] K.E. Kinnear, Jr. Derivative methods in genetic programming. In: Bäck, T., Fogel, D. B. e Michalewicz, T. (Eds). *Evolutionary Computation 1*, 103-113. Institute of Physics Publishing, 2000.

[Kohavi & Sahami 1996] R. Kohavi e M. Sahami. Error-based and entropy-based discretization of continuos features. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD-96)*, 114-119. AAAI, 1996.

[Kuscu 1999] I. Kuscu. A genetic constructive induction model. *Proc. 1999 Congress on Evolutionary Computation (CEC-1999)*, 212-217. IEEE, 2000.

[Mahfoud 2000] S. W. Mahfoud. Niching methods. In: Bäck, T., Fogel, D. B. e Michalewicz, T. (Eds). *Evolutionary Computation 2*, 87-92. Institute of Physics Publishing, 2000.

[Michie et al. 1994] D. Michie, D. J. Spiegelhalter and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood, 1994.

[Miller 1956] G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review 63*, 81-96. 1956.

[Quinlan 1993] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Rao et al. 1995] R. B. Rao, D. Gordon and W. Spears. For every generalization action, is there really an equal and opposite reaction? Analysis of the conservation law for generalization performance. *Proc. 12th Int. Conf. Machine Learning (ML-95)*, 471-479. 1995.

[Schaffer et al. 1994] C. Schaffer. A conservation law for generalization performance. *Proc. 11th Int. Conf. Machine Learning*, 259-265. 1994.

[Syswerda 1989] G. Syswerda. Uniform crossover in genetic algorithms. *Proc. Int. Conf. Genetic Algorithms (ICGA-89),* 2-9, 1989.

[Vafaie & De Jong 1998] H. Vafaie and K. De Jong. Evolutionary Feature Space Transformation. In Liu, H. e Motoda H. (Eds.) *Feature Extraction, Construction and Selection: a Data Mining Perspective*, 307-323. Kluwer, 1998.

[Weiss & Kulikowski 1991] S. M. Weiss and C. A. Kulikowski. Computer System that Learn. San Mateo, CA; Morgan Kaufmann, 1991.

[Zheng 1995] Z. Zheng. Constructing nominal X-of-N attributes. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* 1064-1070. Morgan Kaufmann, 1995.

[Zheng 1998] Z. Zheng. A Comparison of Constructing Different Types of New Feature for Decision Tree Learning. In Liu, H. e Motoda H., (Eds.), *Feature Extraction, Construction and Selection: a Data Mining Perspective*, 239-255. Kluwer, 1998.