APRENDIZADO GENÉTICO DE REGRAS DE DECISÃO UTILIZANDO A CODIFICAÇÃO NATURAL E NOVOS OPERADORES DE RECOMBINAÇÃO

Cristiano Grijó Pitangui

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO

Aprovada por:

Prof. Gerson Zaverucha, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Marco Aurélio Pacheco, Ph.D.

Prof. Carlos Cristiano Borges, D. Sc.

RIO DE JANEIRO, RJ - BRASIL

AGOSTO DE 2007

PITANGUI, CRISTIANO GRIJÒ

Aprendizado Genético de Regras de Decisão Utilizando a Codificação Natural e Novos Operadores de Recombinação [Rio de Janeiro] 2007

X, 149 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Algoritmos Genéticos
2. Aprendizado de Regras de Decisão
3. Codificação Natural

I. COPPE/UFRJ II. Título (série)

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

APRENDIZADO GENÉTICO DE REGRAS DE DECISÃO UTILIZANDO A CODIFICAÇÃO NATURAL E NOVOS OPERADORES DE RECOMBINAÇÃO

Cristiano Grijó Pitangui

Agosto/2007

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta uma proposta de construção de uma máquina de aprendizado genético de regras em que uma nova abordagem híbrida para a evolução dos conceitos é utilizada. O objetivo do sistema proposto é alcançar alta precisão (*accuracy*) de classificação utilizando teorias simples (compostas de poucas regras e poucos atributos). As regras são representadas na Forma Normal Disjuntiva Modificada e codificadas com a Codificação Natural, porém, o sistema possui as exclusivas propriedades: mecanismo implícito de seleção de atributos, nova abordagem híbrida para a evolução de teorias, novo operador de recombinação para dados discretos e novo operador de recombinação para dados contínuos (discretizados). O sistema foi avaliado em comparação a sistemas não genéticos que representam o estado da arte em aprendizado de regras de decisão (C4.5 e RIPPER) bem como comparado com outros sistemas de aprendizado genético de regras (GAssist e XCS). Os resultados mostram que o sistema proposto obtém regras de decisão bastante simples e com alta precisão de classificação quando comparado a outros sistemas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

GENETIC DECISION RULE LEARNING USING NATURAL CODING AND NEW CROSSOVER OPERATORS

Cristiano Grijó Pitangui

August/2007

Advisor: Gerson Zaverucha

Department: System and Computing Engineering

This work presents a genetic based system for rule learning where a new hybrid approach for the evolution of the concepts is used. The objective of the considered system is to achieve high classification accuracy using simple theories (composed of few rules and few attributes. The rules are represented in the Modified Disjunctive Normal Form and codified with the Natural Coding, however, the system has the exclusive properties: an implicit feature selection mechanism, a new hybrid approach for the evolution of theories, new crossover operator for discrete data and a new crossover operator for continuous data (discretized). The system was evaluated in comparison to non-genetic systems that represent the state of the art in rule learning (C4.5 and RIPPER) as well as compared with other genetic systems for rule learning (GAssist and XCS). Results show that the proposed system obtains simple decision rules with high classification accuracy when compared to other systems.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

O capítulo apresenta o problema abordado neste trabalho, bem como um resumo das principais contribuições do sistema proposto para a resolução do problema apresentado. Adicionalmente, uma visão geral de cada capítulo desta tese é também realizada.

1.1 Introdução

Esta tese é focada em uma área de Inteligência Artificial (IA) chamada de Aprendizado de Máquina (AM). O AM lida com questões sobre como construir sistemas que automaticamente melhoram com a experiência (Mitchell, 1997). Mais precisamente, essa tese lida com uma subcategoria do AM: o aprendizado supervisionado, que, de maneira geral, pode ser definido como o processo de aprendizado que apresenta alguma forma de tutor (humano ou automático) que fornece ao sistema um *feedback* direto sobre a sua performance (Mitchell, 1997). Isso é geralmente alcançado com o fornecimento de um conjunto de treinamento para o sistema de aprendizado. Esse conjunto de treinamento pode ser visto como um tipo de experiência em que o sistema se baseia para ajustar a si mesmo e assim comportar-se de maneira correta (Mitchell, 1997).

Este trabalho é baseado em Aprendizado Genético Supervisionado, um dos vários paradigmas de aprendizado supervisionado Este paradigma pode ser definido como qualquer tipo de aprendizado supervisionado que em que o mecanismo de busca se baseie no ramo da Computação Evolucionista (CE) (Michalewicz, 1996). Técnicas CE são ferramentas de otimização inspiradas em certos processos biológicos como a seleção natural de Darwin, recombinação e mutação de indivíduos.

Especificamente, a técnica de CE escolhida para a realização do aprendizado foi o Algoritmo Genético (AG) (Holland, 1975). Tipicamente, em um AG, um conjunto de soluções candidatas (indivíduos) é transformado (evoluído) ao longo de várias iterações (gerações), até que algum critério de parada seja satisfeito. As transformações executadas sobre as soluções candidatas são realizadas pelos operadores de seleção, recombinação e mutação. Estes operadores são os responsáveis pela simulação do processo de evolução natural e, dessa forma, por guiar o conjunto de soluções candidatas a regiões ainda mais promissoras do espaço de busca.

Este trabalho apresenta um sistema genético para aprendizado de regras de decisão na tarefa de classificação. Em muitos problemas, uma única regra não será capaz de classificar toda a base de dados, i.e., serão necessárias mais regras para que o aprendizado ocorra de maneira efetiva. Sobre este assunto, existem basicamente dois tipos de metodologias (Michigan e Pittsburgh) para abordagem do problema de aprendizado genético.

Na abordagem de Michigan (Holland e Reitman, 1978), um indivíduo codifica uma única regra e de maneira geral, o conjunto de indivíduos é considerado a solução do problema.. Na abordagem de Pittsburgh (Smith, 1980), um indivíduo representa a solução do problema, ou seja, o indivíduo é composto de uma ou mais regras. Ambas as abordagens apresentam vantagens e desvantagens, e apesar de Pittsburgh obter resultados (teorias) com menos regras, a abordagem apresenta um maior consumo de tempo e memória (Bacardit e Butz, 2004).

Outro fator de extrema importância ao aprendizado genético se relaciona a forma de representação de uma regra. Para se representar uma regra, deve-se escolher a linguagem que a descreve bem como o tipo de codificação necessária para representá-la. As principais linguagens de representação são: Forma Normal Conjuntiva, Forma Normal Disjuntiva e Forma Normal Disjuntiva Modificada, esta última proposta em (Michalski, 1983). Considerando a questão da codificação, a abordagem mais comumente utilizada é a representação binária (Dejong et al, 1993). Entretanto, (Aguilar-Ruiz et al, 2002) apresenta uma representação mais compacta para o problema. Esta representação é chamada de Codificação Natural.

Uma nova proposta de codificação para abordagem genética implica, geralmente, em uma modificação os operadores de busca (mutação e recombinação). Infelizmente, na maioria das vezes, os novos operadores propostos não exploram o espaço de busca como quando se utiliza a representação binária. Algumas vezes, os novos operadores culminam para uma exploração inadequada do espaço de busca considerando a proposta dos operadores padrões de mutação e recombinação. Esse problema ocorreu com a proposta de Codificação Natural (Aguilar-Ruiz et al, 2002). Embora os operadores de mutação funcionem extremamente bem, tanto para o caso de atributos contínuos e discretos, os operadores de recombinação deixam a desejar. O operador de recombinação de Aguilar-Ruiz et al para dados contínuos, sofre de um baixo poder de exploração, enquanto que ó operador para dados discretos funciona

como um novo tipo de mutação, não se aproximando, portanto, do operador de recombinação padrão.

1.2 Novo Sistema Genético e Contribuições

Este trabalho propõe um sistema genético para aprendizado de teorias. O objetivo do sistema proposto é alcançar alta precisão (*accuracy*) de classificação utilizando teorias simples (compostas de poucas regras e poucos atributos). Com este objetivo em mente, são apresentadas em seguida algumas diretrizes assumidas para o desenvolvimento do sistema, bem como uma explicação que justifica a razão destas serem adotadas:

Utilização da Forma Normal Disjuntiva Modificada (FNDM): A FNDM (Michalski, 1983) permite uma maior compactação para a representação de regras e por isso é utilizada na maioria dos sistemas que usam um AG para aprendizado de regras.

Utilização da Codificação Natural: O uso deste tipo de codificação (Aguilar-Ruiz et al, 2002) garante grande redução na quantidade de memória gasta para representar um indivíduo.

Seleção Genética de Atributos: Diversos trabalhos, (Kohavi e John, 1996) (Guyon e Elisseeff, 2003) demonstram a eficácia da aplicação de métodos de seleção de atributos a problemas de classificação. Mais precisamente, (Yang, 1998) e (Punch et al, 1998) exibem a eficácia da aplicação de AGs ao problema de Seleção de Atributos. Além de um aumento visível da precisão de classificação, chama atenção o baixo número de atributos selecionados para a classificação dos dados. Assim, pretende-se obter teorias com alta precisão de classificação que utilizem um número reduzido de atributos.

Utilização da Abordagem Híbrida para Evolução da Teoria: O sistema codifica um indivíduo como uma teoria (Pittsburgh) enquanto tenta explorar máximo potencial de classificação de cada regra (Michigan) presente no indivíduo. O modelo híbrido proposto neste trabalho adiciona novas regras em uma teoria somente após regras previamente adicionadas terem sido amplamente evoluídas. Dessa forma, espera-se obter teorias compostas por um número reduzido de regras.

Novos Operadores de Recombinação: O trabalho (Aguilar-Ruiz et al, 2002) apresentou juntamente com a Codificação Natural, dois novos operadores de recombinação (relativos a dados discretos e contínuos). Entretanto, como será demonstrado, os operadores de Aguilar-Ruiz et al apresentam algumas desvantagens e

por essa razão dois novos operadores de recombinação são propostos e utilizados no sistema.

Resumidamente, este trabalho é apresenta uma proposta de construção de uma máquina de aprendizado genético de regras em que uma nova abordagem híbrida para a evolução dos conceitos é utilizada. As regras são representadas na Forma Normal Disjuntiva Modificada e codificadas com a Codificação Natural, porém o sistema possui as seguintes e, atualmente, exclusivas propriedades:

(1) – Mecanismo Implícito de Seleção de Atributos (MISA).
(2) – Nova abordagem Híbrida para a evolução de teorias.
(3) – Novo operador de recombinação para dados discretos.
(4) – Novo operador de recombinação para dados contínuos (discretizados).

1.3 Visão Geral

Os principais tópicos expostos em cada um dos capítulos do trabalho são apresentados em seguida.

O capitulo 2 apresenta uma revisão sobre Algoritmos Genéticos. Um pouco de sua teoria, bem como uma simples analogia com o processo de evolução natural é apresentado. Finalmente, este capítulo analisa os AGs em comparação a outros algoritmo de busca.

O capitulo 3 apresenta de que forma um Algoritmo Genético pode ser aplicado ao problema de aprendizado de regras. Três sistemas genéticos com abordagens distintas para a evolução de regras são apresentados com objetivo de elucidar os principais tópicos a serem considerados na aplicação de um AG ao problema abordado.

O capitulo 4 propõe um sistema genético para aprendizado de regras de decisão. As principiais contribuições do trabalho são apresentadas e o ciclo de execução do sistema é discutido.

O capitulo 5 avalia o sistema proposto bem como cada um de seus principais componentes.

O capitulo 6 apresenta as conclusões relativas a cada uma das quatro contribuições deste trabalho. Em seguida, algumas conclusões gerais sobre o sistema proposto, bem como algumas diretrizes para trabalhos futuros são apresentadas.

2. Algoritmos Genéticos: Uma Revisão

Neste capítulo é apresentada uma revisão sobre Algoritmos Genéticos (AGs). Um pouco de sua teoria, bem como uma simples analogia com o processo de evolução natural é apresentado. Seus principais operandos (gene, cromossomo e população) e operadores (mutação, recombinação e seleção) são explicados e uma breve análise de seus principais parâmetros é também apresentada. Finalmente, discutem-se os AGs do ponto de vista de algoritmos de busca, com objetivo de situar a técnica entre os vários outros tipos de algoritmos.

2.1 Introdução

Algoritmos Genéticos (Holland, 1975) são algoritmos de busca estocásticos inspirados na teoria moderna da evolução surgida em 1940. Esta teoria combina as descobertas sobre a hereditariedade feitas no inicio do século XX com a teoria da evolução descrita por Charles Darwin em a *Origem das Espécies* publicado em 1859. Basicamente, as seguintes observações e respectivas conclusões são a essências da teoria da evolução de Charles Darwin (Amabis e Martho, 1994):

Observação 1: As populações naturais de todas as espécies tendem a crescer rapidamente, pois o potencial reprodutivo dos seres vivos é muito grande.

Observação 2: O tamanho das populações naturais, a despeito de seu enorme potencial de crescimento, se mantém relativamente constante ao longo do tempo. Isso se deve ao fato de que número de indivíduos numa população é limitado pelo ambiente (disponibilidade de alimento e locais de procriação, presença de predadores naturais, parasitas etc.).

Conclusão I: Em cada geração, morre grande número de indivíduos, muitos deles sem deixar descendentes.

Observação 3: Os indivíduos de uma população diferem quanto a diversas características, inclusive aquelas que influenciam na capacidade de explorar, com sucesso, os recursos naturais e de deixar descendentes.

Conclusão II: Os indivíduos que sobrevivem e se reproduzem, a cada geração, são, preferencialmente, os que apresentam determinadas características relacionadas com

adaptação às condições ambientais. Essa conclusão resume o conceito darwinista de seleção natural ou sobrevivência do mais apto.

Observação 4: Grande parte das características apresentadas por uma geração é herdada dos pais.

Conclusão III: Uma vez que, a cada geração sobrevivem os indivíduos mais aptos, eles tendem a transmitir aos descendentes as características relacionadas a essa maior aptidão para sobreviver, i.e., para se adaptar. Em outras palavras, a seleção natural favorece, ao longo de gerações sucessivas, a permanência e o aprimoramento de características relacionadas à adaptação.

Resumidamente, os indivíduos mais adaptados ao ambiente possuem maiores chances de produzirem descendentes. Estes descendentes herdam as características de seus pais e consequentemente tendem a ser mais adaptados ao ambiente. Dessa forma, depois de um logo período de tempo, as características que garantem uma melhor adaptação aos indivíduos, tendem a permanecer, enquanto que as características que influenciam os indivíduos de maneira negativa tendem a desaparecer. Assim, a população aproxima-se de uma adaptação ótima ou sub-ótima ao seu ambiente.

A teoria moderna da evolução incorpora a tese central do darwinismo – a teoria da seleção natural –, mas é inovadora porque explica como surge a diversidade de características nos indivíduos de uma população, um ponto que Darwin não pode explicar em sua época.

A teoria moderna da evolução admite que as variações hereditárias presentes nos indivíduos estão diretamente relacionadas aos genes e cromossomos. Dois mecanismos principais são responsáveis pela origem dessas variações: a recombinação e a mutação de genes. Novas combinações gênicas e novos genes originados através destes dois fenômenos garantem que os indivíduos de uma espécie sejam geneticamente variados a cada geração. Essa capacidade de gerar diversidade, denominada de variabilidade gênica, fornece a matéria prima sobre o qual atua a seleção natural.

Os AGs são designados para simular o processo de evolução, e, para isto, utilizam os vários conceitos relacionados a teoria moderna da evolução. A próxima seção apresenta os componentes principais de um Algoritmo Genético.

2.2 Componentes Principais

Para um melhor entendimento, os componentes principais dos AGs serão divididos em operandos e operadores. Os operandos, a saber, gene, cromossomo e população, são manipulados pelos operadores de mutação, recombinação e seleção. A função de aptidão será inclusa na seção de operadores, já que está diretamente relacionada a um cromossomo. Desta forma, esta seção é dividida em operadores e operandos.

2.2.1 Os Operandos

Os AGs mantêm um conjunto de soluções potenciais que são evoluídas ao longo de várias iterações. Na terminologia dos AGs, este conjunto de soluções manipuladas é chamado de população. O processo de evolução de um AG trabalha sobre cada solução denominada cromossomo ou indivíduo[1]. As unidades que constituem um cromossomo são denominadas genes. Assim, considere as seguintes definições:

I. Gene

Definição 2.1 (Gene α): Considere um alfabeto A, i.e., um conjunto de símbolos que são usados na codificação de um problema. Um gene α assume como valor um dos símbolos presentes em A.

II. Cromossomo

Definição 2.2 (Cromossomo χ): Um cromossomo χ é uma cadeia (*string*) de genes. O número de genes de um cromossomo, denotado por $|\chi|$, define o seu comprimento. Considerando A o alfabeto que representa os genes, o conjunto A^n com $n \in N$ é constituído de todos os cromossomos χ com $|\chi| = n$. Ainda, $\chi(i)$ denota o i-ésimo gene do cromossomo χ, com $1 \leq i \leq n$.

O Algoritmo Genético proposto por Holland em 1975, usava a codificação binária para representar os seus cromossomos, i.e., o alfabeto A que representava os genes era formado por apenas dois símbolos, assim $A = \{0,1\}$. Atualmente, diversas formas de alfabeto são usadas, como números naturais (Aguilar-Ruiz e Riquelme, 2002), números reais (Bessaou e Siarry, 2004) e até representações hibridas.

1. Neste trabalho os termos cromossomo e indivíduo são usados de maneira intercambiável.

III. População

Definição 2.3 (População π): Uma população π é um conjunto de cromossomos ou indivíduos. O número de cromossomos numa população é denotado por $|\pi|$. Geralmente, $|\pi|$ é constante durante toda a execução do algoritmo, i.e., o tamanho da população não se modifica durante a evolução. Entretanto, existem casos em que $|\pi|$ varia durante o processo de evolução (Wilson, 1995).

2.2.2 Os Operadores

A seção anterior apresentou os principais operandos de um AG. Esta seção apresenta os operadores que manipulam estes operandos com o objetivo de simular o processo de evolução natural.

I. Função de Aptidão

Para que o processo de evolução possa ocorrer, a distinção entre individuo mais adaptado e individuo menos adaptado deve ser feita. Desta forma, para cada problema defini-se uma função de aptidão que atribui a cada individuo da população uma quantidade numérica que dita a sua adaptação ao ambiente. Portanto, considere a seguinte definição:

Definição 2.4 (Função de Aptidão): Atua sobre um cromossomo χ. Considere A^n o conjunto de todos os cromossomos de tamanho n formados com um alfabeto A. A função Apt: $A^n \rightarrow R$ é chamada função de aptidão e atribui um valor Apt(χ) para cada $\chi \in A^n$.

II. Operador de Recombinação

Durante a evolução natural, os indivíduos de uma população cruzam entre si produzindo descendentes que herdam o material genético dos pais. Este cruzamento é denominado recombinação gênica. De forma geral, é através da recombinação que os genes se organizam em novas combinações nos indivíduos sobre os quais atua a seleção natural.

Do ponto de vista de um AG, a recombinação auxilia na criação de novos indivíduos. A idéia da recombinação é a de que as combinações das características que

garantiram uma boa aptidão aos pais, podem resultar em combinações ainda mais eficazes nos descendentes produzidos. Dessa forma, considere a seguinte definição:

Definição 2.5 (Recombinação): Atua sobre um conjunto de cromossomos. Considere s_1 e s_2, subconjuntos de A^n, i.e., $s_1 \subset A^n$ e $s_2 \subset A^n$. Ainda, assuma que a cardinalidade $|s_1| = i$ e $|s_2| = j$, com $i, j \in N$. Assim, o operador de Recombinação Rec é um mapeamento Rec: $(s_1) \rightarrow (s_2)$. De maneira geral, este operador recebe i cromossomos, também chamados de pais, e retorna j cromossomos também chamados de filhos.

Comumente, o operador de recombinação recebe dois pais para a geração de dois filhos. Diversos tipos de operadores de recombinação foram propostos, os mais comuns serão apresentados em seguida.

- Recombinação de um Ponto: Considere um alfabeto A e dois pais p_1 e p_2 de tamanho n tais que $p_1 \in A^n$ e $p_2 \in A^n$. Dado um ponto de corte k escolhido aleatoriamente no intervalo [0, n - 1], os filhos f_1 e $f_2 \in A^n$ serão dados por::

  $$\circ\ f_1(i) = \begin{cases} p_2(i) \text{ se } i \in [0, k[\\ p_1(i) \text{ se } i \in [k, n] \end{cases} \qquad \circ\ f2(i) = \begin{cases} p_1(i) \text{ se } i \in [0, k[\\ p_2(i) \text{ se } i \in [k, n] \end{cases}$$

  com $0 \leq i < n$.

  Exemplo 2.1: Considere um alfabeto $A = \{\alpha, \beta\}$ e dois pais p_1 e p_2 de tamanho 10, i.e., $|p_1| = |p_2| = 10$. Dado o ponto de corte $k = 3$, a figura 2.1 exibe a recombinação de um ponto.

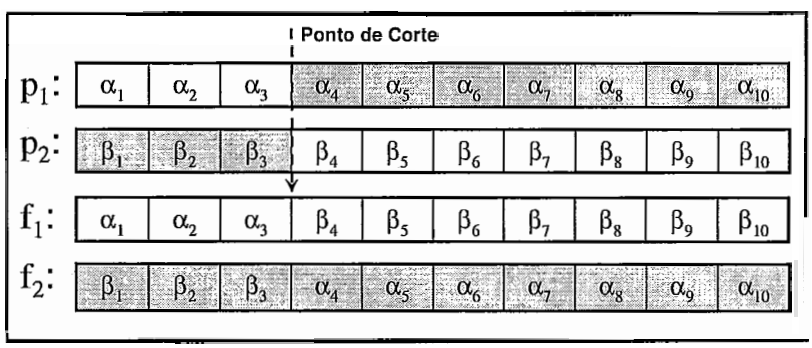

Figura 2.1: Recombinação de um Ponto

- Recombinação de k Pontos: Considere um alfabeto A e dois pais p_1 e p_2 de tamanho n tais que $p_1 \in A^n$ e $p_2 \in A^n$. Dado k pontos de corte escolhidos aleatoriamente no intervalo [0, n-1], os filhos f_1 e $f_2 \in A^n$ serão dados por::
  - f_1: Comece copiando os símbolos de p1 para f1. Para cada ponto de corte encontrado, troque de pai e continue copiando os símbolos do "novo" pai para f_1.
  - f_2: Comece copiando os símbolos de p2 para f2. Para cada ponto de corte encontrado, troque de pai continue copiando os símbolos do "novo" pai para f_2.

Exemplo 2.2: Considere um alfabeto A = {α, β} e dois pais p_1 e p_2 tais que $|p_1| = |p_2| = 10$. Dado dois pontos de corte, $k_1 = 3$ e $k_2 = 7$, a figura 2.2 que mostra a recombinação de dois pontos.

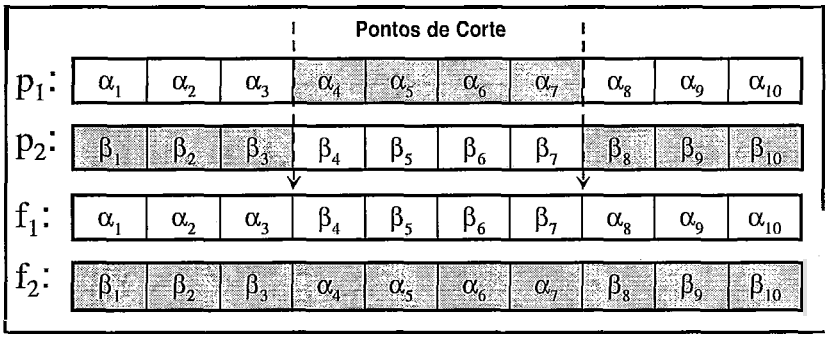

Figura 2.2: Recombinação de 2 Pontos

- Recombinação Uniforme: Este operador foi originalmente descrito em (Syswerda 89). Considere um alfabeto A e dois pais p_1 e p_2 de tamanho n tais que $p_1 \in A^n$ e $p_2 \in A^n$. A recombinação uniforme usa uma máscara numérica para a produção dos descendentes. Esta máscara usa valores reais no intervalo [0,1] que determinam quais os genes serão copiados para os descendentes. Caso o valor da máscara na posição do i-ésimo gene ($1 \leq i \leq n$) seja menor que 0,5, então, $f_1(i) = p_1(i)$ e $f_2(i) =$ p2(i), caso contrário $f_1(i) = p_2(i)$ e $f_2(i) = p_1(i)$. Na maioria das vezes, a máscara numérica é construída de maneira aleatória; assim cada gene possui a probabilidade de 0,5 de ser trocado.

Exemplo 2.3: Considere um alfabeto A = {α, β} e dois pais p_1 e p_2 tais que $|p_1| = |p_2| = 10$. Dada a seguinte máscara de bits: {0,3 - 0,8 - 0,7 - 0,2 - 0,6 - 0,9 - 0,1 - 0,4 - 0,7 - 0,2}, a figura 2.3 que exibe a recombinação uniforme.

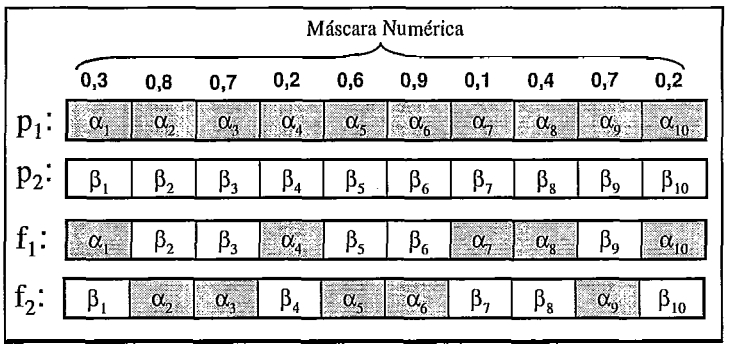

Figura 2.3: Recombinação Uniforme

III. Operador de Mutação

A mutação é a única maneira de se introduzir novos genes na população. Assim, se uma dada mutação favorece o indivíduo em sua adaptação ao ambiente, esse indivíduo possuirá uma maior chance de sobreviver e conseqüentemente de passar a nova característica adquirida por meio desta mutação para outros indivíduos da espécie. Caso a mutação seja desfavorável ao indivíduo, este indivíduo possuirá uma menor chance de se reproduzir e, portanto, de passar essa nova característica adquirida para outros indivíduos (Amabis e Martho, 1994).

Do ponto de vista dos AGs, a mutação tem o objetivo de garantir que todos os possíveis cromossomos possam ser gerados. Já que o operador de recombinação apenas troca os genes já presentes na população como um todo, o operador de mutação é necessário para introduzir novos genes num indivíduo e consequentemente garantir que todos os cromossomos possam ser gerados. Note que a introdução de novos genes na população não adiciona novos símbolos ao alfabeto A dos quais os genes assumem os seus valores. A mutação apenas modifica o valor de um gene fazendo com que o valor do gene modificado assuma outro símbolo de A. Dessa forma, considere a seguinte definição:

Definição 2.6 (Mutação): Atua sobre um cromossomo. Considere A^n o conjunto de todos os cromossomos de tamanho n formados com um alfabeto A O operador de Mutação Mut é um mapeamento Mut: $A^n \rightarrow A^n$. Note que este operador recebe um cromossomo e retorna outro cromossomo.

Precisamente, o operador de mutação atua num gene de um determinado cromossomo. Assim, considerando um cromossomo χ com $|\chi| = n$ e $\chi(i)$ o seu i-ésimo gene ($1 \leq i \leq n$), a mutação atua modificando o valor de $\chi(i)$ de forma que o seu valor seja diferente do seu valor atual. Para a codificação binária, i.e., $A = \{0,1\}$, a mutação tipicamente inverte o valor do gene, assim, se o seu valor era 0 antes da mutação, depois deste operador este gene possuirá o valor 1 e vice-versa.

Exemplo 2.4: Considere um alfabeto $A = \{\alpha, \beta, \gamma\}$ e um cromossomo c_1 formado a partir de A em que $|c_1| = 10$. Dado que o gene escolhido para mutação se localiza na posição 5, a figura 2.4 ilustra o processo de mutação. Note que o gene poderia ser alterado para α ou γ, e, neste exemplo, a alteração foi para α. O cromossomo c'_1 exibe esta alteração.

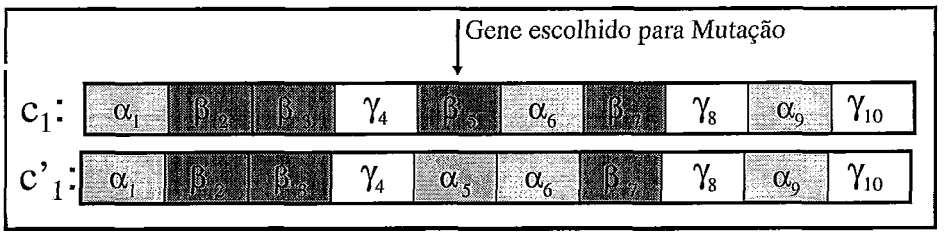

Figura 2.4: Mutação

IV. O Operador de Seleção

Amabis e Martho (Amabis e Marto, 1994) fazem uma descrição precisa do fenômeno de seleção natural:

"A seleção natural não é uma força misteriosa. Ela decorre simplesmente do fato de uma espécie produzir descendência quase sempre superior àquela que efetivamente tem condições de sobreviver. As restrições que o meio impõe à sobrevivência do individuo é que levam a seleção natural: quantidade de alimentos reduzida, competição, predadores, parasitas doenças, acidentes etc. Estes são apenas alguns dos agentes seletivos que causam a morte do indivíduo antes da fase reprodutiva. Os mais aptos a sobreviver são aqueles que, graças à variabilidade gênica, herdaram combinações gênicas favoráveis à sobrevivência em determinado quadro de fatores ambientais."

Pela descrição, fica claro como a seleção natural influencia diretamente o processo de adaptação dos indivíduos ao ambiente em que vivem.

Do ponto de vista dos AGs, o processo de seleção tem como objetivo selecionar os melhores indivíduos da população para que estes possam se reproduzir e sofrer mutações. Com isso, espera-se que novos indivíduos produzidos estejam mais aptos a solucionarem o problema abordado. Embora este processo de seleção seja estocástico, isso não implica que os AGs empregam uma busca sem direção. A chance de um indivíduo ser selecionado para recombinação e/ou mutação, está de alguma forma relacionada à sua aptidão. Assim, considere a seguinte definição:

Definição 2.7 (Seleção): Atua sobre uma população. Dada uma população π, o operador Seleção é um mapeamento Sel: $\pi \rightarrow \pi'$, em que π' é constituída pelos melhores indivíduos π escolhidos por algum critério.

Diversos critérios de seleção foram propostos, os mais comuns são apresentados em seguida.

- Seleção por Roleta: Este é o método de seleção padrão e comumente utilizado. Neste método, cada cromossomo tem a chance de ser selecionado diretamente proporcional a sua aptidão. Assim, o seu efeito depende diretamente da abrangência dos valores de aptidão na população.

  Exemplo 2.5: Considere uma população de quatro cromossomos com as seguintes aptidões descritas na tabela 2.1. A probabilidade de seleção de cada um dos cromossomos, utilizando o método de seleção por roleta é também apresentada na tabela.

Tabela 2.1: Seleção por Roleta

Cromossomo	Aptidão	Probabilidade de Seleção
χ_4	250	83,3%
χ_2	25	10,3%
χ_3	15	6,3%
χ_1	10	3%
Total	**300**	**100%**

Como pode ser verificado, o cromossomo χ_4 possui uma probabilidade muito maior de ser selecionado do que os outros indivíduos. Essa diferença é indesejável em várias aplicações, já que pode facilmente levar a uma estagnação da busca, i.e., uma convergência prematura da população em direção a um cromossomo. Uma

maneira de solucionar o problema é escalonar a aptidão antes da seleção (veja, por exemplo, (Goldberg, 1989)).

- Seleção por *Ranking*: Originalmente proposta por Baker (Baker, 1985), a probabilidade de um cromossomo ser selecionado é proporcional à sua posição ou *ranking* que ele ocupa em relação ao restante da população. Assim, esse tipo de seleção não se baseia diretamente na aptidão, já que esta é usada apenas para ordenar os indivíduos de uma população.

  Exemplo 2.6: Considere os mesmos quatro cromossomos apresentados na tabela 2.1. A tabela 2.2 mostra a probabilidade de seleção de cada cromossomo usando o modelo de seleção em *ranking*. Neste exemplo, os indivíduos foram ordenados em função de sua aptidão sendo que a cada posição foi atribuída uma probabilidade de seleção.

Tabela 2.2: Seleção por *Ranking*

Cromossomo	Aptidão	Posição no Ranking	Probabilidade de Seleção
χ_4	250	1º	40%
χ_2	25	2º	30%
χ_3	15	3º	20%
χ_1	10	4º	10%
Total	**300**	**-**	**100%**

Em relação à tabela 2.1, é possível verificar que as probabilidades de seleção estão mais bem distribuídas entre os indivíduos. Neste caso, não é necessário escalonar a aptidão com o objetivo de se evitar uma convergência prematura, já que o *ranking* pode ser entendido como uma forma de escalonamento.

- Seleção por Torneio: Este método de seleção foi proposto inicialmente por Brindle (Brindle, 1981) e escolhe $k \in N$ indivíduos retornando o que possui o maior valor de aptidão. O número k fornece a pressão de seleção deste método, assim, quanto maior for este valor, maior será a pressão de seleção exercida na população. É importante notar que os indivíduos escolhidos para o torneio são selecionados de maneira aleatória.

Esta seção apresentou os principais operadores dos AGs. A próxima seção dedica-se a apresentar como todos os mecanismos (operandos e operadores) são usados para simular o processo de evolução natural.

2.3 O Ciclo Básico de um Algoritmo Genético

Até agora, pode-se resumir o que foi apresentado da seguinte forma: Um Algoritmo Genético manipula uma população que é constituída por um conjunto de soluções denominadas cromossomos e estes, por sua vez, são constituídos por unidades denominadas genes. Uma função de aptidão avalia a qualidade de adaptação dos indivíduos. A mutação insere novos genes num cromossomo e a recombinação os mistura com os genes já existentes, originando, assim, os indivíduos geneticamente variados de uma população. A seleção natural favorece os portadores de determinados conjuntos gênicos adaptativos. Estes indivíduos tendem a sobreviverem e a se reproduzirem em maior escala que outros.. Em função destes fatores, os indivíduos evoluem ao longo do tempo se tornando cada vez mais aptos a solucionarem o problema abordado.

Essa seção descreve como todos estes componentes são usados para simular o processo de evolução natural. Dessa forma, considere em seguida o algoritmo 2.1 que apresenta o esquema geral dos Algoritmos Genéticos bem como a descrição de seus principais passos.

Algoritmo 2.1 - Algoritmo Genético: Esquema Geral

Entrada:
população **pop**;
Saída:
melhor indivíduo da população;

Começo:
1 – Inicialize pop de forma aleatória;
2 – Avalie cada indivíduo de pop com a função de aptidão;
3 – Enquanto (témino = falso) faça:
4 – pop' ← seleção de pais de pop;
5 – Aplique o operador de recombinação em alguns indivíduos de pop';
6 – Aplique o operador de mutação em alguns indivíduos de pop';
7 – Avalie cada indivíduo de pop' com a função de aptidão;
8 – pop ← Substituição (pop, pop');
Fim-do-Enquanto;
9 – Retorne o melhor indivíduo em pop;
Fim:

Passo 1: Este passo é o responsável pela inicialização dos indivíduos da população. Tradicionalmente, os indivíduos da população são gerados de forma aleatória, tentando,

dessa forma, abranger a maior parte possível do espaço de busca. Algumas vezes, os indivíduos podem ser "semeados" em áreas onde boas soluções são prováveis de serem encontradas. Para mais informações sobre métodos de inicialização, veja (Chou e Chen, 2000).

Passo 2: Este passo avalia cada indivíduo da população utilizando uma função de aptidão definida. De forma geral, esta função é uma medida da capacidade que o indivíduo possui de solucionar o problema.

Passo 3: Este passo define o critério de parada do algoritmo. Na terminologia dos Algoritmos Genéticos, a execução dos passos de 4 a 8 é denominada uma geração. O critério de parada mais comum é definido como sendo um número específico de gerações. Dessa forma, o algoritmo é finalizado assim que ele é executado certa quantidade de vezes. Outro critério de parada está relacionado ao fato de a melhor solução não se modificar após certo número de gerações. Isso acontece quando o algoritmo encontra a solução ótima ou um ponto de ótimo local. Existe ainda outro critério de parada em que o algoritmo é finalizado quando a média da aptidão da população for muito próxima à aptidão do melhor individuo.

Passo 4: Este passo é de extrema importância para o Algoritmo Genético, já que ele simula o processo de seleção natural proposto por Darwin. Neste passo, uma nova população, aqui chamada de pop', é construída selecionando-se os indivíduos mais adaptados de pop. Posteriormente, pop' será utilizada para geração de novos indivíduos.

Passo 5: Neste passo, alguns indivíduos de pop' são recombinados com o objetivo de se produzir novos indivíduos. Estes novos indivíduos são temporariamente armazenados em pop' para que depois sejam colocados em pop.

Passo 6: Este passo é responsável pela inserção de novo material genético em alguns indivíduos de pop'. Também neste caso, estes novos indivíduos são temporariamente armazenados em pop' para posteriormente serem colocados em pop.

Passo 7: Este passo avalia a aptidão dos indivíduos que constituem pop'. Este passo é muito importante, já que em parte define quais indivíduos de pop' serão transferidos para pop.

Passo 8: Esse passo substitui parte ou todos os indivíduos de pop pelos indivíduos de pop'. Aqui, várias possibilidades são possíveis, como por exemplo, substituir os piores indivíduos de pop pelos melhores indivíduos de pop', ou substituir os pais dos

indivíduos de pop' pelos seus filhos (criados com operador de recombinação) ou ainda substituir indivíduos aleatórios de pop pelos indivíduos de pop'. Estes e outros métodos de substituição podem ser encontrados em (Gosh et al, 2000), que apresenta as vantagens e desvantagens de cada abordagem, já que não existe uma abordagem que seja considerada a melhor para os diversos problemas.

Passo 9: Este passo retorna a melhor solução encontrada após o critério de parada ser atingido.

Esta seção apresentou o esquema geral de todos os Algoritmos Genéticos bem como explicou os seus principais passos. A próxima seção discute os principais parâmetros presentes em um AG.

2.4 Principais Parâmetros de um Algoritmo Genético

Independentemente do problema abordado, os AGs apresentam uma série de parâmetros que determinam o seu comportamento e influenciam a qualidade da solução encontrada. Além disso, é importante analisar a influência destes parâmetros para que eles possam ser estabelecidos conforme as necessidades do problema e dos recursos disponíveis. Os principais parâmetros são descritos abaixo.

Tamanho da População: O tamanho da população afeta o desempenho global dos Algoritmos Genéticos. A busca pode ser ineficaz se a população for pequena, pois deste modo existe pouca diversidade genética para representar o espaço a ser explorado. Uma grande população geralmente fornece uma representação adequada do espaço de busca problema, prevenindo, portanto, convergências prematuras para soluções locais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou alternativamente, que algoritmo trabalhe por um período de tempo muito maior ou ainda que o mesmo seja paralelizado. Para mais informações sobre este parâmetro, consulte (Goldberg, 1989).

Taxa ou Probabilidade de Recombinação: Este parâmetro controla a quantidade de indivíduos de uma população que irá sofrer recombinação. De maneira geral, quanto maior a taxa, mais rapidamente novos indivíduos serão produzidos. Se esta for muito alta, indivíduos com boas aptidões podem ser perdidos rapidamente. Com um valor baixo, o algoritmo pode tornar-se muito lento. Geralmente, este parâmetro assume o valor de 80%. Assim, cada indivíduo selecionado possui 80% de chance de ser

submetido a recombinação. Para mais informações sobre este parâmetro, consulte (De Jong e Spears, 1990) e ainda (Eshelman et al, 1989).

Taxa ou Probabilidade de Mutação: Este parâmetro controla a quantidade de indivíduos de uma população que irá sofrer mutação. Argumenta-se que este parâmetro é o mais sensível dentre todos os outros (Bäck, 1995). Uma baixa taxa de mutação previne que uma população fique estagnada em um valor, além de possibilitar que se alcance "qualquer" ponto do espaço de busca. Uma taxa muito alta a induz a uma busca essencialmente aleatória. Geralmente esse valor é de 5%, assim, cada indivíduo selecionado possui 5% de chance de ser submetido à mutação. Para mais informações sobre este parâmetro, consulte (Ochoa et al, 2000).

Taxa de Substituição: Este parâmetro controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, levando, possivelmente, a uma perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo tornar-se muito lento. Geralmente este valor varia de 30 a 50%.

É importante notar que os valores citados para os parâmetros acima descritos devem ser vistos apenas como uma base e não devem ser aplicados diretamente a todos os problemas de forma geral. Geralmente, faz-se necessário a determinação de um conjunto específico de parâmetros para cada problema abordado. Uma maneira robusta de determinar os parâmetros dos AGs para problemas de aprendizado de máquina pode ser encontrada em (Mitchell, 1997).

2.5 Algoritmos Genéticos como um Algoritmos de Busca[2]

Esta seção dedica-se a apresentar de que forma um AG pode ser considerado um algoritmo de busca e desta forma ser aplicado a este tipo de problema. Alguns algoritmos de busca são discutidos nesta seção com objetivo de situar a classe de AGs entre os outros algoritmos conhecidos. Como ilustração de um problema de busca, o clássico problema de 8 rainhas será apresentado. Antes, contudo, se faz necessária a apresentação dos principais componentes de um problema de busca:

2. Todo o conteúdo desta seção, incluindo definições, exemplos e algoritmos foram retirados de (Russell e Norvig, 1995).

- Estado em que um agente[3] começa - denominado de estado inicial.
- Uma descrição das ações possíveis que estão disponíveis para o agente. A formulação mais comum utiliza a função sucessor. Dado um estado particular x, Sucessor(x) retorna um conjunto de pares ordenados (ação, sucessor) em que cada ação é uma das ações válidas no estado x e cada sucessor é um estado que pode ser alcançado a partir de x aplicando-se a ação.
- O espaço de estados que é definido implicitamente pelo estado inicial e a função Sucessor. O espaço de estado nada mais é que o conjunto de estados acessíveis a partir do estado inicial.
- Caminho no estado de espaços: seqüência de estados conectados por uma seqüência de ações.
- Teste de Objetivo: Determina se um dado estado é o objetivo a ser alcançado.
- Custo de Caminho: Atribui um custo numérico a cada caminho.

O objetivo do problema das 8 rainhas é posicionar 8 rainhas em um tabuleiro de xadrez de tal forma que nenhuma rainha ataque qualquer outra. Uma rainha ataca qualquer peça situada na mesma linha coluna ou diagonal. A figura 2.5 ilustra uma tentativa em que a solução falhou: a rainha da coluna mais a direita é atacada pela rainha do canto superior esquerdo.

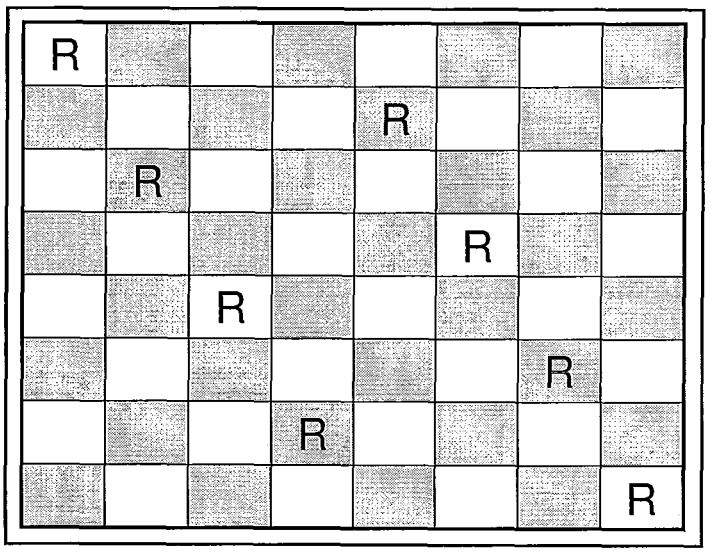

Figura 2.5: Tentativa de Solução que Falhou para o problema de 8 Rainhas

3. Um agente é tudo que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente.

É interessante perceber que existem dois tipos de formulações para o problema de 8 rainhas e de modo geral para qualquer outro problema de busca. As formulações são conhecidas como formulação incremental e formulação de estados completos:

Formulação Incremental: Envolve operadores que ampliam a descrição dos estados, iniciando com um estado vazio: para o problema de 8 rainhas, isso significa que cada ação acrescenta uma rainha ao estado.

Formulação de Estados Completos: Começa com todas as rainhas e as desloca pelo tabuleiro.

A título de esclarecimento, os exemplos 2.7 e 2.8 exibem uma formulação incremental e uma formulação de estados completos para o problema de 8 rainhas.

Exemplo 2.7: Para o problema de 8 rainhas, uma formulação incremental poderia ser dada por:

Estados: Qualquer disposição de 0 a 8 rainhas no tabuleiro.

Estado Inicial: Nenhuma rainha no tabuleiro.

Função Sucessor: Colocar a rainha em qualquer quadrado vazio.

Teste de Objetivo: As 8 rainhas estão no tabuleiro e nenhuma é atacada.

Exemplo 2.8: Para o problema de 8 rainhas, uma formulação de estados completos poderia ser dada por:

Estados: Qualquer disposição de 8 rainhas no tabuleiro.

Estado Inicial: Todas as rainhas no tabuleiro.

Função Sucessor: Retorna todos os estado possíveis gerados pela movimentação de uma única rainha para outro quadrado na mesma coluna que a rainha ocupa.

Teste de Objetivo: As 8 rainhas estão no tabuleiro e nenhuma é atacada.

2.6 Busca de Subida de Encosta (*Hill Climbing*)

Este algoritmo é constituído de apenas um laço repetitivo que se move de forma contínua no sentido do valor crescente – isto é, encosta acima. O algoritmo termina quando alcança um "pico" em que nenhum vizinho tem valor mais alto. A busca subida da encosta não examina antecipadamente valores de estados além dos vizinhos imediatos do estado corrente. O algoritmo 2.2 exibe o processo de subida de encosta.

Algoritmo 2.2 - Algoritmo Subida de Encosta

Entrada:
problema;
Saída:
estado que é um máximo local;
Variáveis Locais:
nó_corrente; nó_vizinho;

```
Começo:
  1 – nó_corrente ← Criar-Nó(Estado-Inicial(problema));
  2 – Repita
      3 – nó_vizinho ← um sucessor corrente com valor mais alto;
      4 – Se (Valor (nó_vizinho) ≤ Valor (nó_corrente) então
            5 – Retorne estado (nó_corrente);
            6 – Finalize Algoritmo;
      7 – nó_corrente ← nó_vizinho
      Fim-Repita;
Fim:
```

A aplicação deste algoritmo para o problema de 8 rainhas, segue a formulação de estados completos exibido no exemplo 2.8. O algoritmo tem que minimizar a solução de custo que retorna a quantidade de pares de rainhas que estão atacando umas as outras, direta ou indiretamente. O mínimo global desta função é 0, que ocorre quando uma das soluções perfeitas é encontrada, ou seja, quando o teste de objetivo é tido como verdadeiro.

Freqüentemente, a subida da encosta funciona muito bem, já que ela progride com grande rapidez em direção a uma solução, porque normalmente é bem fácil melhorar um estado ruim. Infelizmente, a subida da encosta fica paralisada em máximos locais, picos e platôs. Na tentativa de solucionar o problema de paralisação do algoritmo de subida de encosta, algumas variações deste algoritmo foram propostas. Estas variações são explicadas em seguida.

2.6.1 Busca de Subida de Encosta Estocástica (*Stochastic Hill Climbing*)

Subida de Encosta Estocástica escolhe ao acaso entre os movimentos encosta acima: a probabilidade de seleção pode variar com a declividade do movimento encosta acima. Em geral, a subida de encosta estocástica converge mais lentamente que a subida da encosta, mas em algumas topologias de estados encontra soluções melhores.

2.6.2 Busca de Subida de Encosta pela Primeira Escolha (*First Choice Hill Climbing*)

A Subida de Encosta pela Primeira Escolha implementa a subida de encosta estocástica gerando sucessores ao acaso, até ser gerado um sucessor melhor que o estado corrente. Essa é uma estratégia muito utilizada quando um estado tem muitos sucessores (milhares).

2.6.3 Busca Subida de Encosta com Reinício Aleatório (*Random-restart Hill Climbing*)

Os algoritmos de subida de encosta até agora citados são incompleto, i.e., deixam de encontrar um objetivo que existe porque podem ficar paralisados em máximos locais. A subida de encosta com reinicio aleatório conduz uma série de buscas a partir de estados iniciais gerados ao acaso, parando ao encontrar um objetivo. Ela é completa com probabilidade próxima de 1, já que eventualmente irá gerar um estado objetivo como estado inicial.

2.6.4 Considerações Sobre Algoritmos de Subida de Encosta

O sucesso da subida da encosta depende muito da forma da topologia do espaço de estados. Se houver poucos máximos locais e platôs, a subida de encosta com reinicio aleatório encontrará uma solução com muita rapidez. Contudo, em geral, para os problemas que apresentem um grande número de máximos locais, esta gama de algoritmos fica paralisada. Apesar disso, um máximo local razoavelmente bom pode ser encontrada depois de um pequeno número de reinícios.

2.7 Busca de Têmpera[4] Simulada (*Simulated Annealing*)

O algoritmo de têmpera simulada pode ser visto como uma versão de subida de encosta estocástica em que alguns movimentos encosta abaixo são permitidos. Movimentos encosta abaixo são prontamente aceitos no inicio do escalonamento da tempera e depois com menos freqüência com o passar do tempo. Dessa forma, este algoritmo está menos sujeito a ficar paralisado em mínimos locais quando comparado aos algoritmos de subida de encosta. Este algoritmo, bem como o seu funcionamento é apresentado em seguida.

4. Em metalurgia, a têmpera é o processo usado para temperar ou endurecer metais e vidros aquecendo-os em alta temperatura e depois esfriando-os gradualmente, permitindo assim que o material seja misturado em um estado cristalino e de baixa energia.

Algoritmo 2.3 Algoritmo de Têmpera Simulada

Entrada:
problema;
escalonamento; (um mapeamento de tempo para temperatura);

Saída:
estado solução;

Variáveis Locais:
nó_corrente;
nó _próximo;
T (temperatura);

```
Começo:
 1 – Para t ← 1 até ∞ faça:
     2 – T ← Escalonamento(t)
     3 – Se (T = 0) então
         4 – Retorne nó_corrente;
         5 – Finalize Algoritmo;
     6 – nó_próximo ← um sucessor de nó_corrente selecionado aleatoriamente;
     7 – ΔE ← Valor (nó_próximo) - Valor (nó_corrente);
     8 – Se (ΔE > 0) então
         9 – nó_corrente ← nó_próximo;
         10 – Senão: nó_corrente ← nó_próximo com probabilidade e^(ΔE/T);
11 – Fim-Para;
Fim.
```

O laço interno de repetição do algoritmo apresentado é muito semelhante à subida da encosta. Contudo, em vez de escolher o melhor movimento, ele escolhe um movimento aleatório. Se o movimento melhorar a situação, ele será sempre aceito. Caso contrário o algoritmo aceitará este movimento com uma probabilidade menor que 1. Essa probabilidade diminui exponencialmente com a má qualidade do movimento – o valor ΔE segundo o qual a avaliação piora. A probabilidade também diminui à medida que a temperatura T se reduz. Pode-se provar que se o escalonamento diminuir T com lentidão suficiente, o algoritmo encontrará um valor ótimo global com probabilidade próxima de 1.

2.8 Busca em Feixe Local (*Beam Search*)

A busca em feixe local mantém k estados na memória em vez de somente um como ocorreu nos algoritmos de busca outrora descritos. Ele começa com k estados gerados aleatoriamente. Em cada passo são gerados todos os sucessores de todos os k estados. Se qualquer um deles for o objetivo, o algoritmo é finalizado. Caso contrário, ele selecionará os k melhores sucessores a partir da lista completa e repetirá a ação. A busca

em feixe local pode sofrer de falta de diversidade entre os k estados – estes podem ficar rapidamente concentrados em uma pequena região do espaço de estados, tornando a busca um pouco melhor que uma versão dispendiosa da encosta. Uma variante chamada de busca em feixe estocástica, análoga à subida de encosta estocástica, ajuda a atenuar este problema. Em vez de escolher o melhor k a partir do conjunto de sucessores candidatos, a busca em feixe estocástica escolhe k sucessores ao acaso, com a probabilidade de escolher um determinado sucessor que seja uma função crescente de seu "valor" (adaptação ou aptidão).

2.9 Busca com Algoritmos Genéticos

Um AG pode ser visto como uma variante de busca em feixe estocástica, no qual os estados sucessores são gerados pela aplicação dos operadores de seleção, mutação e recombinação. Como ocorre na busca em feixe estocástica, os AGs começam com um conjunto de k estados gerados aleatoriamente (população).

Uma das principais diferenças que devem ser notadas entre as duas técnicas se relaciona a geração dos sucessores. A geração de sucessores na busca em feixe estocástica se faz de forma assexuada, o que difere do AG, já que este apresenta o operador de recombinação que produz novos indivíduos pela troca de seus materiais genéticos, se assemelhando, portanto, a uma reprodução sexuada. Intuitivamente, a recombinação tem a capacidade de recombinar grandes blocos de genes que evoluem de forma independente para executar funções úteis, elevando assim o nível de granularidade em que a busca opera. Entretanto, pode ser demonstrado matematicamente, que se as posições do código genético forem permutadas inicialmente em ordem aleatória, a recombinação não terá nenhuma vantagem. A despeito da recombinação, que parece ser a única vantagem dos AGs sobre a busca em feixe local estocástica, os restantes dos componentes de ambas as técnicas apresentam bastantes pontos em comuns e, dessa forma, estes algoritmos devem exibir um comportamento semelhante.

3. Algoritmos Genéticos para Aprendizado de Regras

Este capítulo apresenta de que forma um Algoritmo Genético pode ser aplicado ao problema de aprendizado de regras. Assim, o capítulo revisa os principais conceitos relacionados ao problema, apresentando algumas linguagens e codificações usadas para representar uma regra. Três sistemas com abordagens distintas para a evolução de regras são apresentados com objetivo de elucidar os principais tópicos a serem considerados na aplicação de um AG ao problema abordado.

3.1 Problemas de Classificação

Essa seção tem o objetivo de introduzir os conceitos básicos relacionados aos problemas de classificação. Posteriormente, será apresentado de que forma um Algoritmo Genético poder ser usado para lidar com esse tipo problema.

Antes de apresentar a definição formal de um problema de classificação, faz-se necessário o entendimento do conceito de instância, também conhecido como exemplo de uma base de dados. De forma geral, os problemas de classificação são apresentados em bases de dados que são constituídas por uma série de instâncias. Cada instância é caracterizada por um conjunto finito de elementos denominados atributos Existem vários tipos de atributos, contudo, eles se organizam em duas categorias:

- **Atributos Discretos:** São atributos que assumem seus valores de um conjunto finito de valores.
- **Atributos Contínuos:** São atributos que assumem valores numéricos de qualquer tipo, sem nenhuma restrição.

De forma geral, um problema de classificação pode ser definido como (Bacardit, 2003):

Definição 3.1 (Problema de Classificação PC): Dado um conjunto finito de n instâncias $I = \{i_1, i_2, ..., i_n\}$ rotulados por um conjunto finito de m classes $C = \{c_1, c_2, ..., c_m\}$, um problema de classificação PC é a tarefa de se criar uma teoria T baseada em I e C de forma que, dada uma instância não rotulada, T forneça a predição da classe para esta instância.

Tipicamente, se um problema de classificação apresenta apenas duas classes, uma classe, a que será aprendida, é denominada de classe positiva e consequentemente as instâncias desta classe são denominadas de exemplos positivos. A classe que não será aprendida é denominada de classe negativa e é constituída por instâncias negativas.

Dada à definição de problema de classificação, pode-se definir, portanto, o conceito de algoritmo de aprendizado para um problema de classificação. Assim:

Definição 3.2 (Algoritmo de Aprendizado[1] Supervisionado AAS para um PC): Dado um problema de classificação PC que possui um conjunto finito de n instâncias $I = \{i_1, i_2,..., i_n\}$ rotulados por um conjunto finito de m classes $C = \{c_1, c_2,..., c_m\}$, um algoritmo de aprendizado supervisionado AAS é o procedimento aplicado em PC para se encontrar uma teoria T baseada em I e C de forma que, dada uma instância não rotulada, T forneça a predição da classe para esta instância.

Em outras palavras, dado um problema de classificação PC e um algoritmo de aprendizado supervisionado AAS, aplica-se AAS em PC para a obtenção de uma teoria T que será usada para predizer instâncias não rotuladas presentes em PC.

Dadas as definições 3.1 e 3.2, o problema de classificação é graficamente representado na figura 3.1.

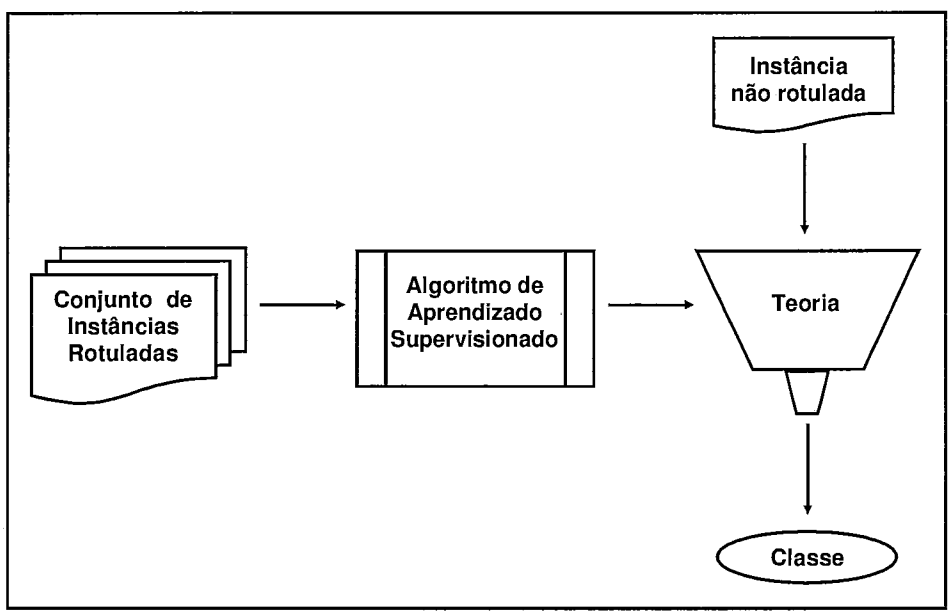

Figura 3.1: Esquema de Problemas de Classificação

1. É importante fazer a distinção entre algoritmo de aprendizado supervisionado e não supervisionado. Segundo (Russell e Norvig, 1995) o aprendizado supervisionado envolve o aprendizado de uma função a partir de instâncias de suas entradas e saídas. Já o aprendizado não supervisionado, envolve a aprendizagem de padrões na entrada quando não são fornecidos valores de saída específicos.

A figura 3.1 ilustra também as duas fases principais de um algoritmo de aprendizado. Estes estágios são conhecidos como etapa de treinamento e etapa de teste.

Cada uma destas etapas utiliza um subconjunto do conjunto de instâncias. O subconjunto utilizado pela etapa de treinamento é chamando de conjunto de treinamento, e é constituído pelas instâncias rotuladas do problema. O subconjunto usado na etapa de teste é designado de conjunto de teste. Este subconjunto é constituído pelas instâncias não rotuladas do problema..

A etapa de treinamento tem o objetivo de construir uma teoria que mapeie as instâncias do conjunto de treinamento em suas respectivas classes. A etapa de teste, por sua vez, tem o papel de rotular as instâncias do conjunto de teste com uma das classes presentes no conjunto de treinamento. Assim, a teoria construída no conjunto de treinamento é utilizada para classificar as instâncias do conjunto de teste. Caso a teoria tenha sido construída de forma correta, a taxa de instâncias corretamente preditas no conjunto de treinamento deverá ser igual ou ligeiramente mais alta que a taxa de predição obtida no conjunto de teste. A taxa de predição alcançada em um determinado conjunto de instâncias é conhecida como precisão (*accuracy*).

Essa seção apresentou de forma geral o problema de classificação. A próxima seção discute algumas formas de representação de uma teoria.

3.2 Representação de uma Teoria por Meio de Regras

Essa seção discute a representação de uma teoria por meio de regras de decisão. Várias outras formas de se representar uma teoria foram propostas e são amplamente usadas, contudo, não é objetivo deste trabalho analisar e discutir todos os modelos que são usados para a representação de uma teoria.

A representação de uma teoria por meio de regras de decisão foi à abordagem escolhida para a construção do AG para proposto neste trabalho. Essa forma de representação foi escolhida devido ao fato de que regras de decisão geralmente representam o conhecimento de forma simples e direta. Contudo, não devem ser esquecidas outras formas de representação de teorias, cada uma apresentando seus pontos positivos e negativos. Alguns dos exemplos mais clássicos para a representação de uma teoria são: redes neurais (McCulloch e Pitts, 1943), redes bayesianas (Pearl, 1988), k vizinhos mais próximos (Aha e Kibler, 1991) Maquinas de Vetores Suporte (Boser et al, 1992) etc.

3.2.1 Regras de Decisão: Forma Geral

Regras de decisão são provavelmente a maneira mais antiga e mais simples de representação de conhecimento. Elas podem ser representadas nas mais diversas linguagens e geralmente utilizam diversas formas de codificação. De maneira geral, uma regra de decisão possui a sintaxe apresentada na figura 3.2.

Figura 3.2: Sintaxe de Regra de Decisão

Em problemas de classificação, a parte da condição avalia os valores dos atributos da instância enquanto que a parte da ação determina a qual classe a instância pertence. Em outras palavras, uma regra de decisão prediz uma classe (ação) para a instância que torne a parte da condição verdadeira. Uma vez discutida a sintaxe geral de uma regra de decisão, é importante realizar a distinção entre a linguagem e a codificação que são usadas para representação de uma regra de decisão.

I. Linguagem para Representação de Regras de Decisão

A linguagem escolhida para a representação de uma regra de decisão determina de que forma a regra será estruturada bem como a sua interpretação. De modo geral, existem três tipos de linguagens para representação de uma regra de decisão: Forma Normal Conjuntiva (FNC), Forma Normal Disjuntiva (FND) e Forma Normal Disjuntiva Modificada (FNDM) (Michalski, 1983).

A. Forma Normal Conjuntiva (FNC)

Considere que as instâncias de um problema de classificação sejam descritas por n atributos. Uma regra de decisão está na FNC se e somente se a parte de sua condição está na forma:

$\alpha_1 \wedge \alpha_2 \wedge ... \vee \alpha_k$, $k \geq 1$, em que α_k está na forma:

$(A_1 = V_1 \vee A_2 = V_2 \vee ... \vee A_n = V_n)$,

em que Vi ($1 \leq i \leq n$) representa algum valor do conjunto de valores que o atributo Ai pode assumir. Note que apesar da definição exposta, uma regra de decisão não precisa necessariamente utilizar todos os atributos que definem uma instância.

B. Forma Normal Disjuntiva (FND)

Considere que as instâncias de um problema de classificação sejam descritas por n atributos. Uma regra de decisão está na FND se e somente se a parte de sua condição está na forma:

$\alpha_1 \vee \alpha_2 \vee ... \vee \alpha_k$, $k \geq 1$, em que α_k está na forma:

$(A_1 = V_1 \wedge A_2 = V_2 \wedge ... \wedge A_n = V_n)$,

em que Vi ($1 \leq i \leq n$) representa algum valor do conjunto de valores que o atributo Ai pode assumir.

C. Forma Normal Disjuntiva Modificada (FNDM)

Esta forma normal foi proposta em (Michalski, 1983). Ela é muito parecida com a forma normal disjuntiva, entretanto ela permite, adicionalmente, que sejam realizadas disjunções entre os valores dos próprios atributos. A FNDM permite uma maior compactação para a representação de regras e por isso é utilizada na maioria dos sistemas que usam um AG para aprendizado de regras. Formalmente, a FNDM pode ser definida da seguinte forma:

Considere que as instâncias de um problema de classificação sejam descritas por n atributos. Uma regra de decisão está na FNDM se e somente se a parte de sua condição está na forma:

$\alpha 1 \vee \alpha 2 \vee ... \vee \alpha_k$, $k \geq 1$, em que α_k está na forma:

$((A_1 = V_1^1 \vee ... \vee A_1 = V_1^m) \wedge ... \wedge (A_n = V_n^1 \vee ... \vee A_n = V_n^j))$,

em que Vi^p ($1 \leq i \leq n$) representa um valor p do conjunto de valores que o atributo Ai pode assumir.

D. Exemplos de Regras com FNC, FND e FNDM

Com objetivo de elucidar as principais linguagens para a representação de uma regra de decisão, esta seção apresenta um exemplo de uma regra de decisão para cada uma das linguagens apresentadas. As regras serão estabelecidas sobre o problema apresentado no exemplo 3.1.

Exemplo 3.1: Considere o problema de se classificar um cão em bonito ou feio. Cada cão é descrito através de três atributos: pêlo (curto, médio ou longo), tamanho (pequeno, médio ou grande) e peso, que varia entre 10 kg e 50 kg. Algumas regras de decisão, na FNC, FND e FNDM poderiam ser dadas por:

- Regra na FNC:

Se ((pêlo = curto ou tamanho = pequeno) e (peso ∈ [10, 15])) **então** cão é bonito.

- Regra na FND:

Se ((pêlo = longo e tamanho = grande) ou (peso ∈ [45, 50])) **então** cão é feio.

- Regra na FNDM:

Se (((pêlo = médio ou pêlo = longo) e (tamanho = médio ou tamanho = grande)) ou (peso ∈ [45, 50])) **então** cão é feio.

É importante notar, como afirmado, que nem todos os atributos precisam ser necessariamente usados pelas regras.

II. Codificação para Representação de Regras de Decisão

Uma vez apresentadas as principais linguagens para a representação de uma regra, esta seção apresenta alguns tipos de codificações adotadas para a estruturação da mesma. Mais precisamente, esta seção apresenta algumas formas de codificação dos atributos que constituem uma regra, portanto, ela será dividida em codificação para atributos discretos e contínuos. Todos os tipos de codificação apresentados nesta seção serão ilustrados com problema apresentado no exemplo 3.1, estando as regras representadas na FNDM.

A. Codificação de Atributos Discretos

A codificação binária (Dejong et al, 1993) para atributos discretos é a mais amplamente usada e difundida. Entretanto, em (Aguilar-Ruiz et al, 2002) foi apresentada a Codificação Natural. Estas duas formas de codificação serão apresentadas a seguir.

- **Codificação Binária:**

A codificação binária utiliza um bit para cada valor que cada atributo pode assumir. Caso o valor deste bit seja 1, o valor do atributo que o bit representa é usado, caso seja 0, o valor do atributo não é utilizado na regra. A figura 3.3 ilustra o caso de uma regra

construída utilizando apenas os atributos discretos do problema apresentado no exemplo 3.1. Lembre-se que a linguagem utilizada foi a FNDM. Esta regra diz que os cachorros que tem pelo = médio ou longo e tamanho = pequeno ou grande são bonitos.

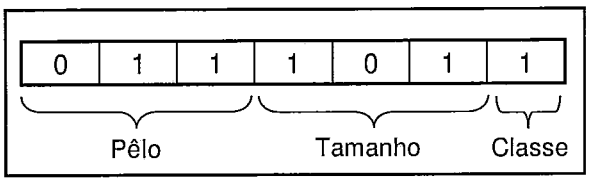

Figura 3.3: Representação Binária para Atributos Discretos

- **Codificação Natural**

Como apresentado, esta codificação foi proposta em (Aguilar-Ruiz et al, 2002). A Codificação Natural representa os atributos por meio de números naturais. Assim, dado um atributo A_i, com $|A_i|$ representando a quantidade de valores que A_i pode assumir, o número natural que representa este número pertencerá ao intervalo $[0, 2^{|A_i|} - 1]$. A figura 3.4 exibe a mesma regra apresentada na figura 3.3, porém com a utilização da Codificação Natural.

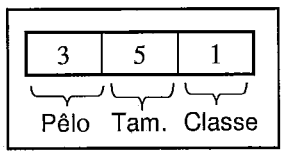

Figura 3.4: Representação Natural para Atributos Discretos

Note que a Codificação Natural codifica os números na representação binária em sua forma natural. Assim, o atributo peso que na codificação binária era representado pela cadeia de 011, é representado pelo número natural 3 na Codificação Natural, enquanto que o atributo tamanho, que era representado pela cadeia 101 na codificação binária, é representado pelo número natural 5 na Codificação Natural.

B. Codificação de Atributos Contínuos

As codificações de atributos contínuos seguem geralmente dois padrões: representação de intervalos por meio de dois números reais para cada atributo, ou a representação por meio da discretização destes atributos. Esta seção apresenta estes dois padrões principais, sendo que a representação de atributos discretizados será dada pela Codificação Natural. Mais uma vez, é importante lembrar que as codificações serão ilustradas com o uso do problema apresentado no exemplo 3.1, utilizando apenas o atributo peso, já que este é o único atributo contínuo que o problema apresenta.

- **Codificação por Intervalos**

Esta é a forma mais direta e intuitiva de se representar um atributo contínuo. Ela consiste no uso de dois números reais que irão representar o intervalo do atributo contínuo. A figura 3.4 exibe a codificação do atributo peso por meio de intervalos. Esta regra diz que os cachorros que tem peso entre 10,5 e 15,8 são considerados bonitos

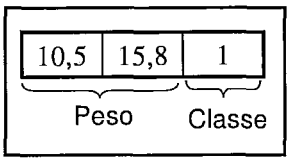

Figura 3.5: Representação por Intervalos para Atributo Peso

- **Codificação Natural**

O atributo contínuo deve ser discretizado por qualquer algoritmo de discretização. Uma vez discretizado, o algoritmo de discretização retorna um vetor de cortes que fornece os intervalos de discretização obtidos. Estes intervalos são então numerados, e estes números, agora representado intervalos, são utilizados pela a Codificação Natural. A tabela 3.1 ilustra os pontos de corte obtidos por algum algoritmo de discretização para o atributo peso do exemplo 3.1, bem como os intervalos que foram obtidos e os números naturais que os representam e que são utilizados pela Codificação Natural.

Tabela 3.1: Intervalos de Discretização para o Atributo Peso

Pontos de Corte	10	20	35	40
5	**1**	**2**	**3**	**4**
10	-	**6**	**7**	**8**
20	-	-	**11**	**12**
35	-	-	-	**16**

A tabela 3.1 informa que o intervalo [5, 10] é codificação pelo número natural 1, que o intervalo [5, 20] é codificado pelo número natural 2 e assim por diante. A figura 3.6 exibe uma regra utilizando o atributo peso com o uso da Codificação Natural. Esta regra diz que os cachorros que tem peso entre 10 e 20 são considerados bonitos

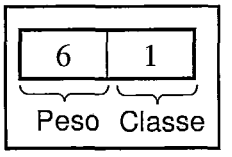

Figura 3.6: Representação Natural para Atributo Peso

3.2.2 Quantidade de Regras de Decisão numa Teoria

Pela seção anterior, pode-se concluir que o entendimento de uma teoria representada por regras de decisão depende diretamente da linguagem e da codificação adotadas para a representação destas regras. É importante notar, contudo, que na maior parte das vezes uma teoria não é composta por apenas uma regra de decisão, e sim por um conjunto destas. Desta forma, não somente o entendimento da teoria é influenciado pela quantidade de regras que a compõem, mas também, deve-se decidir qual a política será adotado para a classificação de uma instância no caso da teoria ser composta por múltiplas regras. Algumas das políticas mais comuns são descritas em seguida.

Lista de Decisão: Esta política para classificação foi originalmente proposta por Rivest em (Rivest, 1987). De maneira geral, as regras são ordenadas de forma hierárquica e a primeira regra que apresentar a sua condição satisfeita pela instância, será usada para classificação.

Heurística: Essa política considera a performance das regras que foram previamente usadas. Caso uma regra de classificação tenha sido usada, sabe-se, dentre outras métricas, a sua precisão e quantas vezes ela foi utilizada. As regras são ordenadas de acordo com essas métricas e o conflito para a classificação de uma instância é resolvido pela posição em que as regras ocupam nesta ordenação. Para uma descrição desta política veja (Fürkranz, 1999).

Votação: Neste tipo de política combinam-se as saídas de todas as regras que cobrem uma determinada instância. O exemplo mais comum de votação determina que a instância seja classificada pela classe majoritária das regras que cobrem a instância.

3.3 Sistemas para Aprendizado de Regras

Existem diversos tipos de sistemas para aprendizado de regras, contudo esta seção apresenta dois dos sistemas clássicos que apresentam essa finalidade. Os dois sistemas escolhidos foram o de árvore de decisão C4.5 (Quinlan, 1993), e o RIPPER (Cohen, 1995). Estes sistemas foram selecionados devido a sua ampla utilização nos mais diversos problemas.

Esta seção não tem o objetivo de descrever todos os detalhes dos sistemas, mas sim de apresentar de forma geral como teorias representadas por regras de decisão podem ser aprendidas.

3.3.1 Árvore de Decisão

As árvores de decisão classificam as instâncias seguindo um caminho de condições satisfeitas da raiz da árvore até uma folha, que representa o valor da classe. Os nós rotulados com os valores da classe são chamados de nós folha enquanto que os outros nós são denominados de nós de decisão. Cada nó de decisão da árvore especifica um teste para algum atributo da instância, e cada ramo descendente deste nó corresponde a um dos possíveis valores deste atributo. Uma instância é classificada começando no nó raiz, testando o atributo especificado neste nó, então movendo-se abaixo nos ramos da árvore que correspondem aos valores dos atributos na instância dada.

A figura 3.7 ilustra uma típica árvore de decisão aprendida para o problema de classificação de cachorros (exemplo 3.1). A árvore de decisão apresentada classifica a instância descrita por [pêlo = curto, tamanho = pequeno e peso = 25 kg] como feia.

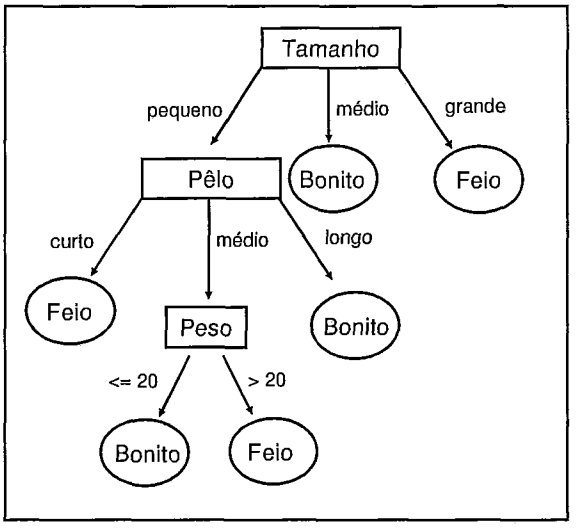

Figura 3.7: Árvore de Decisão para o Problema de Cachorros

I. O C4.5

O sistema C4.5 (Quinlan, 1993) é uma versão descendente do algoritmo ID3 (Quinlan 1986) que aprende árvores de decisão pela construção *top-down*, começando com a pergunta: "Qual o atributo deve ser testado na raiz da árvore?" Para responder esta questão, cada atributo da instância é avaliada usando um teste estatístico que determina o quão bem ele sozinho classifica as instâncias de treinamento. O melhor atributo é selecionado e usado como teste no nó raiz da árvore. Um descendente do nó raiz é então criado para cada valor possível deste atributo, e os exemplos de treinamento são encaminhados ao nó descendente apropriado (i.e., ao ramo correspondente ao valor do atributo para a instância). Esse procedimento é então repetido usando as instâncias de treinamento associadas com cada nó descendente para selecionar o melhor atributo para testar neste ponto da árvore (Mitchell, 1997). Em seguida é apresentado a forma geral do algoritmo ID3: este mesmo esquema de aprendizado é seguido pelo C4.5.

Algoritmo 3.1 – Algoritmo ID3

Entrada:
instâncias: conjunto de instâncias de treinamento;
list_atrib: conjunto de atributos candidatos a teste;
Saída:
Árvore de Decisão

```
Começo:
1 - crie um nó N;
2 - Se (instâncias são todas da classe C) então
    3 - retorne N como nó folha rotulada com a classe C;
    senão
    4 - Se (list_atrib estiver vazia) então
        5 - retorne N como nó folha rotulada com a classe mais comum
            entre as instâncias;
        senão
        6 - selecione o atributo de teste, isto é, o atributo da list_atrib
            com o maior ganho de informação[2];
        7 - rotule o nó N com o atributo de teste;
        8 - Para cada valor possível ai do atributo de teste faça
            9 - inclua um ramo a partir do nó N, com a condição
                "atributo de teste = ai";
            10 - atribua a si o subconjunto de instâncias contido em
                 instâncias que possuem o atributo de teste = ai;
            11 - Se (si estiver vazia) então
                 12 - inclua uma folha rotulada com a classe mais
                      comum entre as instâncias;
                 senão
                 13 - exclua o atributo de teste de list_atrib;
                 14 - inclua o nó retornado;
fim;
```

2. O Ganho de informação mede o quão bem um dado atributo separa os exemplos de treinamento de acordo com as suas classes. O ganho de informação utiliza o conceito de entropia. Para uma definição precisa e formal destes conceitos, veja páginas 55 a 57 de (Mitchell, 1997).

II. Representação de Regras em Árvores de Decisão

Pela árvore exposta na figura 3.7, é possível perceber que as regras de decisão representadas pela árvore adotam a FND. Assim, alguns exemplos de regras retiradas desta árvore são dadas por:

Se ((tamanho = médio) então cachorro = bonito.

Se ((Tamanho = pequeno) e (Pêlo = Longo)) então cachorro = bonito.

Se ((Tamanho = pequeno) e (Pêlo = Médio) e (Peso <= 20)) então cachorro = bonito.

Note que estas regras estão de fato na FND, já que todas apresentam uma conjunção de atributos embora não apresentem disjunções. Este é o caso mais simples da FND, em que k = 1 (veja novamente a seção 3.2.1 – I – B).

III. Superadaptação (*overfitting*) e Poda em Árvores de Decisão

O algoritmo 3.1 cresce cada ramo da árvore de forma suficiente a classificar perfeitamente todos os exemplos de treinamento. Algumas vezes isso parece uma boa estratégia, entretanto, isso pode levar a certas dificuldades quando existe ruído nos dados ou quando o conjunto de treinamento é muito pequeno para produzir um amostra representativa da função de classificação a ser aprendida.

Dize-se que uma teoria está super-adaptada a um conjunto de treinamento, se alguma outra teoria que se adapte menos a este conjunto descreva melhor a função de classificação do que a teoria super-adaptada. Formalmente, a superadapatação pode ser definida como (Mitchell, 1997):

Definição 3.3 (Superadaptação): Dado um espaço de teorias T, uma teoria $t \in T$ é dita que está super-adaptada ao conjunto de treinamento, se existe outra teoria alternativa $t' \in T$, de tal forma que t possua um erro menor que t' sobre o conjunto de treinamento, mas t' possua um menor erro que t se for considerado todo o conjunto de distribuição das instâncias.

Superadaptação é problema significante para o aprendizado de árvores de decisão e para vários outros métodos de aprendizado. Existem várias abordagens para evitar superadaptação no aprendizado de árvores de decisão, contudo, estes métodos podem ser agrupados em duas classes (Mitchell, 1997):

- Abordagens que param o crescimento da árvore mais cedo, antes de ser atingido o ponto em que a teoria classifica perfeitamente o conjunto de treinamento.
- Abordagens que permitem a árvore se super-adaptar aos dados, para então podá-la posteriormente (*post-prune*).

Embora a primeira destas abordagens pareça mais intuitiva, a abordagem de poda posterior parece ter mais sucesso na prática (Mitchell, 1997). Isso se deve a dificuldade na primeira abordagem em se estimar precisamente quando a árvore deve parar de crescer.

Desconsiderando se o tamanho correto da árvore é encontrado pelo uso de qualquer uma das duas abordagens acima expostas, uma pergunta importante se relaciona a qual critério é usado para determinar o tamanho correto da árvore final. Alguns destes critérios são:

- Usar um conjunto de exemplos separado, distinto do conjunto de exemplos de treinamento, para avaliar a abordagem de poda posterior de nós.
- Usar todo o conjunto disponível de exemplos de treinamento, mas aplicar testes estatísticos para estimar se a expansão ou poda de um nó particular é provável de produzir uma melhora além do conjunto de treinamento.

A primeira abordagem é a mais comum e é chamada, na maioria das vezes, de abordagem de treinamento e validação. Nesta abordagem, todo o conjunto de instâncias rotuladas é separado em dois conjuntos de exemplos: o conjunto de treinamento, que como apresentado, é usado para o aprendizado da teoria, e o conjunto de validação, que é usado para avaliar a precisão desta teoria em um conjunto subseqüente de dados e, em particular, para avaliar o impacto de poda desta teoria. A motivação para o uso desta abordagem é a seguinte: embora o algoritmo de aprendizado possa ser influenciado negativamente por erros aleatórios no conjunto de treinamento, é improvável que o conjunto de validação exiba estes mesmo erros. Assim, espera-se que o conjunto de validação forneça um teste de segurança contra a superadaptação às características espúrias do conjunto de treinamento. Obviamente, é importante que o conjunto de validação seja grande o suficiente para fornecer uma amostra estatisticamente significante do conjunto de instâncias. Duas variações principais desta abordagem, denominadas de Poda de Erro Reduzido e Poda Posterior de Regra serão discutidas em seguida.

A. Poda de Erro Reduzido (*Reduced Error Pruning*)

Uma das abordagens em que se usa o conjunto de validação para prevenir a superadaptação é clamada de Poda de Erro Reduzido (Quinlan, 1993), que considera cada um dos nós de decisão dá árvore como sendo candidatos à poda. Podar um nó de decisão consiste em remover a subárvore enraizada neste nó, fazendo dele um nó folha e atribuindo a ele a classe mais comum das instâncias treinamento afiliadas aquele nó. Os nós são removidos apenas se o resultado da árvore podada não for pior que o resultado alcançado pela a árvore inteira sobre o conjunto de validação. Os nós são podados iterativamente; sempre escolhendo primeiro os nós cujas podas irão acrescer ao máximo a precisão obtida sobre o conjunto de validação. A poda de nós continua até que próxima poda seja prejudicial, i.e., reduza a precisão da árvore sobre o conjunto de validação. Obviamente, a poda de reduza a precisão da árvore não é executada.

B. Poda Posterior de Regra (*Rule Post Pruning*)

Na prática, um método bastante útil de poda para se encontrar teorias com alta precisão denomina-se Poda Posterior de Regra. Uma variação deste método é usado no C4.5. De forma geral, a Poda Posterior de Regra envolve os seguintes passos (Mitchell, 1997):

1. Infira a árvore de decisão a partir do conjunto de treinamento, crescendo a árvore até que ela esteja completamente adaptada a este conjunto, permitindo que a superadaptação ocorra.
2. Converta a árvore de decisão em um conjunto equivalente de regras de decisão pela criação de uma regra para cada caminho do nó raiz até um nó folha.
3. Pode (generalize) cada regra pela remoção de quaisquer precondições que resultem numa melhoria de sua precisão estimada sobre o conjunto de validação.
4. Ordene as regras podadas pela estimativa de suas precisões, e considere-as nesta seqüência quando a classificação de instâncias subseqüentes ocorrer.

Considere o exemplo a seguir que ilustra o processo de Poda Posterior de Regra.

Exemplo 3.2: No tópico II desta seção, foram exibidas diversas regras construídas a partir da árvore de decisão apresentada na figura 3.7 para o problema de classificação de cachorros. Considere a regra dada por:

Se ((Tamanho = pequeno) e (Pêlo = Médio) e (Peso <= 20)) **então** cachorro = bonito.

Cada teste de atributo ao longo do caminho da raiz até um nó folha é considerado uma pré-condição. Assim, para a regra acima mostrada, as pré-condições são: (Tamanho = pequeno), (Pêlo = Médio), (Peso <= 20). Dessa forma, essa regra é podada pela deleção de alguma pré-condição cuja remoção melhore ao máximo a precisão estimada para esta regra sobre o conjunto de validação. O processo de remoção de pré-condições continua até que a próxima remoção cause um decréscimo na precisão estimada para esta regra, e dessa forma, obviamente, esta última remoção não é realizada.

Esta seção apresentou dois dos principais métodos de poda, contudo, outros foram propostos em (Endou e Zhao, 2002) e (Cano et al, 2004).

IV. Principais Parâmetros

fatConf: Fator de confidência usado para a poda, de modo geral, valores pequenos implicam em mais poda.

numMinObj: Número mínimo de instâncias que deve conter um nó folha para este ser criado.

numFolds: Determina a quantidade de dados que é usada para a poda de erro reduzido. Um fold é usado para a poda, o restante para o crescimento da árvore.

podaErroReduzido: Determina se a poda de erro reduzido é usada em vez de uma variação de poda posterior de regra.

crescimento: se é considerado o crescimento de uma sub-árvore quando a operação de poda é efetuada.

3.3.2 O RIPPER

O sistema RIPPER (Cohen, 1995) é uma extensão do sistema IREP (*Incremental Reduced Error Pruning*) que foi proposto em (Fürnkranz e Widme, 1994). O IREP foi proposto como alternativa a complexidade computacional do REP (*Reduced Error Pruning*) que exigia uma enorme quantidade de tempo para ser executado em grandes bases de dados ou mesmo para bases de dados que apresentassem alta quantidade de

ruído (Cohen, 1993). Dessa forma, essa seção introduz primeiramente o sistema IREP para posteriormente explicar os novos componentes a ele adicionados para a formação do RIPPER. Ates, porém, se faz necessária a explicação de um típico algoritmo de cobertura para aprendizado de teorias representadas por regras de decisão, já que este algoritmo é a parte essencial do IREP e consequentemente do RIPPER.

I. Típico Algoritmo de Cobertura para Aprendizado de Regras

A idéia geral de todos os algoritmos de cobertura para problemas de classificação pode ser resumida nos seguintes passos:

1. Aprenda uma regra que classifique certa quantidade instâncias positivas[3].
2. Remova as todas as instâncias (positivas e negativas) classificadas pela regra.
3. Repita passos 1 e 2 até que todas as instâncias positivas estejam classificadas ou até que outro critério de parada seja satisfeito.

Em seguida, é apresentado os principais passos de um algoritmo de cobertura.

Algoritmo 3.2 – Típico Algoritmo de Cobertura

```
Entrada:
   (i+, i-): conjunto de instâncias de treinamento formadas por instâncias
         positivas i+ e por instâncias negativas i-;
Variáveis Locais:
   conjunto_regras;
Saída:
   conjunto_regras modificado;

Começo:
1 – conjunto_regras ← ∅;
2 – regra ← Aprenda-Regra((i+,i-));
3 – Enquanto ( |i+| > 0 ) faça
    4 – conjunto_regras ← conjunto_regras + regra;
    5 – (i+,i-) ← (i+,i-) - instâncias classificadas por regra;
    6 – regra ← Aprenda-Regra((i+,i-));
    fim-enquanto;
7 – ordene o conjunto de regras baseado em suas precisões;
8 – retorne conjunto_regras;
Fim;
```

3. Para facilitar o entendimento, o algoritmo de cobertura será apresentado em termos do problema de classificação para duas classes, contudo, este algoritmo pode ser facilmente expandido para lidar com problemas que apresentem mais classes.

II. O Sistema IREP

O sistema IREP de aprendizado de regras é descrito em detalhes em (Fürnkranz e Widme, 1994), contudo, essa seção destina-se a apresentar de forma resumida este sistema.

O IREP integra a técnica de poda de erro reduzido com o algoritmo de cobertura. O algoritmo 3.3 exibe os passos principais deste sistema.

Algoritmo 3.3 - IREP

Entrada:
(i+, i-): conjunto de instâncias de treinamento formadas por instâncias positivas i+ e por instâncias negativas i-;
Variáveis Locais:
conjunto_regras;
Saída:
conjunto_regras modificado;

```
Começo:
1 – conjunto_regras ← ∅;
3 – Enquanto (|i+| > 0) faça
    4 – Divida (i+, i-) em (Cresce_i+, Cresce_i-) e (Poda_i+, Poda_i-);
    5 – regra ← Aprenda-Regra(Cresce_i+, Cresce_i-);
    6 – regra ← Poda-Regra(regra, Poda_i+, Poda_i-);
    7 – Se (taxa de erro de regra em (Poda_i+, Poda_i-) > 50%) então
        8 – retorne conjunto_regras;
        9 – Finalize-Algoritmo;
        senão
        10 – conjunto_regras ← conjunto_regras + regra;
        11 – remova as instâncias cobertas pela regra de (i+, i-);
        fim-se;
    fim-enquanto;
12 – retorne conjunto_regras;
Fim;
```

Como um típico algoritmo de cobertura, o IREP constrói um conjunto de regras adicionando uma regra de cada vez a este conjunto. Após uma regra ter sido encontrada, todas as instâncias classificados pela regra (positivas e negativas) são removidas do conjunto de instâncias. Esse processo é repetido até que não exista nenhuma instância positiva, ou até que a regra encontrada pelo IREP possua uma taxa de erro inaceitável.

Para construir uma regra, o IREP usa a seguinte estratégia: primeiro (passo 4), as instâncias não classificadas são particionadas em dois subconjuntos, um conjunto de crescimento (Cresce_i+, Cresce_i-) e um conjunto de poda (Poda_i+, Poda_i-). Geralmente, o conjunto de crescimento contém 2/3 das instâncias. No passo 5, a regra é aprendida da seguinte maneira: Inicialmente, a regra começa com um conjunto vazio de

conjunção de condições, e então condições da forma An = v, Ac $\leq \theta$, Ac $\geq \theta$, onde An é um atributo discreto e n é um valor que An pode assumir e Ac é um atributo contínuo e θ é algum valor que Ac pode assumir, são a ela adicionadas. O procedimento Aprenda-Regra() é uma versão proposicional do FOIL (Quinlan e Cameron-Jones, 1993), assim, este procedimento repetidamente adiciona condições que maximizem o critério de ganho de informação do FOIL até que a regra não cubra nenhum exemplo negativo do conjunto de crescimento.

Após o aprendizado da regra, esta é imediatamente podada. Para podar a regra, é considerada a remoção de qualquer seqüência de condições. A remoção escolhida é a que maximiza a função:

$$\text{v(regra, Poda\_i+, Poda\_i-)} \equiv \frac{p+(N-n)}{P+N} \tag{3.1}$$

em que P e N representam respectivamente, o número total de instâncias em Poda_i+ e Poda_i-, e p e n representam, respectivamente, o número de instâncias positivas e negativas presentes em Poda_i+ e Poda_i-, que foram classificadas pela regra. O processo de poda é repetido até que nenhuma remoção melhore o valor de v.

III. O Sistema RIPPER

Como afirmado, o sistema RIPPER é uma evolução do sistema IREP. Dessa forma, o sistema RIPPER é definido como sendo o sistema IREP acrescido de duas modificações principais:

- Uma métrica alternativa para o calculo de v (equação 3.1).
- Um procedimento que "otimiza" um conjunto de regras na tentativa de aproximar o método convencional (i.e não incremental) de poda de erro reduzido.

Cada uma destas modificações será explicada em seguida.

A. Métrica do Valor de uma Regra

Para problemas que possuem um grande número de instâncias, o IREP mostrou problemas de convergência que foram provados ser devido à função usada para guiar a poda (Cohen, 1995). Portanto, uma nova função foi construída. Essa função é exibida em seguida:

$$v(\text{regra, Poda\_i+, Poda\_i-}) \equiv \frac{p-n}{p+n} \qquad (3.2)$$

em que e p e n representam, respectivamente, o número de instância positivas e negativas presentes em Poda_i+ e Poda_i-, que foram classificadas pela regra.

B. Otimização de Regra

A abordagem de aprender e podar usada no IREP produz resultados totalmente diferentes da poda de erro reduzido convencional (não incremental). Uma maneira de melhorar a abordagem incremental do IREP é a de pós-processar as regras produzidas pelo IREP de forma a aproximar o método convencional de poda de erro reduzido.

O processo de otimização ocorre sobre um conjunto de k regras {R1, ..., Rk) e pode ser descrito da seguinte maneira: Cada regra deste conjunto é considerada na ordem em que foi aprendida: primeiro R1, depois R2 e assim por diante. Para cada regra Ri ($1 \leq i \leq k$) duas regras alternativas são construídas. A regra de substituição de Ri, denominada de SRi, é aprendida e depois podada de forma minimizar o erro do conjunto de regras {R1, ..., SRi, ..., Rk) sobre o conjunto de poda. A regra de revisão de Ri, denominada RRi, é formada de forma análoga, exceto que RRi é formada por todas as condições de Ri acrescidas de outras condições. A adição de condições a RRi é realizada de forma a minimizar o erro do conjunto de regras {R1, ..., RRi, ..., Rk) sobre o conjunto de poda. Finalmente, é decido se o conjunto final de regras (teoria) deverá incluir a regra de substituição (SRi), a regra de revisão (RRi) ou a regra original (Ri).

A otimização é integrada o IREP da seguinte forma: Primeiro, o IREP é usado para obter um conjunto de regras. Este conjunto de regras é otimizado como explicado acima. Finalmente, algumas regras são adicionadas à teoria, usando o IREP, com objetivo de cobrir alguns exemplos positivos que ainda não foram cobertos.

IV. O Sistema RIPPERk

Como afirmado, o sistema RIPPER é constituído do sistema IREP acrescido das duas modificações acima explicadas. O sistema RIPPERk é obtido pelo uso do sistema RIPPER para a obtenção de um conjunto inicial de regras e pela utilização da fase de otimização k vezes sobre este conjunto de regras.

V. Principais Parâmetros

ChecarTaxaErro: Determina se a taxa de erro de 50% (passo 7) é usado como critério de parada.

numFolds: Determina a quantidade de dados que é usada para a poda de erro reduzido. Um fold é usado para a poda, o restante para o crescimento da árvore.

pesoMin: Peso mínimo de instâncias em uma regra. De forma geral, determina a quantidade de instâncias mínimas em uma regra para esta ser criada.

numOt: Número de otimizações realizadas no sistema.

usarPoda: Determina se a poda é ou não utilizada.

3.4 Algoritmos Genéticos e Evolução de uma Teoria

Tradicionalmente, os Algoritmos Genéticos têm sido aplicados a problemas de aprendizado de regras do ponto de vista de duas abordagens ou modelos que levam em conta o modo de desenvolvimento da teoria. Essas abordagens são conhecidas como Michigan e Pittsburgh. De Jong (De Jong, 1988) fornece uma descrição resumida destas duas abordagens:

"Para todos os que leram o trabalho de Holland (Holland, 1975), uma forma natural de proceder é representar um conjunto de regras como um indivíduo, manter uma população de conjunto de regras, e usar a seleção e os operadores genéticos para produção de novas gerações de conjuntos de regras. Historicamente, essa foi a abordagem usada por De Jong e seus alunos na universidade de Pittsburgh (Smith, 1980) e (Smith, 1983), o que fez surgir a definição de abordagem Pittsburgh.

Entretanto, durante o mesmo período, Holland desenvolveu um modelo de cognição (sistema de classificação) no qual cada membro da população (indivíduo) representa regras individuais e o conjunto de regras é representado pela população como um todo (Holland e Reitman, 1978). Esse modelo ficou rapidamente conhecido como abordagem Michigan."

Assim, existe a abordagem Pittsburgh que é mais simples e mais próxima do paradigma dos Algoritmos Genéticos em que cada um dos indivíduos é uma teoria e estes competem entre si por uma maior adaptação ao ambiente. Por outro lado, existe a abordagem Michigan que é mais complexa e na qual cada indivíduo da população representa uma única regra e a teoria é representada por todos os indivíduos da população.

Recentemente, surgiu uma terceira forma de evolução de uma teoria que combina a abordagem Michigan e Pittsburgh. Essa nova maneira ficou conhecida como abordagem Iterativa. Nesta abordagem, cada indivíduo representa uma regra (Michigan), porém a solução retornada pelo AG é um indivíduo (Pittsburgh) que é formado pela concatenação de indivíduos obtidos pelo uso do AG várias vezes.

A seguir, cada uma destas abordagens é ilustrada com um algoritmo que descreve, de maneira geral, seus principais passos.

3.4.1 A Abordagem Pittsburgh para Evolução de uma Teoria

Nesta abordagem, cada indivíduo representa uma teoria. Embora cada regra possua um tamanho fixo, o número de regras em um indivíduo é variável. Os indivíduos da população competem entre si com o objetivo de maximizar o número de exemplos de treinamento corretamente classificados. O principal sistema representativo desta abordagem é conhecido como GAssist e foi proposto em (Bacardit, 2000). A seguir, o algoritmo que resume a abordagem Pittsburgh é apresentado.

Algoritmo 3.4 – Abordagem Pittsburgh para Evolução de Teorias

Entrada:
arquivo de treinamento **Tr**;
população **pop** composta por **n** indivíduos em que cada indivíduo é composto por uma ou mais regras;
Saída:
melhor **indivíduo** da população que representa a teoria aprendida;

Começo:
1 – Inicialize pop de forma aleatória;
2 – Avalie cada indivíduo de pop com a função de aptidão em Tr;
3 – Enquanto (témino = falso) faça:
4 – Adicione regras aos Indivíduos de pop;
5 – pop' ← seleção de pais de pop;
6 – Aplique o operador de recombinação em alguns indivíduos de pop';
7 – Aplique o operador de mutação em alguns indivíduos de pop';
8 – Avalie cada indivíduo de pop' com a função de aptidão em Tr;
9 – pop ← Substituição (pop, pop');
10 – Remova regras dos Indivíduos de pop;
Fim-enquanto;
11 – retorne o melhor indivíduo em pop (o que possui maior valor de aptidão);
Fim:

De forma geral, a abordagem Pittsburgh apresenta dois passos adicionais ao ciclo básico de um AG. Neste algoritmo, estes passos são representados na linha 4 e linha 10. No passo 4, regras são adicionas aos indivíduos de pop, e no passo 10, algumas regras são retiradas de alguns indivíduos de pop para evitar que estes cresçam de maneira descontrolada e se tornem super-adaptados ao conjunto de treinamento. É importante perceber que embora o passo 10 seja geralmente implementado de forma externa aos operadores de um AG, o passo 4, que adiciona regras, pode ser implementado juntamente com o operador de recombinação. Isso ocorre no sistema GAssist que será posteriormente discutido.

3.4.2 A Abordagem Michigan para Evolução de uma Teoria

Nesta abordagem, cada indivíduo representa uma única regra e a teoria é construída por todos os indivíduos da população. Atualmente, o sistema mais importante que representa essa abordagem é conhecido como XCS (Wilson, 1995). Em seguida é apresentado o algoritmo que resume a abordagem Michigan.

Algoritmo 3.5 – Abordagem Michigan para Evolução de Teorias

Entrada:
arquivo de treinamento **Tr**;
população **pop** composta por no máximo **n** indivíduos em que cada indivíduo é composto por uma única regra;
Saída:
população **pop** modificada que representa a teoria aprendida;

```
Começo:
 1 – Inicialize pop de forma aleatória;
 2 – Avalie cada indivíduo de pop com a função de aptidão;
 3 – Enquanto (témino = falso) faça:
     4 – Se (número de indivíduos em pop < n) então
         5 – crie novo indivíduo e o avalie com a função de aptidão;
         6 – Adicione individuo criado a pop;
         fim-se;
     7 – pop' ← seleção de pais de pop;
     8 – Aplique o operador de recombinação em alguns indivíduos de pop';
     9 – Aplique o operador de mutação em alguns indivíduos de pop';
     10 – Avalie cada indivíduo de pop' com a função de aptidão;
     11 – pop ← Substituição (pop, pop');
     Fim-enquanto;
 12 – Retorne pop;
Fim:
```

De forma geral, a abordagem Michigan possui um valor limite que determina a quantidade máxima de indivíduos na população. Esta abordagem começa com um número de indivíduos inferior a este limite e adiciona novos indivíduos (regras) à população a cada iteração. Esse procedimento é ilustrado nos passos 4, 5 e 6 do algoritmo 3.5. Note que a abordagem retorna a população que representa a teoria aprendida, já que cada indivíduo da população representa uma única regra de decisão.

3.4.3 A Abordagem Iterativa para Evolução de uma Teoria

Na abordagem iterativa, um indivíduo representa uma única regra, entretanto, a solução é constituída pela concatenação de vários indivíduos evoluídos através de várias chamadas do Algoritmo Genético. Tipicamente, a abordagem iterativa utiliza um algoritmo de cobertura como procedimento principal. O sistema mais importante que representa este tipo de abordagem é conhecido como HIDER (Aguilar-Ruiz et al, 2003). Em seguida é apresentado o algoritmo que descreve os principais passos da abordagem iterativa.

Algoritmo 3.6 – Abordagem Iterativa para Evolução de Teorias

Entrada:
arquivo de treinamento **Tr**;
população **pop** composta de **n** indivíduos em que cada indivíduo é composto por uma única regra;
Saída:
teoria **T** formada por várias execuções do Algoritmo Genético;

```
Começo:
 1 – T ← ∅;
 2 – Enquanto (número de instâncias positivas em Tr > 0) faça
     3 – regra ← AG(Tr);
     4 – T ← T ∪ regra;
     5 – Remova_Exemplos_Cobertos(Tr, regra);
     Fim-enquato;
6 – Retorne T;
Fim;

Procedimento AG (Arquivo de Treinamento Tr): retorna um indivíduo (regra);
Começo-AG:
1 – Inicialize pop de forma aleatória;
 2 – Avalie cada indivíduo de pop com a função de aptidão em Tr;
 3 – Enquanto (témino = falso) faça:
     4 – pop' ← seleção de pais de pop;
     5 – Aplique o operador de recombinação em alguns indivíduos de pop';
     6 – Aplique o operador de mutação em alguns indivíduos de pop';
     7 – Avalie cada indivíduo de pop' com a função de aptidão em Tr;
     8 – pop ← Substituição (pop, pop');
     Fim-enquanto;
 9 – Retorne o melhor indivíduo em pop (o que possui maior valor de aptidão);
Fim-AG:
```

Como afirmado, a abordagem iterativa é controlada por um algoritmo de cobertura que chama o AG repetidas vezes com o objetivo de construir uma teoria. O passo 3 ilustra a chamada do AG sobre o conjunto de treinamento e o retorno do melhor indivíduo da população (uma regra) que será usado para a construção da teoria no passo 4. O passo 5 remove os exemplos do arquivo de treinamento que já foram classificados pela regra desenvolvida no passo 3. Ao fim da execução, uma teoria, constituída por varias regras obtidas por sucessivas chamadas do AG, é então obtida.

3.4.4 Comparação das Abordagens Pittsburgh, Michigan e Iterativa

Existe uma grande dificuldade em se comprarar as abordagens Pittsburgh, Michigan e Iterativa para evolução de uma teoria. Isso porque, cada uma das abordagens apresenta

apenas como será a evolução de uma teoria, e com isso, cada sistema implementado sobre cada abordagem é "livre" para escolher os outros componentes que diretamente determinam o seu comportamento. Estes componentes são: a função de aptidão, modo de seleção, tipo de operador de mutação e recombinação, política de adição e exclusão de regras, entre outros. Dessa forma, os resultados obtidos por um sistema não são apenas determinados pela escolha da abordagem para evolução da teoria, mas também pela escolha e implementação destes componentes. Contudo, de modo geral, a abordagem Pittsburgh apresenta um maior consumo de tempo e memória, já que várias teorias são representadas ao mesmo tempo e isso acarreta diretamente um maior consumo de memória e tempo de execução. A abordagem Michigan, por sua vez, apresenta um menor gasto de memória e tempo de execução, já que a população representa uma única teoria e dessa forma a manipulação se torna, de certa forma, mais eficiente. Contudo, de maneira geral, as teorias desenvolvidas pela abordagem Pittsburgh são menos complexas e suas precisões são bastante competitivas quando comparadas às precisões obtidas pela abordagem Michigan (Bacardit e Butz, 2004). Já a abordagem Iterativa, busca o meio termo entre as abordagens Pittsburgh e Michigan, tentando, dessa forma, alcançar bons resultados, sem, entretanto, necessitar de um grande consumo de memória e tempo de execução como ocorre na abordagem Pittsburgh. Como afirmado, entretanto, essas são apenas algumas considerações gerais sobre cada abordagem, já que o comportamento de cada sistema é drasticamente influenciado por outros componentes neles implementados.

Essa seção apresentou as principais abordagens utilizadas para a evolução de uma teoria quando se utiliza AG para aprendizado de regras. A próxima seção apresenta uma descrição dos principais sistemas que representam cada uma das abordagens descritas.

3.5 Principais Sistemas de Aprendizado Genético de Regras

Essa seção descreve três Algoritmos Genéticos aplicados ao problema de aprendizado de regras. Cada um dos sistemas descritos representa um modelo de evolução da teoria adotado, assim, a seção será divida em três partes. A primeira apresenta um sistema construído com a abordagem Pittsburgh, a segunda apresenta um sistema Michigan e a terceira apresenta um modelo Iterativo.

3.5.1 O Sistema GAssist – Abordagem Pittsburgh

O principal sistema representativo da abordagem Pittsburgh é conhecido como GAssist e foi proposto em (Bacardit, 2000). Esta seção descreve este sistema.

I. Representação do Conhecimento

As regras do GAssist são representada na Forma Normal Disjuntiva Modificada (FNDM), explicada na 3.2.1 deste capítulo. Para atributos discretos, a codificação binária, também explicada na seção 3.2.1 é usada. Para atributos contínuos, estes devem ser discretizados e é utilizada uma codificação denominada de Intervalos Discretizados Adaptativos (IDA) (Bacardit e Garrel, 2003). Os IDA usam o processo de discretização, porém de maneira especial: os intervalos de IDA não são estáticos e, portanto, eles são partidos e reunidos durante a execução do sistema. Estes intervalos são construídos utilizando o que Jaume e Garrel denominaram de "tijolos de baixo nível", que nada mais são que os pontos de corte retornados por algum algoritmo de discretização. Este mecanismo permite que vários algoritmos de discretização possam ser usados ao mesmo tempo, o que permite ao sistema escolher o algoritmo de discretização mais adequado para um dado problema.

II. Função de Aptidão

A função de aptidão é baseada no principio da Descrição de Mínimo Comprimento (DMC) (Rissanen, 1978). De forma resumida, o principio DMC é uma métrica usada para avaliar a complexidade e a precisão da teoria. Assim, a função de aptidão que deve ser minimizada, dirige a busca por soluções que sejam simples, mas que possuam alta precisão. A equação 3.3 exibe a função de aptidão usada pelo GAssist.

$$f(x) = (W * \sum_{i=1}^{nr} \sum_{j=1}^{na} x_i^j) + \log_2(ne) + (nm + nu) * (\log_2(ne) + \log_2(nc)) \qquad (3.3)$$

onde x representa um indivíduo; W é o fator peso a ser escolhido; nr é o número de regras no indivíduo x; na é o número de atributos em cada regra de x; x_i^j representa o tamanho do atributo i associado a cada regra j; ne é o número de exemplos; nm é o número de exemplos classificados de forma errada por x; nu é o número de exemplos não classificados por x; nc é o número de classes do problema;

É importante notar que x_i^j recebe valores diferentes para atributos discretos e contínuos. Para uma explicação detalhada sobre o cálculo do tamanho de um atributo veja (Bacardit e Garrel, 2003).

O uso do principio DMC é geralmente utilizado no cálculo da função de aptidão, contudo, caso não deseje-se usá-lo, a função de aptidão será simplesmente o quadrado da precisão obtida no conjunto de treinamento.

III. Estratégia de Classificação

A estratégia utilizada pelo GAssist é semelhante as listas de decisão (Rivest, 1987), i.e., a primeira regra que classifica a instância é de fato utilizada para classificá-la.

IV. Operadores

Operador de Recombinação: O operador de dois pontos é utilizado (veja capitulo 2, caso exista dúvida sobre este operador). A probabilidade de recombinação é relativa a cada indivíduo.

Operador de Mutação: Para a representação de atributos discretos, a mutação que inverte um bit é utilizada. Para a representação de atributos contínuos, a mutação é responsável por modificar o intervalo de discretização sobre o qual ela atua. A probabilidade de mutação é relativa a cada indivíduo.

Operador de Seleção: A seleção por torneio é utilizada. O processo de elitismo é também usado, i.e., o melhor indivíduo da geração atual é copiado para a próxima geração.

V. Política de Adição de Regras

Não existe um mecanismo explicito para a adição de regras neste sistema. Pelo contrário, o operador de recombinação de dois pontos é o responsável pelo crescimento e redução dos indivíduos. A figura 3.8 ilustra como o operador de recombinação atua de forma direta no tamanho dos indivíduos. Nesta figura, os indivíduos são apresentados de maneira a mostrar quantas regras cada um possui. Considerando a implementação do sistema, cada regra de cada indivíduo codifica todos os atributos do problema abordado, embora, como afirmado, nem todos os atributos sejam realmente

utilizados por uma regra de decisão. Perceba na figura 3.8, o crescimento de um indivíduo e a redução de outro.

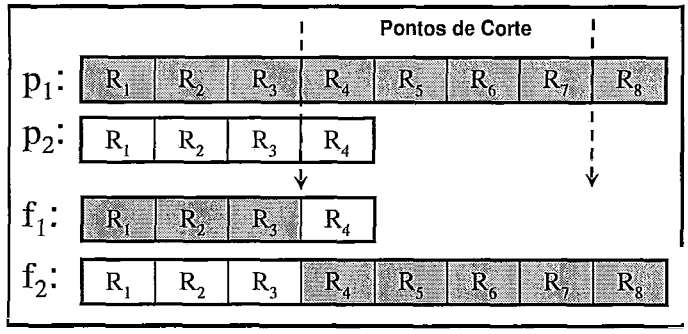

Figura 3.8: Recombinação de dois Pontos do Sistema GAssist

Embora a figura 3.8 apresente dois pontos de corte escolhidos no limite entre duas regras, os pontos de corte do operador de recombinação deste sistema podem ser escolhidos em qualquer parte dos indivíduos, i.e, podem ocorrer dentro dos próprios atributos que representam uma regra.

VI. Outras Considerações

Como o GAssist trabalha com indivíduos que não possuem tamanho fixo, faz-se necessário o controle do tamanho dos indivíduos para que estes não cresçam indefinidamente. As medidas que evitam esse fenômeno são:

Operador de Exclusão: Este operador exclui as regras de um individuo que não classificam nenhuma instância. Esse operador é executado após o cálculo da aptidão e possui duas restrições: (a) este operador é ativado após um número pré-definido de gerações; (b) este operador não é mais executado quando um indivíduo atinge um limite inferior pré-definido de número de regras.

Operador de Seleção por Torneio Hierárquico: Essa variação do operador de seleção do torneio pode ser utilizada caso se deseje. De forma geral, dado dois indivíduos a e b e um limite pré-definido ρ, o algoritmo 3.7 exibe o operador de seleção por torneio hierárquico.

Algoritmo 3.7: Seleção por Torneio Orientada por Tamanho

Entrada:
indivíduos **a, b;**
limite **ρ;**

Saída:
melhor indivíduo;

Começo:
1 – Se (|aptidão(a) – aptidão(b)| < ρ) então:
2 – Se (tamanho (a) < tamanho (b)) então retorne a;
3 – Se (tamanho (a) > tamanho (b)) então retorne b;
4 – Se (tamanho (a) = tamanho (b)) então retorne aleatório (a,b);
Senão
5 – retorne o indivíduo que possui maior aptidão;
Fim:

O algoritmo 3.7 é apresentado para o torneio de tamanho 2, entretanto, a mesma idéia é aplicada quando se utiliza o torneio de tamanho superior a 2.

VII. Descrição dos Parâmetros

popSize: Tamanho da população

numIterations: Número de iterações executadas.

initialNumberOfRules: Número inicial de regras em cada indivíduo.

probCrossover: Probabilidade de recombinação relativa a um indivíduo;

probMutationInd: Probabilidade de mutação relativa a um indivíduo.

probOne: Probabilidade de um gene que codifica o valor de um atributo discreto possuir o valor 1. (O sistema utiliza a codificação binária para representar os atributos discretos).

initMethod: modo de inicialização dos indivíduos, que pode ser aleatório ou fazer com que os indivíduos cubram algumas instâncias do conjunto de treinamento.

numStrata: Determina a quantidade subconjuntos não sobrepostos em que o conjunto de treinamento será dividido. Cada iteração do algoritmo usará um destes subconjuntos para o cálculo da aptidão dos indivíduos. Para um conjunto de treinamento com menos de 1000 instâncias, essa subdivisão não é adotada.

sizePenaltyMinRules: Mínimo de regras que um indivíduo pode possuir sem que a aptidão do mesmo seja penalizada.

tournamentSize: Tamanho do torneio.

defaultClass: Determina qual a classe será escolhida para ser a classe *default,* i.e., quando uma instância não é atribuída a nenhuma outra classe, ela será atribuída a classe *default.*

adiKR: Determina se o sistema irá usar o sistema de Intervalos Discretizados Adaptativos para a representação dos atributos contínuos.

Discretizer [1..10]: Determina quais são os métodos de discretização usados pelo sistema. No máximo dez métodos podem ser escolhidos.

maxIntervals: Máximo número de intervalos para a discretização.

probMerge: Probabilidade de dois intervalos discretizados serem reunidos.

probSplit: Probabilidade de separar um intervalo discretizado.

probReinitializeBegin: Probabilidade de reincialização dos intervalos discretizados na primeira geração.

probReinitializeEnd: Probabilidade de reincialização dos intervalos discretizados na última geração.

iterationHierarchicalSelection: Número da iteração em que a seleção por torneio hierárquico será ativada (veja tópico VI desta seção).

hierarchicalSelectionThreshold: Taxa ρ da seleção por torneio hierárquico (veja tópico VI desta seção).

useMDL: Determina se o principio da Descrição de Mínimo Comprimento deve ser usado para o cálculo da aptidão.

iterationMDL: Número da iteração para que o principio da Descrição de Mínimo Comprimento seja ativado. Enquanto o DMC não for ativado, a aptidão é calculada como sendo o quadrado da precisão obtida no conjunto de treinamento.

initialTheoryLengthRatio: Taxa que balanceia a complexidade da teoria com a sua aptidão.

weightRelaxFactor: Peso W usado na equação 3.3 que calcula a aptidão.

iterationRuleDeletion: Iteração em que o processo de deleção de regras será ativado.

ruleDeletionMinRules: Número mínimo de regras em um individuo para que o operador de deleção possa atuar.

VIII. Ciclo Básico de Execução

O GAssist segue o ciclo básico de execução de um AG; mais precisamente, este sistema segue o ciclo do algoritmo 3.4 que representa a abordagem Pittsburgh (seção 3.4.1).

3.5.2 O Sistema XCS – Abordagem Michigan

Atualmente, o sistema mais importante que representa a abordagem Michigan é conhecido como XCS (Wilson, 1995). Neste sistema, técnicas de aprendizado por reforço (Kaelbling et al, 1996) são usadas e o algoritmo genético é um simples módulo do sistema que é usado para a produção de novos indivíduos.

I. Representação do Conhecimento

As regras do XCS são representadas na Forma Normal Disjuntiva Modificada (FNDM). Tanto os atributos discretos quanto contínuos deste sistema são representados por meio de intervalos. A representação de atributos contínuos por meio de intervalos foi apresentada na seção 3.2.1 deste capítulo, e a representação de atributos discretos por meio de intervalos segue o mesmo padrão utilizado na representação dos atributos contínuos. Considere o exemplo seguinte que exibe a codificação usada no XCS.

Exemplo 3.3: Considere o problema apresentado no exemplo 3.1 em que se deseja classificar um cão em bonito ou feio. Cada cão é descrito através de três atributos: pêlo (curto, médio ou longo), tamanho (pequeno, médio ou grande) e peso, que varia entre 10 kg e 50 kg.. Como o XCS utiliza intervalos para a representação dos atributos, pêlo e tamanho serão convertidos para esta forma de representação. Assim, para o atributo pêlo, o número 1 representa pêlo curto, o número 2 representa pêlo médio e o número 3 representa pêlo longo. Essa codificação é usada para todos os atributos discretos, sendo que é atribuído um número inteiro a cada valor que um atributo pode assumir. Assim, o atributo tamanho também será descrito pelos números inteiros 1, 2, e 3. O atributo peso não sofre nenhum tipo de transformação. Como as regras do XCS utilizam intervalos para representar os atributos, cada atributo será representado por dois números, sendo que para os atributos discretos estes números são naturais e para atributos contínuos estes números são reais. A regra da figura 3.9 diz que os cães que possuam pêlo = médio ou longo, tamanho = pequeno e peso $\in$ [25, 35], são considerados bonitos.

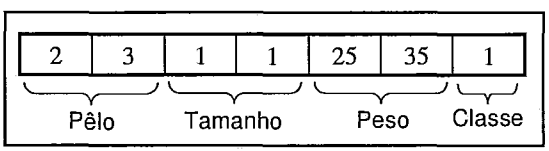

Figura 3.9: Representação de Regra no XCS

Note que a tanto os atributos discretos (pêlo e tamanho) quanto os atributos contínuos (peso) são representados por meio de intervalos. Assim o atributo pêlo representado pelo intervalo [2, 3] cobre os valores de peso = médio ou longo, o atributo tamanho representado por [1, 1] cobre o valor de tamanho = pequeno. Essa forma de representação é bastante simples, entretanto, ela não permite representar todas as combinações possíveis para atributos discretos. Por exemplo, com a codificação adotada, não é possível representar uma regra que possua pêlo = curto ou longo, ou mesmo uma regra que possua tamanho = pequeno ou grande. Isso de deve ao fato de que os atributos discretos serem também representados por meio de um intervalo.

II. Função de Aptidão

Como afirmado, o AG no XCS é apenas uma módulo do sistema, enquanto que o seu principal componente se baseia em aprendizado por reforço. Dessa forma, a função de aptidão é baseada em termos deste tipo de aprendizado. Considere, portanto, um individuo (regra). Quando a parte de condição desta regra é coberta por uma instância, a parte de ação desta regra é disparada, ou seja, a classificação desta instância é efetuada. Assim, o sistema recompensa esta ação (no caso a classificação) efetuada por essa regra. A função de aptidão é baseada numa função inversa do erro de predição de recompensa, i.e., a função de aptidão está baseada na precisão de predição de um valor de recompensa. O calculo da função de aptidão será descrita na seção VII que apresenta ciclo básico de execução deste sistema.

III. Estratégia de Classificação

O processo de votação é usado para classificar uma instância não rotulada. Dessa forma, os vários indivíduos que cobrem uma instância não rotulada votam de acordo com seu valor de aptidão, e o que possuir o maior valor classificará a instância.

IV. Operadores

Operador de Recombinação: O operador de dois pontos é utilizado. A probabilidade de recombinação é relativa a cada indivíduo.

Operador de Mutação: Este operador altera o intervalo dos atributos num indivíduo. Como a representação dos atributos contínuos e discretos é a mesma, não existe

diferença na semântica deste operador para os tipos de atributos. A probabilidade de mutação é relativa a cada atributo.

Operador de Seleção: A seleção por roleta ou por torneio é utilizada. O usuário do sistema define qual o critério de seleção será usado.

V. Política de Adição de Regras

O sistema adiciona regras em uma população por meio do operador de recombinação ou através do operador de cobertura (*covering*). Estes procedimentos serão explicados na seção VII que apresenta o ciclo de execução do sistema.

VI. Componentes do Sistema e Principais Parâmetros

O XCS trabalha com três tipos de população:

[P]: População que contém todos os indivíduos evoluídos.

[C]: Denomina-se conjunto de cobertura e representa um subconjunto de indivíduos de [P] que cobrem uma determinada instância.

[A]: Denomina-se Conjunto de Ação e representa um subconjunto de indivíduos de [C] que predizem a mesma classe para uma dada instância coberta.

Um indivíduo é representado por {rp, ep, apt, exp, temp, tca, num}, em que:

rp: representa a recompensa prevista pela execução da parte de ação da regra, i.e., a recompensa prevista pela classificação da instância.

ep: representa a estimativa do erro ocorrido na recompensa prevista.

apt: função inversa do erro de predição.

exp: representa o número de vezes que o individuo pertenceu a um conjunto de ação.

temp: usado para determinar quando o AG será executado.

tca: estimador do tamanho do conjunto de ação.

num: número de micro-classificadores[4] presentes no indivíduo.

Existem ainda dezessete parâmetros que controlam a execução do XCS. Estes parâmetros influenciam todo o comportamento do sistema, incluindo o AG, a função de

4, Um micro-classificador é um indivíduo que pode ser substituído por outro indivíduo mais geral, i.e., um indivíduo que classifique as mesmas instâncias que o micro-classificador, além de outras.

aptidão, a formação do conjunto de cobertura, a formação do conjunto de ação e assim por diante. Estes parâmetros serão descritos em seguida.

N: número máximo de indivíduos em [P].

β: taxa de aprendizado para rp, ep e apt.

α, e_0, v: parâmetros usados na função de aptidão.

γ : fator de desconto para a atualização de rp.

pmut: probabilidade de mutação para o AG.

prec: probabilidade de recombinação para o AG.

δ: fração da média da aptidão da população.

rp_I, ep_I e apt_I: valores iniciais para rp, ep e apt para uma indivíduo.

pal: probabilidade de se escolher uma classe aleatória.

θsub: Determina quando a substituição de indivíduos deve ocorrer.

θdel: Determina quando a deleção de regras deve ocorrer.

θAG: Determina quando o AG deve ser executado.

θmna: Número mínimo de classes no conjunto de cobertura.

VII. Ciclo Básico de Execução

O clico de execução do XCS é apresentado no algoritmo 3.8 e seus passos principais são discutidos em seguida.

Algoritmo 3.8 – Ciclo Básico de Execução do XCS

Entrada:
população **[P]**;
parâmetros: **N, β, α, e0, v, γ, pmut, prec, δ, rpl, ep_l, apt_l, pa_l, θsub, θdel, θGA, θmna**;

Saída:
população **[P]** modificada;

Começo:
1 – Inicialize [P];
2 – Enquanto (témino = falso) faça:
3 – Selecione uma instância I do conjunto de treinamento;
4 – Forme o conjunto de cobertura [C] a partir de [P] e I;
5 – Se (numero de classes em [C] < θmna) então
6- Chame o procedimento de cobertura;
Fim-se;
7 – Calcule o vetor de predição para [C];
8 – Se (aleatorio > pal) então
9 – Escolha uma classe aleatória em [C];
senão
10 – Escolha a melhor classe de acordo com o vetor de predição;
Fim-se;
11 – Forme o conjunto de ação [A] a partir do conjunto [C] e da classe escolhida;
12 – Classifique I com o melhor indivíduo em [A];
13 – P ← Recompensa obtida pela classificação de I;
14 – Para todas as regras em [A] faça
15 – Calcule ep usando P;
16 – Calcule rp usando P;
17 – Calcule tca;
18 – Atualize apt;
Fim-para;
19 – Se (Aplicar Operador de Substituição em [A] = verdadeiro) então
20 – Aplique operador de Substituição em [A];
Fim-se;
21 – Se (Média de tempo > θGA) então;
22 – Aplique o AG em [A];
23 – Se (Aplicar_Operador_ Substituição _AG = verdadeiro) então
24 – Aplique operador de Substituição do AG em [A];
Fim-se;
25 – Coloque as regras produzidas pelo AG em [P];
Fim-enquanto;
26 – Retorne [P];
Fim:

O passo 1 inicializa [P]. A população pode ser gerada aleatoriamente, ou ser construída a partir de um conjunto de instâncias, pode ainda ser vazia ou possuir um indivíduo que representa cada classe do problema. O passo 2 controla a execução do sistema. No passo 3, uma instância I é lida do conjunto de treinamento. O passo 4 constrói conjunto de cobertura [C] que é formado por todos os indivíduos em [P] que cobrem I. No passo 5, o algoritmo verifica se o número de classes no conjunto [C] é menor que θmna, caso verdadeiro, o procedimento de cobertura é chamado no passo 6. Este procedimento cria indivíduos que cobrem I e os valores das classes para estes indivíduos são determinados de maneira aleatória. Os valores de ep, apt, pa para estes indivíduos são respectivamente ep_I, apt_I, pa_I. Os outros parâmetros para os indivíduos possuem os seguintes valores: exp = 0, temp = 0, tca = 1 e num = 1. No passo 7, o vetor de predição é formado calculando-se dois valores para cada classe presente no conjunto de cobertura. Para cada classe, calcula-se a média ponderada das predições. Essa média é calculada dividindo-se o somatório do produto da predição pe pela aptidão ap de todos os indivíduos que classificam a instância para a mesma classe, pelo somatório das aptidões de todos estes mesmos indivíduos. O conjunto de ação [A] é formado escolhendo-se uma classe do conjunto de cobertura [C], de maneira que todos os indivíduos de [A] possuirão o valor de classe igual à classe escolhida de [C]. [A] pode ser formado de maneira aleatória ou determinística. No método aleatório, uma classe é escolhida aleatoriamente do conjunto [C] (passo 9 e depois passo 11). No método determinístico, a classe que possui a maior média ponderada das predições (que foram armazenadas no vetor de predição) é escolhida (passo 10 e depois passo 11). O passo 12 classifica I com o melhor indivíduo em [A] (o que possuir maior valor de aptidão). O passo 13 calcula um valor P através da recompensa obtida pela classificação de I. Caso I tenha sido classificado corretamente, P recebe o valor 1000, caso I seja classificado de forma errada, P recebe o valor 0. A cada vez que um indivíduo participa de um conjunto de ação [A], sua experiência (exp), seu erro de predição (ep), sua recompensa prevista (rp), o tamanho estimado de seu conjunto de ação (tca) e a sua aptidão (apt) são atualizados pelas seguintes equações: (passos 15, 16, 17 e 18):

$$\exp = \exp + 1 \tag{3.4}$$

$$ep = \begin{cases} ep + \dfrac{(|P - rp| - ep)}{\exp} & se\left(\exp < \dfrac{1}{\beta}\right) \\ ep + \beta * (|P - p| - ep) & caso\ contrário \end{cases} \quad (3.5)$$

$$rp = \begin{cases} rp + \dfrac{(P - rp)}{\exp} & se\left(\exp < \dfrac{1}{\beta}\right) \\ rp + \beta * (P - rp) & caso\ contrário \end{cases} \quad (3.6)$$

$$tca = \begin{cases} tca + \dfrac{\left(\sum_{j \in [A]} num_j - tca\right)}{\exp} & se\left(\exp < \dfrac{1}{\beta}\right) \\ tca + \beta * \left(\sum_{j \in [A]} num_j - tca\right) & caso\ contrário \end{cases} \quad (3.7)$$

A função de aptidão é um pouco mais complexa de ser calculada. Primeiro, o erro de predição (pe) do indivíduo é comparado com o ep_0 que informa que qualquer indivíduo que possuir ep menor que ep_0, possui boa precisão. Assim, o valor de k é calculado como:

$$k = \begin{cases} 1 & se(ep < ep_0) \\ \alpha * \left(\dfrac{ep}{ep_0}\right) & caso\ contrário \end{cases} \quad (3.8)$$

Agora, a precisão relativa k' de um indivíduo é calculada dividido a multiplicação de k pelo seu número de micro-classificadores (k * num), pelo somatório deste mesmo valor para todos os indivíduos pertencentes ao conjunto de ação [A]. Esse cálculo é apresentado na equação 3.9.

$$k' = \frac{k * num}{\sum_{j \in [A]} (k_j * num_j)} \quad (3.9)$$

Finalmente a equação 3.10 mostra como a aptidão é calculada (passo 18).

$$apt = apt + \beta * (k - apt) \quad (3.10)$$

Caso a aplicação do operador de substituição em [A] esteja ativo, ele será sempre aplicado ao conjunto de ação (passo 19). O operador de substituição é designado para remover os indivíduos do conjunto de ação e da população que podem ser substituídos por indivíduos mais gerais, precisos e suficientemente experientes pertencentes ao conjunto de ação. Em particular, dados dois indivíduos I_1 e I_2, I_1 pode ser substituído por I_2 se as instâncias cobertas por I_1 formam um subconjunto próprio[5] de instâncias cobertas por I_2. O individuo substituto tem a sua numerosidade (num) incrementada pela numerosidade (num) de cada indivíduo removido do conjunto de ação (passo 20).

O único componente deste sistema para a geração de novas regras é o AG nele embutido. O AG atua nos indivíduos presentes no conjunto de ação dependendo da equação 3.11 (veja o passo 21).

$$temp_{atual} - \frac{\sum_{j \in [A]} temp_j * num_j}{\sum_{j \in [A]} temp_j} > \theta_{AG} \qquad (3.11)$$

Quando o AG é chamado (passo 22), ele seleciona dois pais do conjunto de ação usando o torneio ou a roleta. Após a seleção, o AG aplica a recombinação de dois pontos nos pais para a produção de dois filhos. Um destes filhos é submetido à mutação. Os dois novos indivíduos produzidos têm seus valores de num e exp atribuídos respectivamente a 1 e 0, e os parâmetros rp, ep e apt atribuídos à média destes parâmetros calculados para os dois pais. Caso o operador de substituição do AG seja usado (passo 24), o utiliza-se o procedimento de substituição de indivíduos do conjunto [A]. Este procedimento é o mesmo descrito no passo 20 e é importante, pois os novos indivíduos produzidos pelo AG podem talvez substituir alguns indivíduos de [A]. Os novos indivíduos produzidos pelo AG são inseridos na população [P], se esta ainda não atingiu o limite N de números de indivíduos. Caso o limite N tenha sido alcançado, os novos indivíduos substituem os indivíduos que possuem a menor aptidão na população, i.e., os indivíduos com menor aptidão na população são deletados e os dois novos produzidos são colocados em seus lugares (passo 25). Finalmente, o passo 26 retorna a população [P] modificada quando o critério de parada for satisfeito. Essa população representa a teoria construída.

5. Dados dois conjunto A e B, A é um conjunto próprio de B se todos os elementos de A pertencem a B e existe ao menos um elemento em B que não pertence a A.

3.5.3 O Sistema HIDER – Abordagem Iterativa

O sistema mais importante que representa a abordagem Iterativa é conhecido como HIDER (Aguilar-Ruiz et al, 2003). Este sistema será apresentado em seguida.

I. Representação do Conhecimento

As regras do HIDER são representadas na Forma Normal Disjuntiva Modificada (FNDM). Este sistema usa a Codificação Natural (Aguilar-Ruiz et al, 2002) que usa números naturais para representar atributos discretos e contínuos. A Codificação Natural foi explicada na seção 3.2.1 deste capítulo. A seguir é apresentado um exemplo que ilustra o tipo de regra utilizado pelo HIDER.

Exemplo 3.4: Considere ainda, o problema apresentado no exemplo 3.1 em que se deseja classificar um cão em bonito ou feio. Cada cão é descrito através de três atributos: pêlo (curto, médio ou longo), tamanho (pequeno, médio ou grande) e peso, que varia entre 10 kg e 50 kg. Considerando que o atributo peso foi discretizado e que o intervalo [5, 35] foi atribuído ao número natural 3 (tabela 3.1 da seção 3.2.1), a regra representada na figura 3.10 diz que os cães que possuam pêlo = médio ou longo, tamanho = pequeno e peso $\in$ [5, 35], são considerados bonitos.

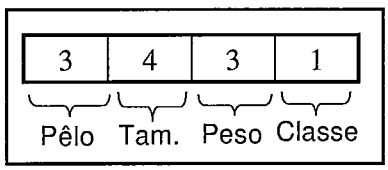

Figura 3.10: Representação de Regra no HIDER

Note que caso a codificação binária fosse adotada, a parte da regra que representa pêlo = médio ou longo, seria dada pela cadeia (*string*) 011, portanto o número 3 é usado. O mesmo raciocínio deve ser empregado para a representação de tamanho = pequeno, que é representado na codificação binária por 100, e, portanto pelo número natural 4. Quanto à representação de atributos contínuos, os mesmos devem ser discretizados e cada intervalo discretizado é representado por um número natural.

II. Função de Aptidão

A função de aptidão deste sistema é apresentada na equação 3.12. O sistema tem o objetivo de maximizar o valor desta função.

$$f(x) = 2*(N - E(x)) + C(x) + cobertura(x) \tag{3.12}$$

onde:

$$cobertura(x) = \prod_{i=1}^{m} \frac{cobertura(x,i)}{ordem(x,i)}$$

em que m representa a quantidade de atributos de um indivíduo x; N representa a quantidade de exemplos no conjunto de treinamento; E(x) representa a quantidade de erros de classificação cometidos por x; C(x) representa a quantidade de acertos efetuados por x; cobertura(x,i) no caso de atributos discretos, representa a quantidade de valores ativos representados no atributo i, e para atributos discretizados, cobertura(x,i) representa a quantidade de intervalos de discretização que o atributo i cobre; ordem(x,i), para um atributo discreto representa a quantidade de valores que i pode assumir, e para atributos discretizados, ordem(x,i) representa a quantidade de intervalos de discretização obtidos para este atributo.

III. Estratégia de Classificação

Um indivíduo é representado por uma única regra, entretanto, ao final do processo de aprendizado, a teoria será representada por uma concatenação de vários indivíduos evoluídos ao longo de sucessivas chamadas do AG. Assim, a estratégia de classificação utilizada pelo HIDER é semelhante às listas de decisão (Rivest, 1987), i.e., a primeira regra que classifica a instância é de fato utilizada para classificá-la.

IV. Operadores

Embora os dados discretos e contínuos sejam representados por números naturais, os operadores de mutação e de recombinação são totalmente diferentes para estes tipos de dados. Essa seção apresenta os operadores para cada tipo de dado. Todas as definições desta seção foram apresentadas em (Aguilar-Ruiz et al, 2002).

Operador de Mutação para Dados Contínuos: Como explicado na seção 3.2.1, os atributos contínuos devem ser discretizados por qualquer algoritmo de discretização. Uma vez discretizados, o algoritmo retorna um vetor de cortes que fornece os intervalos de discretização obtidos. Estes intervalos são então numerados, e estes números, agora representado intervalos, são utilizados pela a mutação. Uma tabela é usada para auxiliar na organização destes intervalos, embora não seja necessário o seu armazenamento para a utilização do operador de mutação. O exemplo a seguir ilustra como essa tabela é construída e numerada.

Exemplo 3.5: Considere o atributo peso apresentado no exemplo 3.1. Assuma que algum algoritmo de discretização tenha retornado o vetor de cortes de tamanho k = 5 dado por: {5, 10, 20, 35, 40}. Assim, exibe-se novamente, com objetivo de facilitar a consulta, a tabela de discretização já apresentada na seção 3.2.1.

Tabela 3.2: Intervalos de Discretização

Pontos de Corte	10	20	35	40
5	**1**	**2**	**3**	**4**
10	-	**6**	**7**	**8**
20	-	-	**11**	**12**
35	-	-	-	**16**

Algumas definições são necessárias para o entendimento do operador de mutação. Todas elas são construídas sobre a tabela que organiza os intervalos discretizados. Assim, considere as seguintes definições:

Definição 3.4 (Linha e Coluna): Considere n um número natural em uma tabela de discretização e k o número dos cortes, i.e., a quantidade de valores que o vetor de discretização possui. A linha (l) e a coluna (c) de n são dadas por:

$$l = (n-1)/(k-1)+1 \tag{3.13}$$

$$c = (n-1)\%(k-1)+1 \tag{3.14}$$

Exemplo 3.6: Para tabela 3.2, e $n_1 = 2$ e $n_2 = 8$. Tem-se: l(2) = 1, c(2) = 2, l(8) = 2, c(8) = 4.

Defimição 3.5 (Limites): Os limites de um número natural n são aqueles valores que limitam n nas quatro direções possíveis: cima, baixo, esquerda e direita. Assim, estes valores representados respectivamente por lci(n), lba(n), les(n), ldir(n) são dados por:

$$lci(n) = c \tag{3.15}$$

$$lba(n) = (k-1)*(c-1)+c \tag{3.16}$$

$$les(n) = (k-1)*(l-1)+l \tag{3.17}$$

$$ldir(n) = (k-1)*l \tag{3.18}$$

Exemplo 3.7: Do exemplo 3.6, tem-se: lci(2) = 2, lba(2) = 6, les(2) = 1, ldir(2) = 4, lci(8) = 4, lba(8) = 16, les(8) = 6, ldi(8) = 8.

Definição 3.6 (Deslocamentos): Os deslocamentos adjacentes para cima, baixo, esquerda e direita para um número natural n, respectivamente representados por cim(n), bai(n), esq(n) e dir(n), são dados por:

$$cim(n) = \max(lci(n), n-k+1) \tag{3.19}$$

$$bai(n) = \min(lba(n), n+k-1) \tag{3.20}$$

$$esq(n) = \max(les(n), n-1) \tag{3.21}$$

$$dir(n) = \min(ldi(n), n+1) \tag{3.22}$$

Os deslocamentos horizontais e verticais para um dado n, representados respectivamente por hor(n) e ver(n), são definidos como todos os deslocamentos possíveis em uma linha e em uma coluna. Assim:

$$hor(n) = \bigcup_{i=1}^{k-1} \max(les(n), (k-1)(l-1)+i) \tag{3.23}$$

$$ver(n) = \bigcup_{i=1}^{k-1} \max(lba(n), (k-1)(i-1)+c) \tag{3.24}$$

Exemplo 3.8: Do exemplo 3.7, tem-se: cim (2) = 2, bai(2) = 6, esq(2) = 1, dir(2) = 3, cim(8) = 4, bai(8) = 12, esq(8) = 7, dir(8) = 8, hor(2) = {1, 2, 3, 4}, ver(2) = {2, 6}, hor(8) = {6, 7, 8}, ver(8) = {4, 8, 12, 16}.

Dadas às definições 3.4, 3.5 e 3.6, defini-se o operador de mutação para dados contínuos do sistema HIDER como:

Definição 3.7 (Mutação Natural para Dados Contínuos): Dado um número natural n, a Mutação Natural de n, denotada por Mut (n), é qualquer valor próximo a n usando todos os deslocamentos. Obviamente, a mutação de n deve retornar um número diferente de n, assim:

$$Mut(n) \in \left\{ n' \mid n' \in \left\{ mov(n) - n \right\} \right\} \tag{3.25}$$

onde mov(n) = cim(n) ∪ bai(n) ∪ esq(n) ∪ dir(n).

Exemplo 3.9: Considerando a tabela 3.2, Mut(2) ∈ { {1, 2 ,3 ,6} – {2} }, i.e., Mut(2) ∈ {1, 3, 6}. O valor é escolhido de forma aleatória.

Operador de Recombinação para Dados Contínuos: Dadas às definições 3.4, 3.5 e 3.6, defini-se o operador de recombinação para dados contínuos do sistema HIDER como:

Definição 3.8 (Recombinação Natural para dados Contínuos): Dados dois números naturais n_i e n_j, a recombinação entre n_i e n_j, denotada por Recom(n_i, n_j), é obtido como:

$$\text{Recom}(n_i, n_j) \in ((hor(n_i) \cap ver(n_j)) \cup (hor(n_j) \cap ver(n_i))) \tag{3.26}$$

Exemplo 3.10: Considerando a tabela 3.2, a recombinação entre os números naturais 2 e 8, é dada por: Recom(2, 8) ∈ { { {1, 2, 3, 4}∩{4, 8, 12, 16} }∪{ {6, 7, 8}∩{2, 6} } }, i.e. Recom(2, 8) ∈ {4, 6}.

A recombinação natural pode ser entendida como um operador que troca o limite superior dos intervalos que são cruzados. De fato, para o exemplo 3.10, os pais são 2 = [5, 20] e 8 = [10, 40] e os filhos são 4 = [5, 40] e 6 = [10, 20].

É importante notar que o procedimento acima define a recombinação para um atributo contínuo em um indivíduo. No HIDER, o operador de recombinação de dois pontos é utilizado, e todos os atributos contínuos localizados entre os pontos de corte são submetidos a esse mesmo procedimento.

Operador de Mutação para Dados Discretos: A mutação para atributos discretos trabalha de maneira idêntica à mutação binária. Assim, considere a seguinte definição:

Definição 3.9 (Mutação Natural): Considere n o valor de um gene de um indivíduo representado por um número natural, assim, a mutação natural do k-ésimo bit de n, denotado por $mut_k(n)$, é o número natural produzido pela troca do bit de posição k em n. A equação 3.27 ilustra como a mutação natural é realizada.

$$mut_k(n) = (n + 2^{k-1}) \,\%\, 2^k + 2^k * chão\left(\frac{n}{2^k}\right) \tag{3.27}$$

onde: $k \in \{1, 2, ..., |A|\}$; |A| representa o número de valores que o atributo n pode assumir; % retorna o resto da divisão inteira; e a função chão(x) retorna o maior inteiro menor ou igual a x.

Exemplo 3.11: Uso da equação 3.27 para alterar o primeiro bit do número natural 1 representado por três bits, i.e, 1 (001).

$$mut_1(1) = (1 + 2^0) \,\%\, 2^1 + 2^1 * chão\left(\frac{1}{2^1}\right) = 0\,(000)$$

Operador de Recombinação para Dados Discretos: Este operador é está diretamente relacionado com no operador de mutação explicado acima. De forma geral, o operador de recombinação é baseado em duas definições, a saber: Conjunto de Mutação e Classe de Mutação de Ordem j.

Definição 3.10 (Conjunto de Mutação): O conjunto de mutação para um gene n, denotado por Mut(n), é o conjunto de todas as mutações válidas para n. Assim:

$$Mut(n) = \bigcup_{k=1}^{|A|} mut_k(n) \tag{3.28}$$

onde $mut_k(n)$ é o número natural retornado pela mutação do k-ésimo bit de n; |A| é a quantidade de valores que o atributo n pode assumir.

Exemplo 3.12: O conjunto de mutação para n = 1(001), é dado por Mut(1) = {0, 3, 5}

Definição 3.11 (Classe de Mutação de Ordem j): Considere Z um conjunto de números naturais. A definição de Classe de Mutação de Ordem j para Z, denotada por [Cl-Mut(Z)]^j é dada por:

$$\begin{aligned} [Cl-Mut(Z)]^0 &= Z \\ [Cl-Mut(Z)]^1 &= \bigcup_{z\in Z} Mut(n) \\ &\vdots \\ [Cl-Mut(Z)]^j &= Mut([Cl-Mut(Z)]^{j-1}) \end{aligned} \tag{3.29}$$

A classe de mutação de ordem j para um número n, denotada por, [Cl-Mut(n)^j], é definida sobre o conjunto de mutação. Por definição, [Cl-Mut(n)^0] é o próprio n. A classe de mutação de ordem j, para j > 0, é a união dos conjuntos de mutação de cada número presente na classe de mutação j-1, acrescido do próprio número n.

Exemplo 3.13: Considere Z = {1}, e que o número natural 1 é codificado com 3 bits, portanto 001. Para calcular [Cl-Mut(1)]^2, tem-se:
Mut(1) = {0, 3, 5}, (veja exemplo 3.7);
[Cl-Mut(1)]^0 = {1} (Por definição);
[Cl-Mut(1)]^1 = Mut(1) ∪ {1} = {1, 0, 3, 5};
[Cl-Mut(1)]^2 = Mut(1) ∪ Mut(0) ∪ Mut(3) ∪ Mut(5) ∪ {1}, assim: Cl-Mut(1)^2 = {1, 0, 3, 5, 2, 4, 7};

O operador de recombinação para dados discretos está diretamente relacionado à classe de mutação. Resumidamente, dados dois números naturais n_1 e n_2, a prole produzida por estes números será escolhida no conjunto formado pela primeira interseção não vazia entre as classes de mutação destes números. Assim, considere a seguinte definição:

Definição 3.12 (Recombinação Natural para Dados Discretos): Considere n_i e n_j dois genes de dois indivíduos x_i e x_j. A prole produzida pela Recombinação de n_i e n_j, Recom(n_i, n_j), será escolhida no conjunto formado pela primeira interseção não vazia entre as classes de mutação de n_i e n_j. Considere $t \geq 0$ e Q^t = [Mut(mut(n_i))]^t ∩ [Mut(mut(n_j))]^t, assim:

$$\mathrm{Re}com(n_i, n_j) = \{\, z \in Q^t \mid Q^t \neq \varnothing, \forall s \geq 0, s < t, Q^s = 0\} \quad (3.30)$$

Exemplo 3.14: Deseja-se realizar a recombinação nos números $n_1 = 29$ (11101) e $n_2 = 6$ (00110). Assim, deve-se encontrar a primeira classe de mutação entre n_1 e n_2 cuja interseção entre ambas não seja vazia, e dessa forma tem-se:

Calculo da Classe de Mutação de Ordem 0:

$[\text{Cl-Mut}(29)]^0 = \{29\}$

$[\text{Cl-Mut}(6)]^0 = \{6\}$

$[\text{Cl-Mut}(29)]^0 \cap [\text{Cl-Mut}(6)]^0 = \{\varnothing\}$, assim deve-se calcular a Cl-Mut de Ordem 1.

Calculo da Classe de Mutação de Ordem 1:

$[\text{Cl-Mut}(29)]^1 = \{29, 28, 31, 25, 21, 13\}$

$[\text{Cl-Mut}(6)]^1 = \{6, 7, 4, 2, 14, 22\}$

$[\text{Cl-Mut}(29)]^1 \cap [\text{Cl-Mut}(6)]^1 = \{\varnothing\}$, assim deve-se calcular a Cl-Mut de Ordem 2.

Calculo da Classe de Mutação de Ordem 2:

$[\text{Cl-Mut}(29)]^2 = \{ 29, 28, 31, 25, 21, 13, 30, 24, 20, 12, 27, 23, 15, 17, 9, 5, 9, 5\}$;

$[\text{Cl-Mut}(6)]^2 = \{6, 7, 4, 2, 14, 22, 5, 3, 15, 23, 0, 12, 20, 10, 18, 30\}$;

$[\text{Cl-Mut}(29)]^2 \cap [\text{Cl-Mut}(6)]^2 = \{30, 20, 12, 15, 5, 23\}$

Assim, os filhos produzidos por 29 e 6 são escolhidos aleatoriamente no conjunto {30, 20, 12, 15, 5, 23}.

É importante notar que o procedimento acima define a recombinação para um atributo discreto em um indivíduo. No HIDER, o operador de recombinação de dois pontos é utilizado, e todos os atributos discretos localizados entre os pontos de corte são submetidos a esse mesmo procedimento.

Operador de Seleção: A seleção por roleta é utilizada. O processo de elitismo é também usado, i.e., o melhor indivíduo da geração atual é copiado para a próxima geração.

V. Política de Adição de Regras

O sistema evolui um conjunto de regras a cada chamada do AG. A melhor regra de cada conjunto é usada para construir a teoria. Esse processo será explicado na seção VII que apresenta o clico básico de execução do sistema.

VI. Parâmetros do Sistema

Em relação aos sistemas anteriores, este sistema apresenta uma quantidade bem reduzida de parâmetros, são eles:

tam_pop: tamanho da população.
num_ger: número de gerações;
rep: porcentagem de indivíduos da geração atual que serão copiados para a próxima geração.
prec: probabilidade de recombinação relativa a cada indivíduo.
pmut: probabilidade de mutação relativa a cada indivíduo.

VII. Ciclo Básico de Execução

O clico de execução do HIDER é apresentado no algoritmo 3.9 e seus principais passos são discutidos em seguida.

Algoritmo 3.9 – Ciclo Básico de Execução do HIDER

Entrada:
Arquivo de Treinamento **Tr**;
parâmetros: **tam_pop, num_ger, rep, prec, pmut**

Saída:
Teoria **T**;

```
Começo:
 1 – T ← ∅;
 2 – Enquanto (número de instâncias positivas em Tr > 0) faça
      3 – r ← AG(Tr);
      4 – T ← T ∪ r;
      5 – Remova_Exemplos_Cobertos(Tr, r);
      Fim-enquato;
6 – Retorne T;
Fim;

Procedimento AG (Arquivo_Exemplos Tr)
Começo-AG:
 1 – Inicialize P;
 2 – Para (i = 1 até num_ger) faça
      3 – Avalie P;
      4 – P' ← Seleção P;
      5 – P' ← P' + Replicação P;
      6 – P' ← P' + Recombinação P;
      7 – P' ← P' + Mutação P;
      8 – P ← P';
      Fim-para;
 9 – Avalie P;
 10 – retorne Melhor_Indivíduo(P)
Fim-AG;
```

O ciclo de execução do HIDER está divido em dois passos básicos: o primeiro, chamado de ciclo principal, que controla a execução geral do sistema, e o segundo, chamado de Procedimento AG, é o procedimento utilizado para a adição de regras na teoria.

O ciclo principal é tipicamente um algoritmo de cobertura, sendo que o mesmo é executado até que o conjunto de exemplos de treinamento seja totalmente coberto (passo 2). O passo 3 chama o Procedimento AG que é o responsável por gerar regras

que serão adicionadas à teoria T. O passo 4 é responsável pela construção da teoria, e, dessa forma, a nova regra r gerada pelo AG é simplesmente concatenada com o restante de regras já presentes em T. O passo 5 remove as instâncias do conjunto de treinamento que foram cobertas pela regra r retornada pelo AG. Quando todo o conjunto de treinamento estiver coberto, o ciclo principal retorna a teoria T produzida (passo 6).

O passo 1 do procedimento AG inicializa a população de maneira aleatória e o passo 2 é o responsável pelo controle do fim da execução do procedimento. O passo 3 avalia toda a população com a função de aptidão e o passo 6 seleciona os indivíduos utilizando a roleta. Os indivíduos selecionados são colocados em uma população temporária chamada de P' (passo 4). O passo 5 adiciona a P' alguns indivíduos de P (selecionados também por roleta). Os passos 6 e 7 realizam respectivamente, a recombinação e a mutação em indivíduos de P e adicionam os novos indivíduos gerados a P'. O passo 8 faz com que P se torne P', dando assim continuidade ao processo de evolução. Assim que o critério de parada for atingido, o melhor indivíduo é retornado (passo 10). É importante notar, mais uma vez, que embora o procedimento AG manipule indivíduos com apenas uma regra, a teoria T é construída através da concatenação de várias regras retornadas por várias chamadas deste procedimento.

4. Novo Sistema para Aprendizado Genético de Regras

Este capítulo propõe um sistema genético para aprendizado de regras de decisão. As principiais contribuições do trabalho são apresentadas e o ciclo de execução do sistema é discutido. Posteriormente, apresentam-se os principais parâmetros utilizados pelo sistema, juntamente com algumas últimas considerações para que a compreensão do sistema possa ser de fato realizada.

4.1 Introdução

O objetivo do sistema proposto é alcançar alta precisão (*accuracy*) de classificação utilizando teorias simples (compostas de poucas regras e poucos atributos). Com este objetivo em mente, são apresentadas em seguida algumas diretrizes assumidas para o desenvolvimento do sistema, bem como uma explicação que justifica a razão destas serem adotadas:

Utilização da Forma Normal Disjuntiva Modificada (FNDM): A FNDM (Michalski, 1983) permite uma maior compactação para a representação de regras e por isso é utilizada na maioria dos sistemas que usam um AG para aprendizado de regras.

Utilização da Codificação Natural: O uso deste tipo de codificação (Aguilar-Ruiz et al, 2002) garante grande redução na quantidade de memória gasta para representar um indivíduo. A mesma forma de representação utilizada pelo HIDER (veja capítulo 3), é usada em nosso sistema.

Seleção Genética de Atributos: Diversos trabalhos, (Kohavi e John, 1996) (Guyon e Elisseeff, 2003) demonstram a eficácia da aplicação de métodos de seleção de atributos a problemas de classificação. Mais precisamente, (Yang, 1998) e (Punch et al, 1998) exibem a eficácia da aplicação de AGs ao problema de Seleção de Atributos. Além de um aumento visível da precisão de classificação, chama atenção o baixo número de atributos selecionados para a classificação dos dados. Assim, pretende-se obter teorias com alta precisão (*accuracy*) de classificação que utilizem um número reduzido de atributos.

Utilização da Abordagem Híbrida para Evolução da Teoria: O sistema codifica um indivíduo como uma teoria (Pittsburgh) enquanto tenta explorar ao máximo o potencial de classificação de cada regra (Michigan) presente no indivíduo. O modelo híbrido proposto neste trabalho, adiciona novas regras em uma teoria somente após regras

previamente adicionadas terem sido amplamente evoluídas. Dessa forma, espera-se obter teorias compostas por um número reduzido de regras.

Novos Operadores de Recombinação: O trabalho (Aguilar-Ruiz et al, 2002) apresentou juntamente com a Codificação Natural, dois novos operadores de recombinação (relativos a dados discretos e contínuos). Entretanto, como será demonstrado, os operadores de Aguilar-Ruiz et al apresentam algumas desvantagens e por essa razão dois novos operadores de recombinação são propostos e utilizados no sistema.

Resumidamente, este trabalho é apresenta uma proposta de construção de uma máquina de aprendizado genético de regras em que uma nova abordagem híbrida para a evolução dos conceitos é utilizada. As regras são representadas na Forma Normal Disjuntiva Modificada e codificadas com a Codificação Natural, porém o sistema possui as seguintes e, atualmente, exclusivas propriedades:

(1) – Mecanismo Implícito de Seleção de Atributos (MISA).
(2) – Nova abordagem Híbrida para a evolução de teorias.
(3) – Novo operador de recombinação para dados discretos.
(4) – Novo operador de recombinação para dados contínuos (discretizados).

Nas próximas seções, cada um destes novos componentes será discutido em detalhes, juntamente com as razões que levaram a proposição e conseqüente utilização de cada um deles.

4.2 Utilização da Codificação Natural

Como apresentado no capitulo anterior, a Codificação Natural utiliza um número natural para representar atributos discretos e contínuos, sendo que estes últimos devem ser discretizados. Para dados discretos, o número natural representa diretamente o número binário, ou seja, o número natural pertencerá ao intervalo $[0, 2^{|N_i|} - 1]$, em que $|N_i|$ representa o número de valores que o atributo i pode assumir. O sistema proposto utiliza como linguagem da regra a FNDM e a codifica utilizando a Codificação Natural Considere o exemplo seguinte que ilustra a economia de memória adquirida pela utilização da Codificação Natural.

Exemplo 4.1: Considere o problema apresentado no capítulo 3 em que se deseja classificar um cão em bonito ou feio. Cada cão é descrito através de três atributos: pêlo

(curto, médio ou longo), tamanho (pequeno, médio ou grande) e peso, que varia entre 10 kg e 50 kg. Considere abaixo duas representações distintas da mesma regra, ou seja, considerando que o atributo peso tenha sido discretizado e que o intervalo [20, 35] foi atribuído ao número natural 11, as regras representadas na figura 4.1 descrevem que os cães que possuam pêlo = médio ou longo, tamanho = pequeno e peso ∈ [20, 35], são considerados bonitos.

A primeira regra utiliza a codificação híbrida (sistema GAssist), em que atributos discretos são representados por valores binários e atributos contínuos são representados por meio de intervalos discretizados adaptativos. A segunda regra utiliza a Codificação Natural (sistema HIDER).

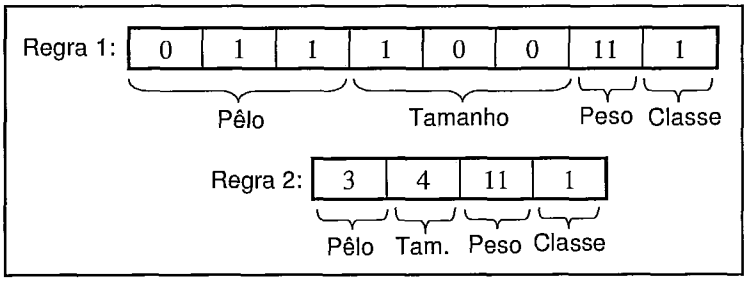

Figura 4.1: Comparação de Regras

Para o entendimento da economia de memória alcançada, é necessário compreender como o espaço de memória é usado por diferentes tipos de dados. De maneira geral, os tipos de dados básicos de uma linguagem de programação se organizam como apresentados na tabela 4.1:

Tabela 4.1: Tipo de Dados

Tipo de Dado	Memória Utilizada (em bytes)
booleano	1
_int8	1
_int16	2
_int32	4
_int64	8
flutuante	4
duplo	8

As variáveis do tipo booleano podem assumir os valores 'verdadeiro' ou 'falso'. As variáveis int8, int16, int32 e int64 podem assumir valores inteiros, sendo que a diferença está na quantidade de valores que determinada variável pode assumir. Assim, a variável _int8 pode assumir os valores inteiros no intervalo [-128, 127], a variável _int16, pode assumir os valores inteiros no intervalo [-32768, 32767] e assim por

diante. As variáveis do tipo flutuante e duplo podem assumir valores reais, sendo que a diferença entre ambas está na quantidade de valores que cada uma pode assumir e também na precisão de cada uma, i.e, na quantidade de dígitos depois do separador decimal.

É importante perceber que embora existam diferenças no armazenamento das variáveis para cada linguagem de programação, a menor unidade de alocação para todas as linguagens é de 1 byte (8 bits). Assim, mesmo quando se utiliza a variável do tipo booleano que pode assumir apenas os valores 'verdadeiro' ou 'falso', a quantidade de memória de 1 byte é utilizada. Dessa forma, quando se usa a codificação binária, para cada valor (0 ou 1) usado, a quantidade de memória de 1 byte é utilizada. Assim, para a regra 1, a quantidade de 8 bytes de memória é usada. O calculo é fornecido pela seguinte equação:

$$\sum_{i=1}^{n} numValAtrDis_i + numAtrCon + 1 \qquad (4.1)$$

em que $NumValAtrDis_i$ representa o número de valores que um atributo discreto i pode assumir (sendo n a quantidade de atributos discretos presentes na regra) e numAtrCon representa a quantidade de atributos discretizados presentes na regra.

Os atributos que foram discretizados serão representados por variáveis que utilizam 1 byte de memória, e a adição de um byte ao fim da equação 4.1, representa a variável que codifica o valor da classe. Assim, para a regra 1, tem-se a quantidade de memória gasta expressa por: (3 + 3) + 1 + 1 = 8 bytes.

Para o cálculo da quantidade de memória gasta na representação quando a Codificação Natural é usada, é necessário analisar os dois tipos de atributos.

Para atributos discretos, verificam-se quantos valores diferentes cada atributo pode assumir, já que o número natural pertencerá ao intervalo $[0, 2^{|Ni|} - 1]$, em que $|N_i|$ representa o número de valores que o atributo i pode assumir. O número de valores que um atributo pode assumir determina o tipo de variável inteira que será usada para a sua representação. No caso deste exemplo, como ambos os atributos discretos (pêlo e tamanho) podem assumir três valores diferentes, a variável inteira _int8 que usa 1 byte de memória é usada. Note que este tipo de variável pode ser usado para representar atributos que assumam no máximo 8 valores distintos. Para atributos que assumam mais valores, as outras variáveis inteiras devem ser usadas. Assim, caso um atributo possa assumir 9 valores distintos, o tipo de variável _int16 que usa 2 bytes deverá ser usado.

Contudo, na maioria das vezes, o tipo _int8 servirá para a representação dos atributos discretos, já que é incomum um atributo discreto que possa assumir mais que 8 valores.

Para a representação do atributo contínuo que foi discretizado, a variável inteira _int8 será também utilizada, já que é raro que um atributo seja discretizado em mais que 255 intervalos[1]. Dessa forma, o calculo da quantidade de memória usada pela regra 2 é dado pela equação 4.2:

$$numAtrDis * tamDado + numAtrCon + 1 \qquad (4.2)$$

em que numAtrDis representa a quantidade de atributos discretos presentes na regra; tamDado representa o tamanho em bytes do tipo de dado necessário para a representação dos valores que os atributos discretos podem assumir e numAtrCon representa a quantidade de atributos discretizados presentes na regra. Assim, a regra 2 utiliza (2 * 1) + 1 + 1 = 4 bytes de memória.

Em relação à regra 1, a regra 2 economizou 50% de memória. Essa economia torna-se ainda mais evidente caso cada indivíduo da população represente uma teoria que pode ser composta de várias regras (Pittsburgh). Deve-se reforçar uma vez mais, que o sistema proposto utiliza indivíduos que representam uma teoria e, dessa forma, a economia de memória se faz mais presente.

4.3 O Mecanismo Implícito de Seleção de Atributos (MISA)

A abordagem clássica para a construção de um método genético para Seleção de Atributos (Yang, 1998), consiste em usar uma variável do tipo booleano para cada atributo do problema. Assim, para um atributo, caso esta variável possua o valor 'verdadeiro', significa que o atributo é usado, enquanto que o valor 'falso' significa que o mesmo não é utilizado. Embora o tipo de variável booleano seja utilizado nesta representação, esse tipo de dado apresenta um grande desperdício de memória, uma vez que para alocá-lo é necessária a quantidade de memória de 1 byte (8 bits) - menor unidade de alocação das linguagens de programação. Assim, embora seja necessário apenas um bit (1 ou 0) para representar a utilização ou não utilização do atributo, o uso do tipo de dados booleano implicada na perda de 7 bits de memória.. Esse problema se agrava se o problema é composto de muitos atributos e quando cada indivíduo representa uma teoria que pode ser composta de várias regras. Dessa forma, a alocação de um vetor do tipo booleano para cada regra presente num indivíduo, sendo que cada

1. A variável inteira _int8 pode assumir valores inteiros no intervalo [0, 255] caso o bit de sinal seja utilizado para armazenamento e não para definir se a variável é positiva ou negativa.

um destes vetores possui tamanho igual ao número de atributos do problema, é algo inviável, um desperdício excessivo de memória.

O mecanismo de seleção de atributos proposto não utiliza nenhuma variável externa e por esse motivo ele foi denominado de Mecanismo Implícito de Seleção de Atributos (MISA). Embora a Codificação Natural utilize números naturais para a representação de atributos contínuos e discretos, o MISA é distinto para cada tipo de atributo.

4.3.1 MISA para Atributos Discretos

Considere um atributo discreto N_i em que $|N_i|$ representa a quantidade de valores que este atributo pode assumir. Para atributos discretos, um bit, o de número $(|N_i| + 1)$, será utilizado como mecanismo de seleção de atributos. Caso o bit seja 1, o atributo será usado, caso seja 0, o mesmo não será utilizado. Assim, existe uma grande economia de memória, já que será utilizado apenas um bit da representação natural como o mecanismo da seleção. Considere o exemplo 4.2 que ilustra a utlização do MISA para atributos discretos.

Exemplo 4.2: Considere a regra 2 do exemplo 4.1. Os atributos pêlo e tamanho podem assumir três valores, assim |pêlo| = |tamanho| = 3. Caso estes atributos sejam utilizados, o bit de posição 4 (3 + 1 = 4) destas variáveis deverá possuir o valor 1. Assim, a figura 4.2 apresenta a regra exposta no exemplo 4.1 em que os atributos peso e tamanho são utilizados com o uso do MISA para dados discretos.

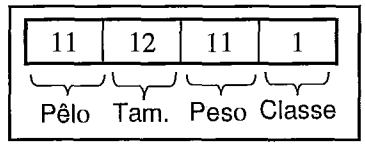

Figura 4.2: MISA para Atributos Discretos

Note que foi somado o valor 8 aos valores dos atributos pêlo e tamanho. Este valor corresponde ao bit da quarta posição destas variáveis assumindo o valor 1.

O mecanismo além de muito econômico é bastante simples de ser testado. Caso o número codificado no indivíduo para algum atributo i seja maior que $2^{|N_i|}$, sabe-se que o atributo é utilizado. Caso o número seja menor, o atributo não é usado. Este teste possui exatamente a mesma complexidade de ser efetuado se a seleção de atributos fosse efetuada por uma variável externa a codificação. Note, entretanto, a grande

economia de memória que se faz com este tipo de representação, já que não é necessária nenhuma variável externa a própria variável usada para a codificação do atributo.

A única ressalva que deve ser feita em relação ao MISA para dados discretos, se relaciona à quantidade de valores que cada variável suporta com a utilização deste tipo de mecanismo. Deve-se lembrar que agora, um bit da variável inteira é usado para a seleção de atributos e, portanto, esse bit não pode ser mais usado para representar valores que o atributo pode assumir. Assim, a variável inteira de 1 byte (_int8) suporta não mais 8 valores de atributos, mas sim 7, já que um bit da mesma é usado para determinar se o atributo é ou não utilizado.

4.3.2 MISA para Atributos Contínuos

Para dados contínuos, o mecanismo é baseado no número de intervalos retornados por algum algoritmo de discretização. Para θ representando a quantidade de intervalos retornados, existe um conjunto $\mu = \{n_1, n_2.., n_\theta\}$ de números naturais que representam estes intervalos. Considere n_θ o numero natural atribuído ao último intervalo. Assim, o número representado por $\alpha = n_\theta + 1$, constitui o número que verifica se um atributo será ou não utilizado. Portanto, para um dado n_i ($1 \leq i \leq \theta$), a sua utilização é dada pelo número $n_i + \alpha$. Ou seja, caso n_i seja maior que α, o atributo por ele representado é utilizado. Caso contrário, se o n_i for menor que α, o atributo não será usado. Considere o exemplo 4.3 que ilustra a utlização do MISA para atributos contínuos que foram discretizados. .

Exemplo 4.3: Assuma que para o atributo peso do exemplo 4.1, algum algoritmo de discretização tenha retornado o vetor de cortes de tamanho k = 5 dado por: {5, 10, 20, 35, 40}. Assim, a tabela 4.2 exibe os intervalos discretizados do atributo peso.

Tabela 4.2: Intervalos de Discretização

Pontos de Corte	10	20	35	40
5	**1**	**2**	**3**	**4**
10	-	**6**	**7**	**8**
20	-	-	**11**	**12**
35	-	-	-	**16**

Considere o intervalo [20, 35] que foi atribuído o número natural 11. O intervalo [35, 40] representa o último intervalo e foi codificado com o número natural 16. Dessa forma, tem-se que $\alpha = 17$ (16 + 1). Portanto, o número que representa a utilização do

atributo peso, quanto este é representado pelo intervalo [20, 35], é dado pelo número 28 (11 + 17). Caso o intervalo [20, 35] não fosse utilizado, este seria representado em um indivíduo pelo número natural 11. Assim, a figura 4.3 apresenta a regra exposta no exemplo 4.1 em que o atributo peso representando o intervalo [20, 35] é utilizado com o uso do MISA para dados contínuos.

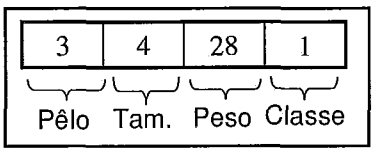

Figura 4.3: MISA para Atributos Contínuos

A operação para verificar se um atributo contínuo é ou não utilizado, depende, como explicado acima, de apenas de uma comparação. Novamente, o MISA fornece grande economia de memória com alta rapidez de utlização.

4.4 Nova Abordagem Híbrida para a Evolução de Teorias

I. Introdução

Antes de descrever a nova abordagem proposta para a evolução de teorias, é importante esclarecer que o sistema proposto trabalha diretamente apenas com problemas que possuam duas classes. Assim, todas as regras de uma teoria são construídas para aprender as instâncias positivas, e qualquer instância que seja coberta pela teoria, i.e., qualquer instância que torne verdadeiro a parte da condição de alguma regra da teoria, é classificada como uma instância da classe positiva. Dessa forma, se uma instância não é coberta por nenhuma regra presente na teoria, esta instância é classificada como sendo da classe negativa.

Para problemas com mais de duas classes, o aprendizado é realizado executando o sistema várias vezes, sendo que em cada execução, a classe a ser aprendida é assumida como classe positiva e todas as outras classes são consideras como uma única classe negativa. Assim, executa-se o sistema em cada classe do problema e a teoria final é construída pela concatenação das várias teorias geradas para cada classe.

II. A Abordagem Proposta

A abordagem proposta adiciona uma nova regra a um indivíduo, somente após as regras previamente adicionadas terem sido evoluídas. Isso significa que o sistema, antes de

adicionar uma nova regra a um indivíduo, tenta explorar o potencial de classificação das regras anteriormente adicionadas. O algoritmo 4.1 exibe essa nova abordagem e os seus principais passos são discutidos em seguida.

Algoritmo 4.1 – Nova Abordagem para Evolução de Teorias

Entrada:
num_gerações: número máximo de gerações;
num_sub_gerações: número máximo de sub-gerações;
k: número natural que determina quando o processo de sub-geração irá ocorrer;
pp: população principal;
ps: população da sub-geração;
e+: conjunto de todas as instâncias positivas no conjunto de treinamento;
e-: conjunto de todas as instâncias negativas no conjunto de treinamento;
Variáveis Locais:
enc+: conjunto de instâncias positivas não cobertos pelo melhor indivíduo em pp.
enc+ é atualizada a cada chamada do AG no passo 3;
Saída:
população **pp** modificada;

```
Começo:
 1 – Inicialize pp aleatoriamente;
 2 – Enquanto (geração < num_gerações) faça:
     3 – pp ←AG(pp, e+ ∪ e- );
     4 – Se ( (geração % k) = 0 ) então
         5 – Inicialize ps aleatoriamente;
         6 – Enquanto (sub-geração < num_sub_gerações) faça
             7 – ps ← AG(ps, enc+ ∪ e-);
             8 – sub-geração ← sub-geração + 1;
             9 – fim-enquanto;
        10 – sub-geração ← 0;
        11 – pp ← pp concatenada com ps;
         fim-se;
    12 – geração ← geração + 1;
    fim-enquanto;
 13 – retorne melhor indivíduo de pp (teoria aprendida);
Fim;

Procedimento AG (pop, conj_de_instâncias): retorna pop modificada;
Começo-Procedimento-AG:
 1 – pop' ← seleção de pais de pop;
 2 – Aplique o operador de recombinação em alguns indivíduos de pop';
 3 – Aplique o operador de mutação em alguns indivíduos de pop';
 4 – Avalie cada indivíduo de pop' com a função de aptidão em conj_de_instâncias;
 5 – pop ← Substituição (pop, pop');
 6 – retorne pop;
Fim-Procedimento-AG:
```

Nossa abordagem para a evolução de teorias trabalha com dois tipos de população: a população principal (pp) e a população da sub-geração (ps). Ambas as populações possuem o mesmo número de indivíduos, mas é importante perceber que os indivíduos de pp irão crescer durante o processo de evolução (regras serão adicionas a cada indivíduo em pp), enquanto cada indivíduo em sp possuirá sempre uma única regra.

Inicialmente, cada indivíduo de pp é inicializado aleatoriamente com uma única regra. Estes indivíduos são evoluídos por um AG no conjunto de treinamento (passo 3). Numa geração k, ou múltipla de k (passo 4), outra instância do AG é executada (Passo 7). Essa nova instância é chamada de sub-geração e trabalha com ps. Quando a sub-geração ocorre, indivíduos com apenas uma regra são criados aleatoriamente em ps (passo 5). Como dito anteriormente, os indivíduos em ps possuem apenas uma única regra, assim, a sub-geração evolui ps num subconjunto (enc+) das instâncias positivas de treinamento, junto com todas as instâncias negativas, sendo que enc+ contém apenas as instâncias positivas não cobertas pelo melhor indivíduo em pp (o que possui o maior valor de aptidão). A sub-geração ocorre até um valor fixo de iterações (num_sub_gerações, passo 6). Quando o processo da sub-geração é encerrado, a regra do indivíduo 1 de ps é adicionada ao indivíduo 1 de pp, a regra do indivíduo 2 de ps é adicionada ao indivíduo 2 de pp e assim por diante, até a regra do último indivíduo em ps seja adicionada ao ultimo indivíduo em pp. Estas regras são adicionadas na última posição da lista de regras que constituem cada indivíduo em pp. Esse processo é chamado de concatenação de populações, e é mostrado no passo 11 do algoritmo 4.1. Perceba que, por este processo, cada vez que o procedimento de sub-geração é chamado, uma nova regra é adicionada a cada indivíduo em pp. Assim, pode-se notar que todos os indivíduos em pp possuem a mesma quantidade de regras e que estas regras foram previamente evoluídas antes de serem adicionadas a pp.

É importante observar que existem dois tipos de evolução de regras em nosso sistema: uma que ocorre no passo 3, denominada de evolução global e outra que acontece no passo 7, chamada evolução local. Para ambas as evoluções, o ciclo básico de um AG, neste algoritmo representado por Procedimento-AG, é executado.

A evolução local trabalha com a população ps cujos indivíduos são constituídos por apenas uma regra. O conjunto de instâncias utilizado na evolução local é reduzido, sendo que este é composto por todas as instâncias negativas, juntamente com todas as instâncias positivas não cobertas pelo melhor indivíduo em pp. Dessa forma, a evolução

local tem o objetivo de desenvolver regras mais focadas a um determinado conjunto de instâncias.

A evolução global trabalha com a população pp que é composta por indivíduos formados por várias regras, sendo que estas regras foram evoluídas durante a evolução local. A evolução global usa todo o conjunto de instâncias de treinamento e com isso ela tenta melhorar os indivíduos de uma forma global, i.e., utilizando todas as regras que foram adicionadas e as evoluindo em todo o conjunto de treinamento. Dessa forma, espera-se que a evolução global, que evolui o indivíduo como todo, e a evolução local, que evolui uma regra focada em um conjunto específico de instâncias, se complementem mutuamente, para que a teoria final possua tanto as vantagens de generalização, que só um método global de evolução pode trazer, juntamente com a capacidade de classificação de instâncias mais especificas, que só um método de evolução local pode alcançar.

O processo de adição de regra é muito simples e reduz o número de regras em um indivíduo, já que as regras adicionadas são previamente evoluídas. Esse procedimento é semelhante ao usado pelo sistema HIDER, entretanto, no HIDER, as regras adicionadas previamente não são evoluídas de forma global como no passo 3 de nossa abordagem.

4.5 Novo Operador de Recombinação Natural para Dados Contínuos

Antes de apresentar o Novo Operador de Recombinação Natural para dados Contínuos (NORNC), uma breve analise do operador de recombinação natural para dados contínuos proposto em (Aguilar-Ruiz et al, 2002), de agora em diante chamado de ORNC, será realizada. Essa análise tem o objetivo de demonstrar a razão que levou ao desenvolvimento do NORNC.

I. Uma Breve Análise do ORNC

O ORNC possui um problema que está relacionado à exploração da matriz que representa os intervalos discretizados. Considere o exemplo abaixo que ilustra este problema.

Exemplo 4.4: Considere o atributo peso apresentado no exemplo 3.1 (capítulo 3). Assuma que algum algoritmo de discretização tenha retornado o vetor de cortes de tamanho k = 5 dado por: {5, 10, 20, 35, 40}. Assim, a tabela 4.3 exibe os intervalos discretizados do atributo peso. Essa tabela é construída para utilização do ORNC.

Tabela 4.3: Intervalos de Discretização para ORNC

Pontos de Corte	10	20	35	40
5	**1**	**2**	**3**	**4**
10	-	**6**	**7**	**8**
20	-	-	**11**	**12**
35	-	-	-	**16**

No capitulo anterior foi apresentado o exemplo em que os intervalos [5, 20] e [10, 40], representados respectivamente pelos números naturais 2 e 8, são recombinados produzindo os intervalos [10, 20] e [5, 40], representados respectivamente por 6 e 4. Ainda, afirmou-se que o ORNC poderia ser entendido como um operador que troca os limites superiores dos intervalos sendo recombinados e dessa forma produz dois novos intervalos. Contudo, se ORNC for usado para realizar a recombinação dos intervalos [5, 10] e [35, 40] representados respectivamente por 1 e 16, um único intervalo será produzido, e ele será o [5, 40], representado pelo número 4, já que:

Recom (1, 16) = { {16}∩{1} } ∪ { {16, 12, 8, 4}∩{1, 2, 3, 4} } } = {4}, (veja definições 3.4 à 3.8 do capítulo 3 caso exista dúvida de como ORNC foi construído).

A utilização do ORNC não produz intervalo [10, 35], como descendente do cruzamento entre [5, 10] e [35, 40]. Este problema reduz a capacidade da exploração deste operador e torna-se pior como o aumento do número de cortes (a matriz se torna maior). Nosso operador objetiva reparar este problema.

II. Proposta do NORNC

O NORNC trabalha com o mesmo vetor de cortes retornado por um algoritmo de discretização, porém, a primeira diferença se faz na numeração dos intervalos retornados e organizados na tabela. O exemplo 4.5 exibe como a numeração dos intervalos retornados é feita para o NORNC.

Exemplo 4.5: Considere o mesmo vetor dos cortes usado no exemplo 4.4, i.e., {5, 10, 20, 35, 40}. A tabela 4.4 exibe numeração dos intervalos obtidos através do vetor de cortes.

Tabela 4.4: Intervalos de Discretização para NORNC

Pontos de Corte	5	10	20	35	40
5	1	2	3	4	5
10	6	7	8	9	10
20	11	12	13	14	15
35	16	17	18	19	20
40	21	22	23	24	25

A matriz é simétrica se considerarmos os valores que limitam cada intervalo. Para essa forma de numeração, têm-se duas equações que serão usadas pelo NORNC. A primeira equação retorna a posição (linha e coluna) na tabela de um dado número natural, e a segunda, por sua vez, retorna o número natural da tabela, dado sua posição (linha e coluna). Considere, portando, as seguintes definições:

Definição 4.1 (Linha lin e Coluna col): Seja n um número natural na tabela de discretização e seja k o número de cortes do vetor de cortes. Assim, a linha (lin) e a coluna (col) de n são, respectivamente, dadas por:

$$lin = q + \text{teto}(r/(r+1)) \tag{4.3}$$

$$col = r + chão((k-r)/k) * k \tag{4.4}$$

onde,

$$q = chão(n/k) \tag{4.5}$$

$$r = n \,\%\, k \tag{4.6}$$

em que: teto(x) retorna o menor inteiro não menor que x; chão(x) retorna o maior inteiro menor ou igual a x; % retorna o resto da divisão inteira.

Considere o exemplo 4.6 que ilustra o uso destas equações.

Exemplo 4.6: Considerando a tabela 4.4 e $n_1 = 2$ e $n_2 = 20$, tem-se:

r(2) = 2 % 5 = 2	r(20) = 20 % 5 = 0
q(2) = chão(2/5) = 0	q(20) = chão(20/5) = 4
lin(2) = 0 + teto(2/3) = 1	lin(20) = 4 + teto(0/1) = 4
col(2) = 2 + chão(3/5)*5 = 2	col(20) = 0 + chão(5/5)*5 = 5

Assim, sabe-se que o número natural 2 está localizado na posição (1, 2) da tabela, enquanto que o número natural 20 está localizado na posição (4, 5).

Considere a próxima definição que calcula o valor de um número natural localizado numa tabela quando se sabe a sua posição, i.e., linha e coluna.

Definição 4.2 (Número n localizado numa tabela de discretização): Dada uma linha lin, uma coluna col e o número de cortes k, o número natural n que ocupa a posição (lin, col) na tabela de discretização, é dado por:

$$n = k * (lin - 1) + col \tag{4.7}$$

Exemplo 4.7: Considere o mesmo vetor dos cortes usado no exemplo 4.4, i.e., {5, 10, 20, 35, 40} e a tabela 4.4 construída com base neste vetor. Deseja-se saber qual o número natural que se localiza na posição (2, 3) desta tabela, ou seja, na linha 2 e coluna 3. Dessa forma, usa-se a equação 4.7 e assim:

n = 5 * (2 - 1) + 3 = 8.

De fato, é o número natural 8 que se localiza na posição (2, 3) da tabela 4.4.

O NORNC é baseado nas duas definições acima e no fato de que a matriz de busca é simétrica (considerando os valores que limitam cada intervalo). O NORNC retorna sempre dois números naturais (filhos) produzidos por outros dois números naturais (pais). Algumas vezes, é necessário corrigir a posição do segundo filho gerado.

Em seguida, apresenta-se o algoritmo que representa a aplicação do NORNC seguido de um exemplo que ilustra o seu uso.

Algoritmo 4.2 – NORNC

Entrada:
pai1 (**p1**), pai2 (**p2**);
número de cortes (**k**).
Saída:
filho1 (**f1**), filho2 (**f2**).

```
Começo
  1 – Ordena_Pais(p1, p2);
  2 – Pega_Linha_Coluna(p1, k, linP1, colP1);
  3 – Pega_Linha_Coluna(p2, k, linP2, colP2);
  4 – f1 ← p1 + (colP2 – colP1);
  5 – f2 ← (p1 + p2) – f1;
  6 – Pega_Linha_Coluna (f2, k, linF2, colF2);
  7 – Se (linF2 > colF2) então:
      8 – f2 ← Pega_Elemento(k, colF2, linF2);
      fim-se;
Fim;
```

O passo 1 é responsável, se necessário, por trocar os valores de p1 e p2. Neste passo, a função Ordena_Pais(), atribui o menor valor entre p1 e p2 a p1 e o maior valor é atribuído a p2. Em outras palavras, o passo 1 coloca p1 como sendo o menor dos pais e p2 como sendo o maior deles. No passo 2, a função Pega_Linha_Coluna(), armazena respectivamente em linP1 e colP1, a linha e a coluna de p1. O passo 3 faz o mesmo que

o passo 2, só que para o p2. A função Pega_linha_Coluna() é baseada na definição 4.1. Os passos 4 e 5 calculam os filhos f1 e f2. O passo 6 pega a linha e a coluna de f2 para verificar se a posição do mesmo precisa ser modificada. O passo 7 corrige caso necessário, a posição de f2 com o seguinte procedimento: se f2 estiver localizado abaixo da diagonal principal (passo 7), então deve-se pegar o número natural simétrico a esse valor. Para isso, a função Pega_Elemento() (baseada na definição 4.2) é chamada com as posições de linF2 e ColF2 invertidas (passo 8). Isso garante que o intervalo é o mesmo, porém o número natural que o representa está correto, ou seja, se situa acima da diagonal principal.

Exemplo 4.8: Considere ainda, o vetor de cortes de tamanho k = 5 dado por: {5, 10, 20, 35, 40}. Assuma um dos casos em que ORNC não retorna todos os filhos possíveis. O caso escolhido é dado pelos intervalos [5, 10] e [35, 40], representados, respectivamente, pelos números 2 e 20 (tabela 4.4). Sabe-se que a prole produzida trocando os limites superiores dos intervalos deve ser f_1 = [5, 40] e f_2 = [10, 35], representados, respectivamente, pelos números naturais 5 e 9 na tabela 4.4. Dessa forma, apresenta-se a execução do algoritmo.

Entrada:
p1 ← 2, p2 ← 20,
k ← 5;

Inicio
1 – p1 ← 2, p2 ← 20; //Não é preciso ordenar os pais, já que p1 < p2.
2 – linP1 ← 1, colP1 ← 2;
3 – linP2 ← 4, colP2 ← 5;
4 – f1 ← (2 + (5 – 2)) = 5;
5 – f2 ← ((2 + 20) – 5) = 17;
6 – linO2 ← 4, colO2 ← 2;
7 – 4 > 2 ? (sim)
f2 ← Pega_Elemento(5, 2, 4); //Pega elemento na linha 2 e coluna 4. Assim,
// f2 ← 9, como esperado.
Fim;

O NORNC foi capaz de retornar os dois intervalos produzidos pelo cruzamento de [5, 10] e [35, 40], enquanto que o ORNC retornou apenas um intervalo. Dessa forma, observa-se que o NORNC possui uma maior capacidade de exploração quando comparado ao ORNC.

É importante perceber que o NORNC atua num único atributo em um indivíduo. Entretanto, o operador de recombinação de dois pontos usando duas regras é utilizado pelo sistema e, dessa maneira, todos os atributos contínuos que se localizem entre os

pontos de corte serão submetidos ao NORNC. Os intervalos produzidos pelo NORNC são então colocados nas duas novas regras produzidas (veja última seção deste capítulo).

4.6 Novo Operador de Recombinação Natural para Dados Discretos

Antes de apresentar o Novo Operador de Recombinação Natural para dados Discretos (NORND), uma breve analise do operador de recombinação natural para dados discretos proposto em (Aguilar-Ruiz et al, 2002), de agora em diante, chamado de ORND, será realizada. Essa análise tem o objetivo de demonstrar a razão que levou ao desenvolvimento do NORND.

I. Uma Breve Análise do ORND

Como apresentado no capitulo 1 deste trabalho, existem dois tipos de operadores que manipulam os genes dos indivíduos da população: o operador de mutação e o operador de recombinação. O primeiro, tem por objetivo inserir novo material genético em um indivíduo e consequentemente numa população, garantindo, assim, que todos os possíveis indivíduos possam ser gerados. O operador de recombinação, por sua vez, tem o objetivo de refinar os indivíduos existentes, i.e., melhorar as soluções já representadas pelos indivíduos. Do ponto de vista da exploração do espaço de busca, o operador de mutação tem o objetivo de gerar novas soluções e dessa forma, garantir que todo o espaço de busca possa ser explorado. Já o operador de recombinação, tem o objetivo de explotar uma determina região, i.e., focar em soluções já existentes com o objetivo de refiná-las e melhora-as ainda mais. A questão de buscar por novas soluções, ou tentar melhorar as soluções já existentes, é conhecido como dilema de exploração/explotação (Holland, 1975). O balanceamento entre exploração e explotação, não apenas efetuados pelos operadores de mutação e recombinação, é essencial para o bom funcionamento do AG e consequentemente para que boas soluções sejam encontradas.

O ORND é baseado no operador de mutação natural (veja tópico IV da seção 3.5.3 do capítulo 3 para maiores informações) também proposto em (Aguilar-Ruiz et al, 2002). Dessa forma, ele se comporta como um operador de mutação, não se aproximando, portanto, da idéia original de troca de material genético proposta pelos operadores de recombinação. Assim, o ORND não se apresenta como um operador de troca de material genético entre dois indivíduos, mas sim como uma mutação entre

ambos, inserindo, portanto, novo material genético na população. O exemplo 4.9 ilustra esse fato.

Exemplo 4.9: Considere o exemplo 3.14 do capítulo 3, em que se deseja realizar a recombinação natural nos números $n_1 = 29$ (11101) e $n_2 = 6$ (00110). Assim, deve-se encontrar a primeira classe de mutação entre n_1 e n_2 cuja interseção entre ambas não seja vazia. Os filhos produzidos serão escolhidos aleatoriamente no conjunto formado por esta interseção. Pelo exemplo 3.9, sabe-se que a classe de mutação de ordem 2 apresenta a primeira interseção não vazia. Assim:

$[\text{Cl-Mut}(29)]^2 \cap [\text{Cl-Mut}(6)]^2 = \{30, 20, 12, 15, 5, 23\}$

Dessa forma, considerando que dois filhos serão gerados por este tipo de recombinação, têm-se o seguinte conjunto de pares de filhos para serem escolhidos aleatoriamente:

{30-20, 30-12, 30-15, 30-5, 30-23, 20-12, 20-15, 20-5, 20-23, 12-15, 12-5, 12-23, 15-5, 15-23, 5-23}.

De outra maneira, se o operador binário de recombinação de dois pontos fosse usado, tendo como pais $n_1 = 29$ (11101) e $n_2 = 6$ (00110), todos os possíveis pares de descendentes, considerando todos os passíveis pontos de corte, seriam dados por:

{28-7, 30-5, 22-3, 31-4, 29-6, 21-14, 23-12, 13-22}.

Verifica-se que quantidade de proles criadas pelo ORND é maior do que a criada com a recombinação binária de dois pontos. Entretanto, ORND cobre apenas um dos casos (30-5) do operador binário. Isso ocorre porque o operador natural é baseado na mutação e não respeita a troca de material genético. Embora existam mais proles geradas pelo ORND, ele não se apresenta como um operador de troca genética entre dois indivíduos, e sim uma mutação entre ambos. Para exemplificar, considere uma das proles criadas pelo ORND:

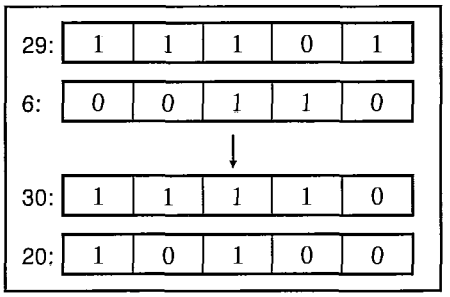

Figura 4.4: ORND

Note pela figura 4.4, que de fato não ocorreu troca genética, e sim mutações em ambos os indivíduos. Dessa forma, o operador ORND, sob ponto de vista da teoria evolucionária, culmina para uma exploração inadequada do espaço de busca.

A próxima seção propõe NORND que apesar de manipular números naturais, é capaz de realizar a recombinação de dois pontos como se a representação binária fosse adotada.

II. Proposta do NORND

O novo operador de recombinação é baseado em operadores *bit-wise* presentes na maioria das linguagens de programação. Estes operadores são os mais rápidos da linguagem de programação, já que são executados diretamente na representação binária do número alocado na memória do computador.

Os operadores usados são os de deslocamento de bit para direita e esquerda, representados respectivamente por >> e <<. Considere o exemplo 4.10 que ilustra o uso destes operadores.

Exemplo 4.10: Considere o número 6 representado em uma variável de 5 bits, assim (00110). O número 6 será deslocado em dois bits para direita e, posteriormente, em 3 bits para esquerda. Assim:

6 (00110) >> 2 = 1 (00001)
6 (00110) << 3 = 16 (10000)

Deve ser notado que não existem círculos neste deslocamento, ou seja, caso não exista mais espaço para o deslocamento para esquerda ou para direita, os bits serão zerados e não colocados no início ou no final da variável. Para ilustrar, considere o mesmo número 6 (00110) deslocado primeiramente em 3 bits para a esquerda e depois em 4 bits para a esquerda, assim:

6 (00110) >> 3 = 0 (00000)
6 (00110) << 4 = 0 (00000)

Como afirmado, NORND utiliza os operadores de deslocamento de bits para a direita e para esquerda para realizar a recombinação genética de dois pontos como se a representação binária fosse usada. Para entender como os operadores de deslocamento de bits podem ser usados com esse propósito, considere o exemplo 4.11 que ilustra uma recombinação de dois pontos entre dois números.

Exemplo 4.11: Considere a *recombinação* binária de dois pontos entre os pais $p_1 = 29$ (11101) e $p_2 = 6$ (00110). Assuma que os pontos de corte foram[2] 1 e 4 (a contagem do ponto de corte começa no número 0). Assim, a recombinação é representada na figura 4.5.

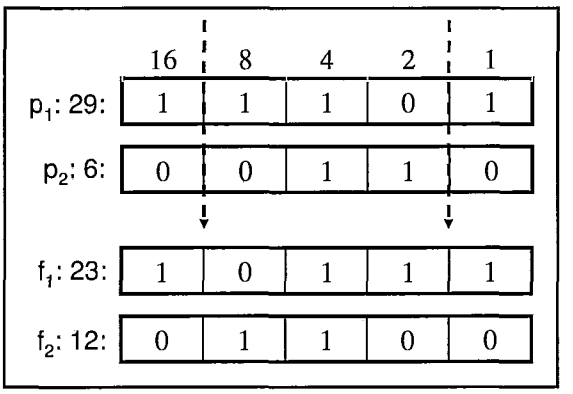

Figura 4.5: Recombinação Binária de dois Pontos

Observe que a parte do p_1 que será trocada é equivalente ao número natural 12, e que a parte de p_2 que será trocada é equivalente ao número natural 6. Assim, o f_1 pode ser calculado como:

$$f_1 = p_1\text{-parte\_trocada\_}p_1 + \text{parte\_trocada\_}p_2 \qquad (4.8)$$

De fato, $f_1 = 23$ (29 - 12 + 6). O mesmo procedimento pode ser utilizado para o cálculo de f_2, só que utilizando o p_2, a parte _trocada_p_2 e a parte _trocada_p_1. Dessa forma, os operadores de deslocamento para direita e para esquerda têm o objetivo de encontrar o valor natural do material genético a ser trocado. Assim que este valor é encontrado, usa-se a equação 4.8 para executar o NORND que explora o espaço de busca como a recombinação binária de dois pontos. Dessa forma, o algoritmo 4.3 representa a aplicação do NORND e é seguido de um exemplo que ilustra o seu uso.

2. A numeração dos pontos de corte para o NORND é considerada da esquerda para direita. Assim, para este exemplo, o ponto de corte entre os bits 01 é considerado o ponto 1 e o ponto entre os bits 11 é considerado o ponto 4.

Algoritmo 4.3 – NORND

Entrada:
pai1 (**p1**), pai2 (**p2**);
pontos de corte X e Y (X < Y);
Saída:
filho1 (**f1**), filho2 (**f2**).

Inicio:
1 - AuxP1 ← p1 >> X;
2 - AuxP1 ← AuxP1 << X;
3 - AuxP1' ← p1 >> Y;
4 - AuxP1' ← AuxP1' << Y;
5 - Parte_Trocada_P1 ← AuxP1 - AuxP1';
6 - AuxP2 ← p2 >> X;
7 - AuxP2 ← AuxP2 << X;
8 - AuxP2' ← p2 >> Y;
9 - AuxP2' ← AuxP2' << Y;
10 - Parte_Trocada_P2 ← AuxP2 - AuxP2';
11 - f1 ← p1 - Parte_Trocada_P1 + Parte_Trocada_P2;
12 - f1 ← p2 - Parte_Trocada_P2 + Parte_Trocada_P1;
Fim;

Os passos de 1 a 4 são necessários para que o cálculo da parte trocada do pai 1 (Parte_Trocada_P1) possa ser realizado no passo 5. Similarmente, os passos de 6 a 8 são necessários para que o cálculo da parte trocada do pai 2 (Parte_Trocada_P2) possa ser realizado no passo 10. Os passos 11 e 12 calculam, com a equação 4.8, os valores dos filhos usando as partes trocadas dos pais.

Exemplo 4.12: Considere a recombinação binária de dois pontos entre os pais $p_1 = 29$ (11101) e $p_2 = 6$ (00110) com pontos de corte 1 e 4. Como apresentado no exemplo 4.11, os filhos f_1 e f_2 retornados para esta recombinação são respectivamente 23 e 12. O NORND será aplicado com objetivo de verificar se ele é capaz de explorar o espaço de busca como o a recombinação binária de dois pontos. Assim:

Entrada:

p1 ← 29 (11101), p2 ← 6 (00110)

X ← 1, Y ← 4;

Início

1 – AuxP1 ← 29 (11101) >> 1 = 14 (01110);

2 – AuxP1 ← 14 (01110) << 1 = 28 (11100);

3 – AuxP1' ← 29 (11101) >> 4 = 1 (00001);

4 – AuxP1' ← 1 (00001) << 4 = 16 (10000);

5 – Parte_Trocada_P1 ← (28 – 16) = 12;

6 – AuxP2 ← 6 (00110) >> 1 = 3 (00011);

7 – AuxP2 ← 3 (00011) << 1 = 6 (00110);

8 – AuxP2' ← 6 (00110) >> 4 = 0 (00000);

9 – AuxP2' ← 0 (00000) << 4 = 0 (00000);

10 – Parte_Trocada_P2 ← (6 – 0) = 6;

11 – f1 ← 29 - 12 + 6 = 23;

12 – f1 ← 6 - 6 + 12 = 12;

Verifica-se, por este exemplo, que os filhos retornados pelo NORND sãos os mesmos produzidos pelo operador binário de dois pontos.

III. Aplicação do NORND em uma Regra

O NORND explora o espaço de busca da mesma forma que a recombinação binária de dois pontos. Entretanto, deve ser notado, que o operador apresentado é aplicável a apenas um atributo em uma regra, e que, no entanto, o operador de recombinação de dois pontos que ocorre entre vários atributos de uma regra é utilizado no sistema. Assim, para que se possa afirmar que as regras são exploradas como se a codificação binária e o operador de recombinação de dois pontos fossem adotados, devem ser considerados quatro casos básicos em que os pontos de corte cortam os atributos para serem submetidos ao operador de recombinação de dois pontos. Os quatro casos são:

Caso 1: Ambos os pontos de corte do operador de recombinação de dois pontos cortam o interior do mesmo atributo em uma regra.

Caso 2: Ambos os pontos de corte do operador de recombinação de dois pontos cortam o interior de atributos diferentes.

Caso 3: Um ponto de corte do operador de recombinação de dois pontos corta o interior de um atributo, enquanto o outro ponto corta o limite de outro atributo.

Caso 4: Ambos os pontos de corte do operador de recombinação de dois pontos cortam os limites de um mesmo atributo ou de atributos diferentes.

Esses quatro casos serão discutidos em seguida e exibidos por meio de um exemplo. Antes, porém, é importante perceber que embora a codificação natural seja adotada, os pontos de corte são sorteados como se a codificação binária fosse usada, ou seja, para uma regra que contenha os atributos pêlo e tamanho, em que ambos podem assumir 3 valores, os pontos de corte podem variar de 0 a 6. Lembre-se, como

apresentado no exemplo 4.11, que a numeração dos pontos de corte para o NORND é considerada da esquerda para direita.

Caso 1: Ambos os pontos de corte do operador de recombinação de dois pontos cortam o interior do mesmo atributo em uma regra.

Neste caso, a aplicação do NORND deve ser realizada nos atributos que foram cortados. O restante das regras permanece inalterado.

Exemplo 4.13: Considere ainda o problema de se classificar um cão em bonito ou feio. Já que a análise é focada no NORND, apenas os atributos discretos serão considerados neste exemplo, portanto, cada cão é descrito através de dois atributos: pêlo (curto, médio, ou longo), tamanho (pequeno, médio, ou grande). A figura 4.6 ilustra duas regras representadas no formato binário, porém com a suas respectivas representações na codificação natural (ver número natural abaixo do número binário). Nesta figura, os dois pontos de corte cortam o interior do mesmo atributo, no caso, o atributo pêlo.

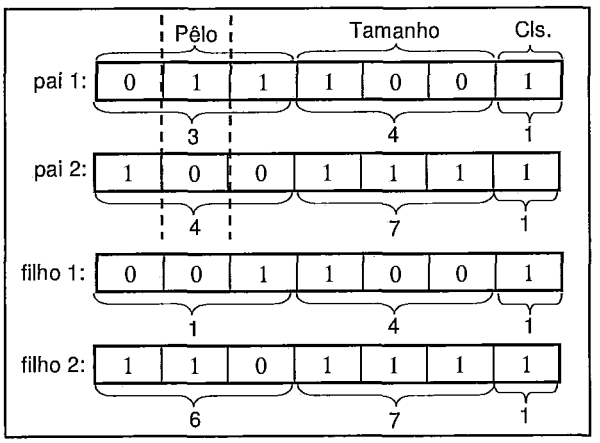

Figura 4.6: Pontos de Corte Cortam o Interior de um Atributo

Neste caso, o NORND deve ser aplicado nos números 3 e 4 com pontos de corte 1 e 2, assim, o NORND retornará os números naturais 1 e 6. Estes números serão copiados para o filho 1 e filho 2 respectivamente Os restos do pai 1 e pai 2 serão copiados respectivamente para filho 1 e filho 2, assim, o NORND é capaz de simular o operador de recombinação binário de dois pontos. Existem duas variações deste caso:

(a) - primeiro ponto de corte se localiza no limite inferior do atributo, enquanto o segundo ponto se localiza no interior deste mesmo atributo;

(b) - o primeiro ponto de corte se localiza no interior do atributo e o segundo ponto se localiza no limite superior deste mesmo atributo;

A figura 4.7 ilustra o primeiro tipo de variação, em que o primeiro ponto de corte se localiza no limite inferior do atributo, enquanto o segundo ponto se localiza no interior do atributo.

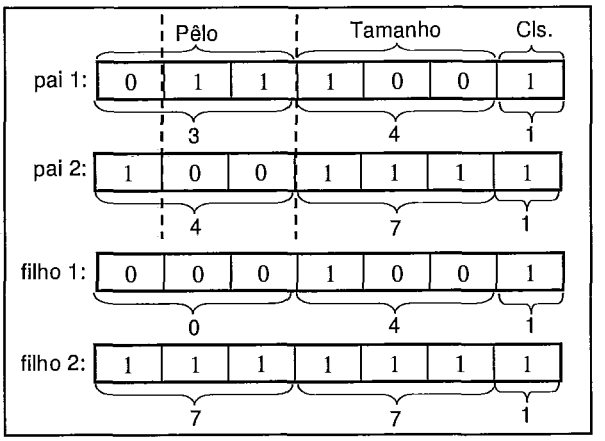

Figura 4.7: Pontos de Corte no Limite Inferior e no Interior de um mesmo Atributo

Nesta variação, como no caso anterior, o NORND deve ser aplicado nos números 3 e 4 só que com pontos de corte 0 e 2. Dessa forma, o NORND retornará os números naturais 0 e 7. Estes números serão copiados para o filho 1 e filho 2 respectivamente. O restante do pai 1 e pai 2 será copiado respectivamente para filho 1 e filho 2, simulando, assim, operador de recombinação binário de dois pontos.

A figura 4.8 ilustra o segundo tipo de variação, em que o primeiro ponto de corte se localiza no interior do atributo e o segundo ponto se localiza no limite superior do atributo.

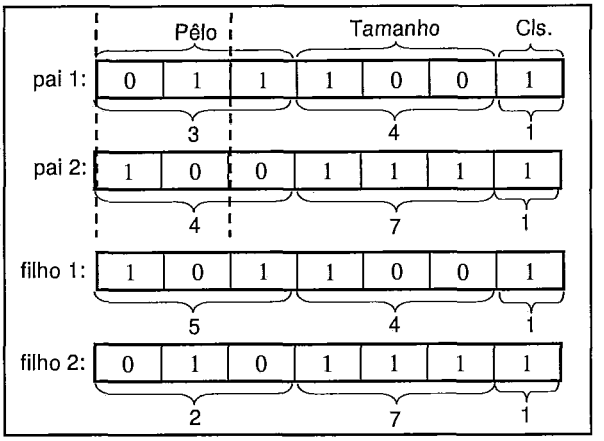

Figura 4.8: Pontos de Corte no Limite Superior e no Interior de um mesmo Atributo

Nesta variação, como no caso anterior, o NORND deve ser aplicado nos números 3 e 4, porém os pontos de corte serão 1 e 3. Dessa forma, o NORND retornará

os números naturais 5 e 2. Estes números serão copiados para o filho 1 e filho 2 respectivamente. Os restos do pai 1 e pai 2 serão copiados respectivamente para filho 1 e filho 2, simulando, assim, operador de recombinação binário de dois pontos.

Caso 2: Ambos os pontos de corte do operador de recombinação de dois pontos cortam o interior de atributos diferentes.

Neste caso, a aplicação do NORND deve ser realizada em ambos os atributos que foram cortados. O restante das regras permanece inalterado.

Exemplo 4.14: Considere o mesmo problema de se classificar um cão em bonito ou feio. Já que a análise é focada no NORND, apenas os atributos discretos serão considerados neste exemplo, portanto, cada cão é descrito através de dois atributos: pêlo (curto, médio, ou longo), tamanho (pequeno, médio, ou grande). A figura 4.9 ilustra duas regras, representadas no formato binário, porém com a suas respectivas representações na codificação natural. Nesta figura, os dois pontos de corte cortam o interior de atributos diferentes, no caso, o atributo pêlo e tamanho são cortados.

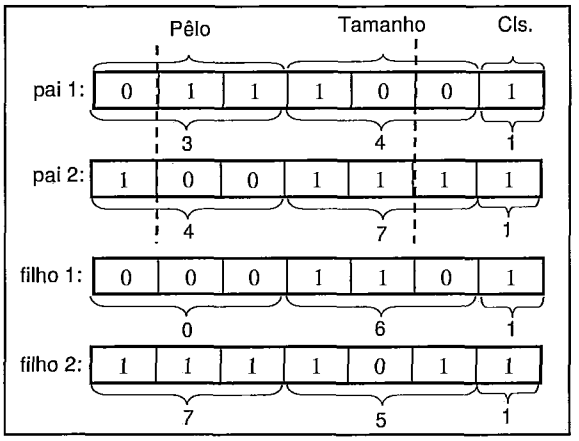

Figura 4.9: Pontos de Corte Cortam o Interior de Atributos Diferentes

Neste caso, o NORND deve ser aplicado aos pares de números 3 e 4, 4 e 7. Para o par 3 e 4, os pontos de corte seriam 0 e 2 e para o par 4 e 7, os cortes seriam 1 e 3. Os números naturais produzidos pelo NORND aplicado ao par 3 e 4 são 0 e 7. Similarmente, o NORND aplicado ao par 4 e 7, produzirá os números 6 e 5. Assim, os números 0 e 6, 7 e 5 serão copiados para o filho 1 e filho 2, respectivamente. O restante do pai 1 e pai 2 será copiado respectivamente para filho 1 e filho 2. Note que, caso existisse outros atributos discretos entre os atributos pêlo e tamanho, os números naturais que os codificam, representados no pai 1 e pai 2, deveriam ser copiados para os

filhos de forma que os atributos de pai 1 fossem copiados para o filho 2 e os atributos do pai 2 fossem copiados para o filho 1 (simples realização da recombinação). Assim, o NORND é capaz de simular o operador de recombinação binário de dois pontos.

Caso 3: Um ponto de corte do operador de recombinação de dois pontos corta o interior atributo enquanto o outro ponto corta o limite (inferior ou superior) de outro atributo.

Neste caso, a aplicação do NORND deve ser realizada no atributo que foi cortado interiormente O restante das regras permanece inalterado.

Exemplo 4.15: Considere o mesmo problema de se classificar um cão em bonito ou feio. Já que a análise é focada no NORND, apenas os atributos discretos serão considerados neste exemplo, portanto, cada cão é descrito através de dois atributos: pêlo (curto, médio, ou longo), tamanho (pequeno, médio, ou grande). A figura 4.10 ilustra duas regras, representadas no formato binário, porém com a suas respectivas representações na codificação natural. Nesta figura, um ponto de corte corta o limite inferior do atributo tamanho, enquanto que o outro ponto corta interiormente o atributo pêlo.

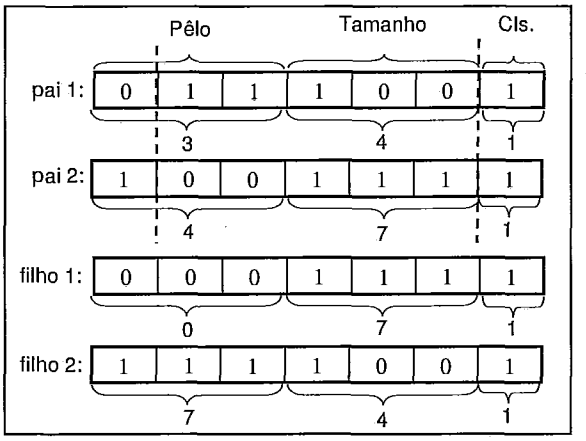

Figura 4.10: Um ponto de corte no interior e outro no limite de um atributo

Neste caso, o NORND deve ser aplicado ao par de números 3 e 4 com os pontos de corte 0 e 2. Com isso, os números naturais 0 e 7 seriam produzidos. Assim, os números 0 e 7 são copiados para o filho 1 e filho 2, respectivamente, enquanto que os números 4 e 7 pertencentes ao pai 1 e pai 2, serão respectivamente copiados para o filho 2 e o filho 1. Dessa forma o NORND é capaz de simular o operador de recombinação binário de dois pontos.

Existe uma variação deste caso, em que um ponto de corte corta o limite superior de um atributo e o outro corta o interior de outro atributo. A figura 4.11 ilustra o caso

em que o primeiro ponto de corte corta interiormente o atributo tamanho, enquanto que o segundo ponto de corte corta o atributo pêlo em seu limite superior.

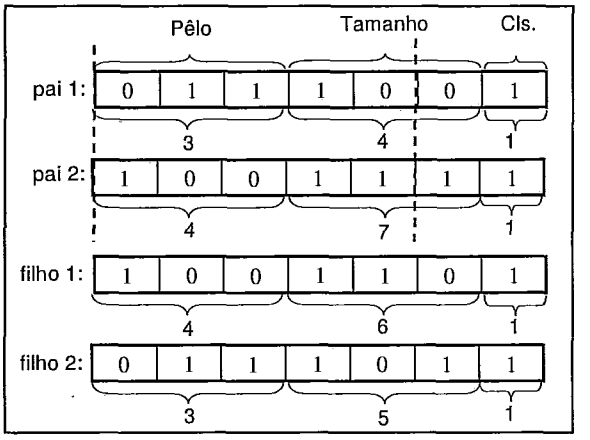

Figura 4.11: Um ponto de corte no interior e outro no limite de um atributo II

Nesta variação, o NORND deve ser aplicado ao par de números 4 e 7 com os pontos de corte 1 e 3, assim, os números naturais produzidos serão 6 e 5. Portanto, 6 e 5 serão copiados para o filho 1 e filho 2, respectivamente, enquanto que os números 3 e 4 pertencentes ao pai 1 e pai 2, serão respectivamente copiados para o filho 2 e o filho 1.

Da mesma forma que no caso 2, se existissem outros atributos discretos entre os atributos pêlo e tamanho, os números naturais que os codificam, representados no pai 1 e pai 2, deveriam ser copiados para os filhos de forma que os atributos de pai 1 fossem copiados para o filho 2 e os atributos do pai 2 fossem copiados para o filho 1 (simples realização da recombinação). Assim o NORND é capaz de simular o operador de recombinação binário de dois pontos.

Caso 4: Ambos os pontos de corte do operador de recombinação de dois pontos cortam o limites de um mesmo atributo ou de atributos diferentes.

Neste caso, o NORND não é aplicado. Apenas invertem-se os atributos localizados entre os pontos de corte para a produção dos dois filhos.

Exemplo 4.16: Considere o mesmo problema de se classificar um cão em bonito ou feio. Já que a análise é focada no NORND, apenas os atributos discretos serão considerados neste exemplo, portanto, cada cão é descrito através de dois atributos: pêlo (curto, médio, ou longo), tamanho (pequeno, médio, ou grande). A figura 4.12 ilustra duas regras, representadas no formato binário, porém com a suas respectivas representações na codificação natural. Nesta figura, ambos os pontos de corte se

localizam nos limites dos atributos. O primeiro ponto de corte corta o limite inferior do atributo tamanho, enquanto que o segundo ponto de corte corta o limite superior da variável pêlo.

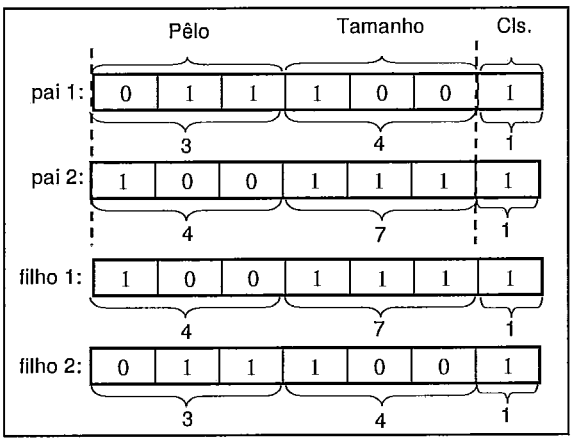

Figura 4.12: Pontos de Corte no Limite dos Atributos

Perceba que, neste caso, basta copiar os números naturais 3 e 4 do pai 1 para o filho 2, e da mesma forma se faz com os números naturais do pai 2, copiando-os para o filho 1. Neste caso, não é necessário à aplicação do NORND para simular o operador de recombinação binário de dois pontos. Caso os pontos de corte se localizassem no limite superior e inferior do mesmo atributo, o mesmo procedimento acima deveria ser feito, porém, utilizando apenas esse atributo.

IV. O Operador de Recombinação de dois Pontos Utilizando o NORND

Considerando os quatros casos expostos na seção anterior, o algoritmo 4.4 apresenta a descrição de recombinação de dois pontos para regras utilizado o NORND.

Algoritmo 4.4 – Operador de Recombinação de dois Pontos para Regras

Entrada:
pai1 (**p1**), pai2 (**p2**): ambos os pais representam regras;
pontos de corte pc1 e pc2 (pc1 < pc2) : sorteados como se a codificação binária fosse usada.
Saída:
filho1 (**f1**), filho2 (**f2**): ambos os filhos representam regras;

```
Inicio:
1 – caso ← Identifica_Caso(p1, p2, pc1, pc2, atr1, atr2, atr3, atr4, pt1, pt2, pt3, pt4);
2 – Se (caso = 1) então
    3 – NORND (atr1, atr2, pt1, pt2, nAtr1, nAtr2)
    4 – Recombinação_Dois_Pontos(p1, p2, nAtr1, nAtr2, f1, f2);
    5 – retorne f1 e f2;
    6 – Finalize o algoritmo;
7 – Se (caso = 2) então
    8 – NORDND (atr1, atr2, pt1, pt2, nAtr1, nAtr2);
    9 – NORDND (atr3, atr4, pt3, pt4, nAtr3, nAtr4);
    10 - Recombinação_Dois_Pontos(p1, p2, nAtr1, nAtr2, nAtr3, nAtr4, f1, f2);
    11 – retorne f1 e f2;
    12 – Finalize o algoritmo;
13 – Se (caso = 3) então
    14 – NORDND (atr1, atr2, pt1, pt2, nAtr1, nAtr2);
    15 – Recombinação_Dois_Pontos(p1, p2, nAtr1, nAtr2, f1, f2);
    16 – retorne f1 e f2;
    17 – Finalize o algoritmo;
18 – Se (caso = 4) então
    19 – Recombinação_Dois_Pontos(p1, p2, pc1, pc2, f1, f2);
    20 – retorne f1 e f2;
    21 – Finalize o algoritmo;
Fim;
```

O algoritmo 4.4 é baseado nos quatro casos explicados na seção anterior. O passo 1 do algoritmo é o responsável por identificar qual dos quatro casos será executado. Assim, os quatro primeiros argumentos da função Identifica_Caso(), são os dois pais e os dois pontos de corte sorteados (pc1 e pc2). As varáveis atr1, atr2, atr3 e atr4 representam os atributos de p1 e p2 que devem ser recombinados, i.e., serem submetidos ao NORND. Os valores destas variáveis são calculados pela função Identifica_Caso() baseando-se em p1, p2 e nos pontos de corte pc1 e pc2. São necessárias quatro variáveis para armazenar os atributos, já que para cada caso existem quantidades diferentes de atributos que devem ser submetidos ao NORND. Assim, no

caso 1, dois atributos serão submetidos ao NORND, um do pai 1 e outro do pai 2; no caso 2, são quatro atributos, sendo dois do pai 1 e dois do pai 2; no caso 3 são dois atributos, sendo um de cada pai, e, finalmente no caso 4, nenhum atributo será submetido ao NORND. As variáveis pt1, pt2, pt3, pt4 representam os pontos de corte que serão usados no NORND. Os valores destas variáveis são também calculados pela função Identifica_Caso() baseando-se em p1, p2 e nos pontos de corte X e Y. Novamente, são necessárias 4 variáveis para se armazenar estes pontos, já que cada caso utiliza uma quantidade diferente de pontos de corte para a aplicação do NORND. Assim, o caso 1 utiliza 2 pontos (pt1 e pt2), o caso 2 utiliza os quatro pontos (pt1, pt2, pt3, pt4), o caso 3 utiliza 2 pontos (pt1 e pt2), e o caso 4 não utiliza nenhum ponto, já que para este caso, não será necessário a aplicação do NORND.

O passo 2 do algoritmo cuida da realização do caso 1, assim, o passo 3 realiza o NORND com os atributos atr1 e atr2 com os pontos de corte pt1 e pt2, retornando os dois novos atributos produzidos (nAtr1 e nAtr2) que serão usados na função Recombinação_de_Dois_Pontos() no passo 4. Essa função é a responsável por alocar os dois novos atributos produzidos (nAtr1 e nAtr2) nos novos filhos f1 e f2, e também por realizar a recombinação dos atributos que não foram submetidos ao NORND (veja a descrição do caso 1). O passo 5 retorna os novos filhos produzidos, e o passo 7 finaliza o algoritmo, já que o caso 1 foi executado.

O passo 7 do algoritmo cuida da realização do caso 2. Lembre-se que, neste caso, dois pares de atributos devem ser submetidos ao NORND, assim, o passo 8 realiza o NORND com os pares de atributos atr1 e atr2 com os pontos de corte pt1 e pt2, enquanto que o passo 9, por sua vez, realiza o NORND com os pares de atributos atr3 e atr4 com os pontos de corte pt3 e pt4. Os novos atributos nAtr1 e nAtr2 produzidos pelo passo 8, juntamente com os novos atributos nAtr3 e nAtr4 produzidos pelo passo 9, são usados na função Recombinação_de_Dois_Pontos() no passo 10. Essa função é a responsável por alocar os quatro novos atributos produzidos (nAtr1, nAtr2, nAtr3, nAtr4) nos novos filhos f1 e f2, e também por realizar a recombinação dos atributos que não foram submetidos ao NORND (veja a descrição do caso 2). O passo 11 retorna os novos filhos produzidos, e o passo 12 finaliza o algoritmo, já que o caso 2 foi executado.

O passo 13 do algoritmo cuida da realização do caso 3, assim, o passo 14 realiza o NORND retornando os dois novos atributos produzidos (nAtr1 e nAtr2) que serão usados na função Recombinação_de_Dois_Pontos() no passo 15. Essa função é a

responsável por alocar os dois novos atributos produzidos (nAtr1 e nAtr2) nos novos filhos f1 e f2, e também por realizar a recombinação dos atributos que não foram submetidos ao NORND (veja a descrição do caso 3). O passo 16 retorna os novos filhos produzidos, e o passo 17 finaliza o algoritmo, já que o caso 3 foi executado.

O passo 18 do algoritmo cuida da realização do caso 4. Lembre-se que, neste caso, não é necessário à realização do NORND. Assim, a função Recombinação_de_Dois_Pontos() (passo 19) recebe diretamente os pontos de corte sorteados para a recombinação de regras, e não os pontos de corte calculados pelo NORND para a recombinação de atributos. A função Recombinação_de_Dois_Pontos() neste caso, atua apenas trocando os atributos entre os pontos de corte pc1 e pc2. Assim, esta função faz com que os atributos entre estes pontos sejam trocados, i.e., os atributos de p1 vão para f2 e os atributos de p2 vão para f1 (veja caso 4). O passo 20 retorna os novos filhos produzidos, e o passo 21 finaliza o algoritmo, já que o caso 4 foi executado.

A seção 4.3 apresentou o MISA para atributos discretos e contínuos, a seção 4.5 apresentou o NORNC enquanto a seção 4.6 apresentou o NORND. Estas propostas foram intencionalmente apresentadas de maneira independente para que as idéias contidas em cada uma delas fossem mais facilmente assimiladas. Entretanto, quando uma regra do sistema é submetida à recombinação de dois pontos, estas três propostas devem ser entendidas em conjunto. Dessa forma, a próxima seção apresenta um exemplo de recombinação de dois pontos que ocorre no sistema proposto. Essa recombinação é realizada considerando o MISA o NORNC e o NORND.

4.7 A Recombinação de dois Pontos Utilizando o MISA, o NORNC e o NORNC

Exemplo 4.17: Considere, ainda, o problema apresentado no capítulo 3 em que se deseja classificar um cão em bonito ou feio. Cada cão é descrito através de três atributos: pêlo (curto, médio ou longo), tamanho (pequeno, médio ou grande) e peso, que foi discretizado e que seu vetor de cortes é dado por {5, 10, 20, 35, 40}. Novamente, apresenta-se a tabela de discretização dos intervalos usadas para o NORNC. Note que agora, que o MISA para dados contínuos utiliza esta tabela.

Tabela 4.5: Intervalos de Discretização para NORNC

Pontos de Corte	5	10	20	35	40
5	1	2	3	4	5
10	6	7	8	9	10
20	11	12	13	14	15
35	16	17	18	19	20
40	21	22	23	24	25

Considere duas regras p1 e p2. Os atributos pêlo e peso são utilizados por p1 que dizem que os cachorros que apresentam pêlo = médio ou longo e peso ∈ [5, 10] são considerados bonitos. Já p2, utiliza apenas o atributo tamanho e diz que os cachorros que apresentam tamanho = pequeno ou grande são considerados bonitos. Assuma os pontos de corte pc1 = 3 e pc2 = 9, para a produção de duas novas regras denominadas f1 e f2. A figura 4.13 ilustra o processo de recombinação de dois pontos.

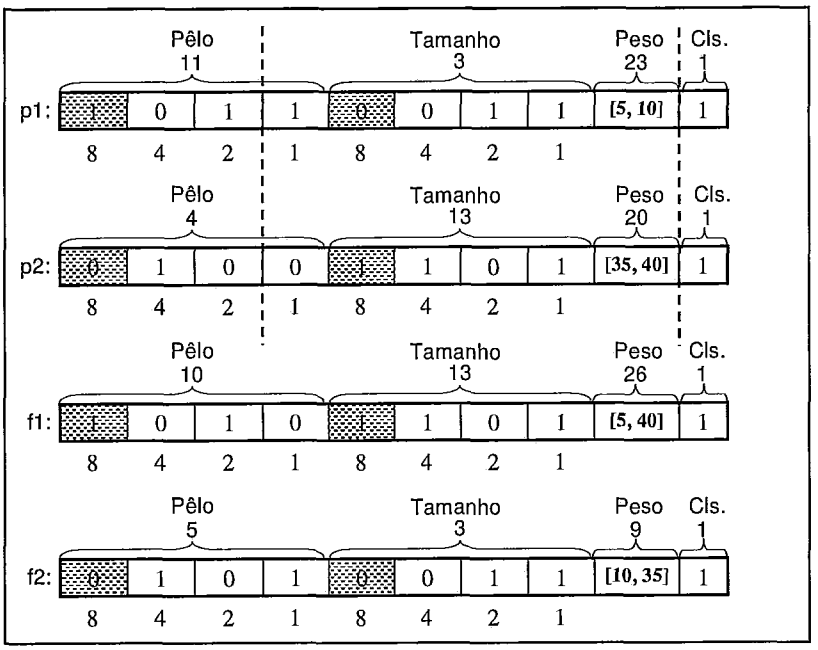

Figura 4.13: Recombinação de Dois Pontos Utilizada no Sistema

Note que o sistema trabalha com a representação natural, portanto p1 é representado pelos números 11, 3 e 23. Da mesma forma, o p2 é representado pelos números 4, 13 e 20. O vetor logo abaixo de cada indivíduo, indica apenas o significado dos números naturais, sendo que para atributos discretos, o vetor significa a disjunção interna dos valores utilizados no atributo, enquanto que para atributos contínuos, o vetor indica o intervalo que o número natural codifica (veja a tabela 4.5). É importante perceber ainda que, para dados discretos, o vetor abaixo de cada atributo representa

como o número natural é armazenado na memória do computador. Para os atributos discretos, os valores relativos à posição de cada bit foram colocados para auxiliar no cálculo dos valores dos números naturais que representam estes tipos de atributos.

Note que o atributo pêlo de p1 é usado, pois já que este atributo pode assumir 3 valores, pelo MISA para atributos discretos, se o valor que ele possui for maior que 8 (2^3), o mesmo é utilizado. Isso é o mesmo que dizer que o bit de número 4 da variável pêlo possui o valor 1. Este bit está rachurado na representação binária do número natural. Note que o atributo tamanho de p1 não é utilizado, pois como este atributo também pode assumir 3 valores, o mesmo seria usado se o valor natural que o representasse fosse maior que 8 (2^3). No entanto, o valor que o representa é 3, e por isso ele não é utilizado.

Para os atributos contínuos, o número α determina se um dado atributo é ou não utilizado. Lembre-se que α é calculado somando-se o valor 1 ao valor natural que representa o último intervalo discretizado. Para a tabela 4.5, tem-se $\alpha = (1 + 20)$, já que 20 foi atribuído ao último intervalo discretizado. Para verificar se um determinado intervalo é usado, compara-se o valor natural que o representa com α. Caso este valor seja maior que α, o intervalo que ele representa é usado; caso seja menor, este intervalo não é usado. Como o atributo peso de p1, que representa o intervalo [5, 10], possui o valor 23 (2 + 21), o mesmo é utilizado nesta regra. Note que o intervalo [5, 10] é representado pelo número natural 2 na tabela 4.5, entretanto, para ele ser utilizado pela regra, o mesmo deve possuir o valor dado pela soma do valor que o representa com o número α, no caso 21. Assim, o valor que representa o intervalo [5, 10] do atributo peso, sendo utilizado pelo sistema, corresponde ao valor 23 (2 + 21). Os mesmos raciocínios utilizados em p1 devem ser realizados para verificar que apenas o atributo tamanho é usado por p2.

Perceba que os números naturais que representam os valores contínuos, sofrem a recombinação de acordo com o NORNC que utiliza a tabela 4.5. Desta forma, a recombinação do intervalo [5, 10], representado pelo número 23 (que afirma que o atributo é usado), com o intervalo [35, 40] representado pelo número 20 (que afirma que este atributo não é usado), retorna os intervalos [5, 40] representado pelo número 5 e o intervalo [10, 35], representado pelo número 9. Entretanto, note que, da mesma forma que apenas o intervalo [5, 10] é utilizado, apenas um dos intervalos gerados será utilizado, e este será o intervalo [5, 40] representado pelo número 26 (5 + 21). O outro

intervalo gerado [10, 35] representado pelo número 9 não é utilizado. Assim, a recombinação dos números naturais 23 e 10, retorna os números naturais 26 e 9. Estes números são então copiados respectivamente para f1 e f2.

Considerando os atributos discretos, o atributo tamanho, representado pelo número natural 3 no p1 e 13 no p2, é simplesmente copiado para f1 e f2. Assim, f1 recebe o valor 13 e f2 recebe o valor 3. Neste caso não foi necessário à aplicação do NORND, já que o ponto de corte não cortou o interior deste atributo. Já para o atributo pêlo, é necessária a aplicação do NORND, como pontos de corte 0 e 1, pois como pode ser notado, o ponto de corte pc1 corta o seu interior. Dessa forma, o NORND retorna os números naturais 10 e 5. Note que também neste caso, a recombinação é realizada sobre um atributo que é utilizado pela regra (atributo com valor 11 em p1), com outro atributo que não é usado (atributo com valor 4 em p2). Obviamente, dois números naturais serão produzidos, sendo que um é utilizado pela regra (número 10) e o outro não (número 5). Como o bit que representa o uso do atributo pêlo não é recombinado, e no caso o p1 utiliza este atributo, enquanto p2 não o utiliza, f1 utilizará este atributo e o f2 não o utilizará. Desta forma, o número natural 10 é copiado para o f1 enquanto o número 5 é copiado para f2.

Com isso, a recombinação de dois pontos é realizada, sendo que para atributos discretos a regra é explorada como se a codificação binária e o operador de recombinação de dois pontos fossem adotados.

4.8 Os Operadores de Mutação

Os operadores de mutação utilizados no sistema são os mesmos usados no sistema HIDER. Assim, para um atributo contínuo, o valor para a mutação de um determinado número natural deve ser escolhido aleatoriamente entre os seus vizinhos (superior, inferior, esquerdo ou direito).

Para atributos discretos, a equação 3.11 apresentada no capítulo 3 é usada, sendo que ela altera o valor de um bit da numa posição k.

Os exemplos seguintes demonstram como estes operadores são usados para ambos os tipos de atributos.

Exemplo 4.18: Considere, novamente, a tabela que exibe os intervalos discretizados para o atributo peso. Para facilitar a consulta, esta tabela é novamente apresentada neste exemplo.

Tabela 4.5: Intervalos de Discretização para NORNC

Pontos de Corte	5	10	20	35	40
5	1	2	3	4	5
10	6	7	8	9	10
20	11	12	13	14	15
35	16	17	18	19	20
40	21	22	23	24	25

Considere o número natural 9 que representa o intervalo [10, 35]. Caso este atributo seja submetido à mutação, o novo intervalo produzido deverá ser escolhido aleatoriamente entre os números 14, 9, 8 e 10. Estes são os todos os vizinhos possíveis do número 9. Claramente, se o número 5 fosse sofrer mutação, apenas os números 4 e 10 poderiam ser escolhidos, sendo que estes são os seus únicos vizinhos. Note que a mutação troca apenas um dos limites do intervalo, por isso os vizinhos considerados são os verticais (acima e abaixo) e os laterais (à esquerda e à direita), já que o vizinho da diagonal significa a troca de ambos os limites.

Exemplo 4.19: Considere o atributo pêlo representado por quatro bits, sendo que o bit na quarta posição é o bit relativo ao MISA. Assuma que este atributo é representado pelo valor 11 (1011). Assim a mutação do primeiro bit deste atributo seria representada por:

$$mut_1(11) = (11 + 2^0) \,\%\, 2^1 + 2^1 * chão\left(\frac{11}{2^1}\right) = 10\,(1010)$$

4.9 Outras Considerações

Operador de Seleção: A seleção por torneio é utilizada. O processo de elitismo é também usado, i.e., o melhor indivíduo da geração atual é copiado para a próxima geração.

Função de Aptidão: A aptidão é simplesmente a da precisão (*accuracy*) obtida no conjunto de dados.

Estratégia de Classificação: A estratégia é semelhante às listas de decisão (Rivest, 1987), i.e., a primeira regra que classifica a instância é de fato utilizada para classificá-la.

Algoritmo de Discretização Utilizado: Foi utilizado o método proposto em (Fayyad e Irani, 1993) e a sua implementação no WEKA (Witten e Frank, 2005).

4.10 Principais Parâmetros

Em relação aos outros sistemas apresentados, o sistema proposto usa utiliza um número bastante reduzido de parâmetros. São eles:

NUM_GERACOES: Número iterações do algoritmo.

NUM_SUB_GERACOES: Número de iterações da sub-geração.

TAM_POP: Tamanho da população, tanto da população principal quanto a população utilizada na sub-geração.

TAM_TOR: Tamanho do torneio usado para a seleção.

k: Valor inteiro que determina quando o processo de sub-geração deve ocorrer.

P_RECOM: Probabilidade de recombinação relativa a um indivíduo.

P_MUTA: Probabilidade de mutação relativa a um indivíduo.

5. Resultados Experimentais

Este capítulo avalia o sistema proposto bem como cada um de seus principais componentes. Primeiramente, cada componente proposto é avaliado para que depois o sistema como um todo possa ser analisado em comparação a outros sistemas de aprendizado de regras.

5.1 Introdução

Este capítulo tem o objetivo de avaliar os componentes principais do sistema proposto, bem como o sistema como um todo em comparação com outros sistemas de aprendizado de regras. Dessa forma, este capítulo está dividido em três seções principais: a primeira exibe os resultados de economia de memória obtidos com a utilização da codificação natural e do MISA para atributos discretos e contínuos. A segunda seção do capítulo avalia separadamente cada um dos principais componentes do sistema. Finalmente, a terceira seção compara o sistema proposto com dois sistemas de aprendizado genético: o GAssist e o XCS. Estes sistemas foram descritos no capítulo 3 deste trabalho e a comparação realizada nesta seção leva em consideração a precisão e a complexidade das teorias obtidas por cada sistema. Adicionalmente, o tempo gasto para execução de cada sistema é também apresentado. Nesta seção, o sistema é também comparado com o clássico algoritmo de árvore de decisão C4.5 (Quinlan, 1993) e RIPPERk (Cohen, 1995).

Antes de efetuar tais avaliações, é necessário que as bases de dados usadas nestas comparações sejam apresentadas. Assim, a próxima seção destina-se a apresentar todas as bases dados utilizadas para a avaliação do sistema.

5.2 Bases de Dados Utilizadas

Esta seção destina-se a apresentar as bases de dados usadas para avaliação e comparação do sistema proposto. No total, 11 bases de dados foram utilizadas e todas elas foram retiradas do UCI (Blake e Merz, 1998).

As bases de dados foram escolhidas de forma a representar diversos tipos de problemas. Assim, alguns dos problemas escolhidos apresentam apenas atributos discretos, outros apresentam apenas atributos contínuos, e a maioria apresenta os dois tipos de atributos. O sistema foi aplicado ainda a problemas com múltiplas classes, i.e.,

problemas de classificação que apresentam mais de duas classes. A diversidade dos problemas escolhidos tem o objetivo de garantir uma melhor estimativa da capacidade de aprendizado do sistema. As bases de dados usadas são apresentadas na tabela 5.1.

Tabela 5.1: Bases de Dados Utilizadas: #Instâncias = Número de Instâncias na Base de Dados; #Atributos = Número de Atributos do Problema; #Atr. Con. = Número de Atributos Contínuos do Problema; #Atr. Dis. = Número de Atributos Discretos do Problema; #I. F. = Número de instâncias que apresentam atributos faltando valores #Classes = Número de Classes do Problema;

Nome	#Instâncias	#Atributos	#Atr. Con.	#Atr. Dis.	#I.F.	#Classes
bre	286	9	-	9	9	2
cr-a	690	15	6	9	37	2
cr-g	1000	20	7	13	0	2
gls	214	9	9	-	0	6
he-c	303	13	6	7	7	2
hep	155	19	6	13	79	2
h-col	368	22	7	15	58	2
ino	351	34	34	-	0	2
son	208	60	60	-	0	2
vel	846	18	18	-	0	4
w-bca	699	9	9	-	16	2

5.3 Economia de Memória Obtida com o uso da Codificação Natural e do MISA

No capitulo anterior, foi apresentado um exemplo que ilustrava a economia de memória obtida pela utilização da codificação natural. Nesta seção, será apresentada a economia obtida com o uso da codificação natural e do MISA para todas as bases de dados utilizadas no trabalho. Dessa forma, esta seção compara a quantidade de memória usada por uma regra quando é utilizada a abordagem apresentada (codificação natural e MISA), com quantidade de memória gasta quando se adota a codificação binária e um mecanismo de seleção de atributos baseado em variáveis do tipo booleano. Para as duas representações, considera-se que os atributos contínuos foram discretizados e estes são representados por um número natural.

Considere a tabela 5.2 que exibe a quantidade de valores que os atributos discretos de cada base de dados podem assumir. Esta tabela é necessária para o calculo da quantidade de memória utilizada por uma regra que use a codificação natural e o MISA.

Tabela 5.2: Domínio dos Atributos Discretos

Nome	Número de Valores dos Atributos
bre	3(2), 2(3), 1(5), 1(6), 1(11), 1(7)
cr-a	4(2), 2(3), 1(4), 1(9), 1(14)
cr-g	2(2), 3(3), 4(4), 3(5), 1(10)
he-c	3 (2), 3(3), 1(4)
hep	13(2)
h-col	2(2), 5(3), 5(4), 2(5), 1(6)

Para a base de dados bre, por exemplo, existem 3 atributos que podem assumir 2 valores, 2 atributos que podem assumir 3 valores, 1 atributo que pode assumir 5 valores, 1 atributo que pode assumir 6 valores, 1 atributo que pode assumir 11 valores e 1 atributo que pode assumir 7 valores. O mesmo raciocínio deve ser seguido para as outras bases de dados.

Note que é a quantidade de valores que os atributos podem assumir que determinam o tipo de variável inteira que será usada na codificação natural. Mais precisamente, é o atributo que pode assumir a maior quantidade de valores que determina a escolha do tipo de variável inteira. Para a base de dados bre, por exemplo, a variável inteira _int8 de um byte (8 bits) é capaz de representar todos os seus atributos, já que a maior quantidade de valores assumidos por um atributo desta base é 7, assim, a variável _int8 será utilizada. Para a base de dados cr-a, a variável inteira _int16 de dois bytes (16 bits) será usada para a representação dos atributos, já que a maior quantidade de valores assumidos por um atributo desta base é 14. Lembre-se que o bit que representa o MISA para dados discretos deve ser levado em consideração para o calculo da variável inteira que representará o atributo.

A tabela 5.3 mostra o tipo de variável usada para representar os atributos discretos quando a codificação natural é usada. O raciocínio utilizado no parágrafo anterior foi usado para determinar o tipo de dado usado.

Tabela 5.3: Tipo de Variável Inteira Usada para na Codificação Natural

Nome	Tipo de Variável Usada	Custo (bytes)
bre	_int8	1
cr-a	_int16	2
cr-g	_int16	2
he-c	_int8	1
hep	_int8	1
h-col	_int8	1

Para a utilização da codificação natural, cada atributo discreto em cada base de dados, gastará a quantidade de memória expressa na coluna Custo da tabela 5.3. Para os

atributos contínuos, a quantidade de memória de 1 byte é suficiente para representar os números naturais que exprimem os intervalos de discretização. É importante perceber que o MISA, como já explicado, não adiciona custo extra de memória à representação da regra. Dessa forma, a tabela 5.4 expõe, para cada base de dados, a quantidade de memória gasta por uma regra quando a codificação natural e o MISA são utilizados.

Tabela 5.4: Memória Gasta por uma Regra com a Codificação Natural e o MISA

Nome	#Atr. Con.	#Atr. Dis.	Custo (Bytes) Atr. Con.	Custo (bytes) Atr. Disc	Custo (Bytes) Total de uma Regra
bre	-	9	-	1	9
cr-a	6	9	1	2	24
cr-g	7	13	1	2	33
gls	9	-	1	-	9
he-c	6	7	1	1	13
hep	6	13	1	1	19
h-col	7	15	1	1	22
ino	34	-	1	-	34
son	60	-	1	-	60
vel	18	-	1	-	18
w-bca	9	-	1	-	9

O cálculo de memória é dado pelo somatório da multiplicação entre os atributos e os seus respectivos gastos de memória. Assim, para o caso cr-a, por exemplo, tem-se que uma regra gasta quantidade de memória dada por (6 * 1) + (9 * 2) = 24, já que ela utiliza 6 atributos contínuos que são representados pela variável inteira de 1 byte, juntamente com 9 atributos discretos, que são representados pela variável _int16 e que gastam 2 bytes cada um.

Para a representação binária, deve ser considerada a quantidade de valores que cada atributo discreto pode assumir. Lembre-se que cada atributo gastará 1 byte (menor unidade de alocação das linguagens de programação) para cada valor que ele pode assumir. Assim, para a base de dados bre, por exemplo, a quantidade de bytes dada por (3 * 2) + (2 * 3) + (1 *5) + (1 * 6) + (1 * 11) + (1 * 7) = 41, será usada para a representação dos atributos discretos. Esse valor foi calculado com base na tabela 5.2 que exibe o domínio para cada atributo discreto. O mesmo raciocínio deve ser usado para calcular a quantidade de bytes utilizados para a representação de cada atributo discreto nas outras bases de dados. A tabela 5.5 exibe esta quantidade para cada base de dados. (Obviamente, apenas as bases de dados que possuem ao menos um atributo discreto são consideradas).

Tabela 5.5: Memória Gasta para Representação dos Atributos Discretos na Codificação Binária

Nome	Custo (Bytes)
bre	41
cr-a	41
cr-g	54
he-c	19
hep	26
h-col	55

Mesmo para a representação binária, considera-se que os atributos contínuos foram discretizados e cada um foi atribuído a um número inteiro de 1 byte. Assim, para cada atributo contínuo, a quantidade de 1 byte é usada mesmo para a representação binária. É importante lembrar, entretanto, que o mecanismo de seleção de atributos mais comumente usado, utiliza 1 byte para cada atributo do problema. Este mecanismo será denominado de Mecanismo Explícito de Seleção de Atributos (MESA) Assim, a tabela 5.6 exibe a quantidade de memória usada por cada regra quando a codificação binária e o MESA são adotados.

Tabela 5.6: Memória Gasta por uma Regra com a Codificação Binária e o MESA

Nome	#Atr. Con.	#Atr. Dis.	Custo (bytes) Atr. Con. Total	Custo (bytes) Atr. Disc Total	Custo (bytes) MESA	Custo Total (bytes)
bre	-	9	-	41	9	50
cr-a	6	9	6	41	15	62
cr-g	7	13	7	54	20	81
gls	9	-	9	-	9	18
he-c	6	7	6	19	13	38
hep	6	13	6	26	19	51
h-col	7	15	7	55	22	84
ino	34	-	34	-	34	68
son	60	-	60	-	60	120
vel	18	-	18	-	18	36
w-bca	9	-	9	-	9	18

Note que a coluna Custo (Bytes) Atr. Con. Total, exibe o total de bytes gastos para representar todos os atributos contínuos de uma base de dados. Este valor é obtido simplesmente multiplicando-se a quantidade de atributos contínuos na base, pela quantidade de bytes gastos, no caso, 1 byte para cada atributo. Assim, para cr-a, por exemplo, essa coluna é calculada pela expressão (6 * 1) = 6. A coluna Custo (Bytes) Atr. Dis. Total foi retirada da tabela 5.5 que exibe o custo total em bytes gastos para representar os atributos discretos com a codificação binária. Já a coluna Custo (bytes) MESA, exibe a quantidade de bytes gastos pelo MESA. Ela é calculada multiplicando-

se a quantidade de atributos por 1 byte, já que cada atributo necessita da variável de 1 byte para informar se ele é ou não utilizado. Assim, para a base cr-a, por exemplo, essa coluna é calculada por ((6 + 9) * 1) = 15, já que existem 6 atributos discretos e 9 contínuos. Finalmente, o custo total é calculado pela soma de todos os custos, i.e, Custo (Bytes) Atr. Con. Total + Custo (Bytes) Atr. Dis. Total + Custo (bytes) MESA. Essa coluna fornece a quantidade de bytes utilizados por uma regra para cada base de dados quando a codificação binária e o MESA são adotados.

A tabela 5.7 compara a quantidade de memória gasta por cada abordagem discutida.

Tabela 5.7: Comparação de Custos de Memória por Regra

Nome	Custo Total (bytes) Cod. Natural + MISA	Custo Total (bytes) Cód. Binário + MESA
bre	9	50
cr-a	24	62
cr-g	33	81
gls	9	18
he-c	13	38
hep	19	51
h-col	22	84
ino	34	68
son	60	120
vel	18	36
w-bca	9	18

A economia de memória é evidente. Para o caso bre, por exemplo, a Cód. Binária + MESA chega a gastar 4 vezes mais memória que a Cod. Natural + MISA. As menores economias de memória obtidas foram para os casos gls, ino, son, vel e w-bca. Isso é devido ao fato de estas bases possuírem apenas atributos contínuos, e, desta forma, a economia é realizada "apenas" pela a utilização do MISA em vez do MESA. Perceba que mesmo para estes casos, a economia de 50% de memória é alcançada. É importante notar que os cálculos de economia de memória foram realizados para uma única regra, e que, para o caso em um indivíduo seja composto por várias regras, a economia de memória se faz ainda mais evidente. A memória economizada poderia ser usada para representar uma maior população ou mesmo pode se fazer necessária para que problemas com muitos atributos (milhares) possam ser tratados diretamente sem depender de procedimentos de pré-processamento.

5.4 Metodologia Experimental

Os resultados para o todos os sistemas foram obtidos usando 5-*fold* validação cruzada (treinamento e teste) e 5-*fold* validação cruzada interna (dividindo o conjunto de treinamento em treinamento e validação) (Mitchell, 1997) (Veja apêndice A). Para atributos contínuos, foi usado o método do discretização proposto em (Fayyad e Irani, 1993) e sua implementação no Weka (Witten e Frank, 2005). Os atributos das instâncias que possuíam valores faltantes foram substituídos pelas médias dos valores para estes atributos. Os parâmetros de todos os sistemas utilizados foram ajustados usando o conjunto de validação de cada "*fold*" para cada série de dados. Como cada base de dados determina um conjunto particular de parâmetros para cada sistema, os parâmetros exibidos para cada sistema são a média dos parâmetros calculados para todas as bases de dados. Os resultados apresentados para cada sistema (número de regras, número de atributos e precisão de uma teoria) são a média dos resultados dos 5 *folds*. Para os sistemas genéticos, devido ao seu comportamento estocástico, o resultado em cada um dos 5 *folds* é obtido calculando-se a média de 20 execuções em cada *fold*. É importante perceber que os mesmos *folds*, para cada base de dados, foram utilizados em cada sistema. Finalmente, T-testes (corrigidos) de Student (Nadeau e Bengio, 2003) usando um intervalo da confiança de 95%, foram usados a fim analisar e comparar os resultados destes testes.

O tempo de execução exibido para cada sistema é a média dos resultados dos 5 *folds*. Não foi levado em consideração o tempo necessário para determinar os parâmetros para cada sistema, assim, o valor de tempo exibe, em média, o tempo gasto para a execução do sistema uma vez que os seus parâmetros tenham sido determinados. Preferiu-se este modo de apresentação de tempo de execução, já que uma vez que se conhece em média o tempo de execução de um sistema, pode-se estimar o tempo para a determinação dos parâmetros do mesmo (o que não é nada mais que várias execuções do sistema sobre conjuntos de dados menores). Finalmente, os experimentos foram realizados num AMD Sempron 1.74 GHz, com 512 megabytes de memória RAM.

5.5 Avaliação dos Componentes do Sistema

Esta seção tem o objetivo de avaliar os principais componentes do sistema. Primeiramente, será avaliado o MISA para atributos contínuos e discretos. Em seguida,

avaliam-se o NORNC e o NORND e finalmente a nova abordagem híbrida para adição de regras será analisada.

Note que, para a avaliação de cada componente do sistema, é preciso, primeiro, obter os resultados para cada base de dados usando o sistema com a utilização de todos os componentes propostos. Essa configuração do AG é denominada AG-MISA-NC-ND-H, já que o AG é usado com o MISA, bem como os novos operadores de recombinação para dados contínuos e discretos (NC e ND) e utiliza a abordagem híbrida (H) para a adição de regras. Para a avaliação de cada componente, o mesmo não será utilizado na nova configuração do sistema e algumas vezes ele será substituído por outro componente já existente na literatura.

Em seguida, apresentam-se os resultados do AG-MISA-NC-ND-H para todas as bases de dados descritas anteriormente. Primeiro, a média dos parâmetros obtidos sobre todas as bases de dados é também apresentada.

Tabela 5.8: Média dos Parâmetros para o AG-MISA-NC-ND-H

AG-MISA-NC-ND-H	
Parâmetro	**Valor**
Tamanho da População	45
Tamanho do Torneio	2
Probabilidade de Recombinação	0,8
Probabilidade de Mutação do Individuo	0,05
Número de Gerações	200
Número para ativar a Sub-geração	50
Número de Sub-gerações	30

Tabela 5.9: Resultados para o AG-MISA-NC-ND-H: Nome = Nome da Base de Dados; #Regras = Número Médio de Regras Utilizadas pela Teoria; #Atributos = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria;

AG-MISA-NC-ND-H			
Nome	**#Regras**	**#Atributos**	**Precisão**
bre	1	2,4	76,1
cr-a	1,4	3,2	92,4
cr-g	2,2	6,6	74,4
gls	9	27,8	78,5
he-c	2,6	6,4	86,8
hep	1,8	3,8	88,4
h-col	1,8	5	84,8
ino	2,4	19,8	91,3
son	1,6	5	79,0
vel	14	82,6	74,0
w-bca	2	4,2	96,1

Os parâmetros exibidos na tabela 5.8, bem como os resultados apresentados na tabela 5.9, por representarem o sistema usando todos os componentes propostos, serão usados como base de comparação para avaliar cada componente isolado.

5.5.1 Avaliação do MISA

A economia de memória alcançada com o uso do MISA foi abordada na seção 5.3 deste capítulo. Esta seção pretende analisar a influência deste mecanismo na obtenção de teorias simples e precisas. Para isso, o sistema foi executado sem o MISA em todas as bases de dados já apresentadas. Como o objetivo é de verificar a influência deste mecanismo no aprendizado das teorias, o mesmo conjunto de parâmetros utilizado na seção anterior foi usado nesta seção. Dessa forma, pode-se analisar até que ponto o MISA influência na complexidade e na precisão das teorias obtidas.

A tabela 5.10 exibe os resultados obtidos pelo sistema sem o MISA (GA-NC-ND-H). Nesta mesma tabela, são apresentados os resultados do sistema completo (GA-MISA-NC-ND-H), exibidos na seção anterior, para que a contribuição do MISA possa ser de fato analisada.

Tabela 5.10: Resultados para o AG-NC-ND-H: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	AG-NC-ND-H			AG-MISA-NC-ND-H			
Nome	**#Reg.**	**#Atr.**	**Pre.**	**#Reg.**	**#Atr.**	**Pre.**	**T-Teste**
bre	1	6	72,4	1	2,4	76,1	Sim
cr-a	1,4	6	77,7	1,4	3,2	92,4	Sim
cr-g	2,2	13,6	71,6	2,2	6,6	74,4	Sim
gls	9	39	71,0	9	27,8	78,5	Sim
he-c	2,6	16	75,8	2,6	6,4	86,8	Sim
hep	1,8	5	82,6	1,8	3,8	88,4	Sim
h-col	1,8	13,2	68,5	1,8	5	84,8	Sim
ino	2,4	79,4	62,0	2,4	19,8	91,3	Sim
son	1,6	16	58,3	1,6	5	79,0	Sim
vel	14	197,6	59,4	14	82,6	74,0	Sim
w-bca	2	18,2	88,9	2	4,2	96,1	Sim

Perceba a notável diferença entre as precisões obtidas pelo AG-MISA-NC-ND-H e o AG-NC-ND-H. Para todos os casos, a precisão obtida pelo AG-MISA-NC-ND-H é estatisticamente significativa considerando o T-Teste com intervalo de confiança de 95%. A quantidade de atributos usados por ambas as versões dos sistemas é também

muito diferente. Para todos os casos, o AG-MISA-NC-ND-H usou um número menor de atributos, chegando a casos em que o esta versão do sistema usa 3 vezes menos atributos que o AG-NC-ND-H para a obtenção de uma precisão bem maior (caso ino). Note que o número de regras utilizadas por ambas as versões do sistema é o mesmo. De fato, como afirmado, os parâmetros usados para o AG-MISA-NC-ND-H foram também utilizados no AG-NC-ND-H, e, desta forma, os parâmetros Numero de Gerações e Numero para Ativar a Sub-Geração, que juntos determinam quantas regras possuirá uma teoria, não foram modificados. Isso tornou a análise ainda mais precisa, pois mostra que quantidade excessiva de atributos utilizados pelo AG-NC-ND-H se deve realmente ao fato da não utilização do MISA, e não devido ao fato de que mais regras foram usadas, o que acarretaria consequentemente a um maior número de atributos na teoria.

Resumidamente, o MISA provou ser um mecanismo importante para a evolução de teorias que possuam alta precisão e utilizem um número reduzido de atributos.

5.5.2 Avaliação do NORNC

Esta seção tem o objetivo de avaliar o novo operador de recombinação natural contínuo (NORNC) proposto neste trabalho. Como afirmado no capítulo anterior, esse operador possui uma maior capacidade de explorar a matriz de discretização quando comparado ao operador de recombinação natural contínuo (ORNC) proposto em (Aguilar-Ruiz et al, 2002).

Para avaliar a capacidade de exploração do NORNC este foi substituído no sistema e em seu lugar utilizou-se o ORNC, dando origem ao sistema GA-MISA-C-ND-H. Apenas as bases de dados que possuem somente atributos contínuos foram usadas nesta avaliação, já que se pretende analisar o NORNC. Assim, as bases gls, ino, son, vel e w-bca foram usadas. A tabela 5.11 exibe a média dos parâmetros utilizados por essa configuração do sistema. Adicionalmente, como base para comparação, esta tabela exibe os parâmetros utilizados pelo AG que utiliza NORNC, representado por AG-MISA-NC-ND-H. Estes parâmetros foram retirados da tabela 5.8.

Tabela 5.11: Média dos Parâmetros para o AG-MISA-C-ND-I

Parâmetros	AG-MISA-C-ND-I	AG-MISA-NC-ND-I
Tam. da População	50	45
Tam. do Torneio	2	2
Prob. de Recombinação	0,8	0,8
Prob. de Mutação do Individuo	0,05	0,05
Num. de Gerações	250	200
Num. para ativar a Sub-Geração	60	50
Num. de Sub-gerações	40	30

Para as séries de dados examinadas, o AG-MISA-C-ND-H obteve os mesmos resultados obtidos pelo AG-MISA-NC-ND-H. Observe, entretanto, na tabela 5.11, que em comparação com o AG-MISA-NC-ND-H, o AG-MISA-C-ND-H teve os parâmetros tamanho da população, número de gerações e número de sub-gerações aumentados respectivamente em 11%, 25% e 33.3%. Estes resultados mostram que, como esperado, o NORNC possui uma maior capacidade de exploração que o ONRC.

5.5.3 Avaliação do NORND

Esta seção tem o objetivo de avaliar o novo operador de recombinação natural discreto (NORND) proposto neste trabalho. Como afirmado no capítulo anterior, esse operador possui a capacidade de explorar o espaço de busca como se a codificação binária e o operador de recombinação de dois pontos fossem utilizados. Isso não acontece com o operador de recombinação natural discreto (ORND) proposto em (Aguilar-Ruiz et al, 2002). Como mostrado na seção 4.6-I do capítulo 4, o ORND apresenta-se como um operador de mutação que utiliza dois indivíduos, e não como um operador de recombinação.

Para avaliar a capacidade de exploração do NORND este foi substituído no sistema e em seu lugar utilizou-se o ORND, dando origem ao sistema GA-MISA-NC-D-H. As bases de dados que apresentassem ao menos um atributo discreto foram utilizadas nesta comparação, assim, as bases usadas foram: bre, cr-a, cr-g, he-c, hep e h-col. Note que ambas as versões do sistema usam o MISA e o NORNC, ou seja, a única diferença entre estas duas versões é realmente o operador de recombinação para dados discretos. Dessa forma, garante-se que as diferenças de resultados entre as duas versões do sistema são devidas a utilização de diferentes tipos de operadores de recombinação para dados discretos.

A tabela 5.12 exibe a média dos parâmetros utilizados pelo GA-MISA-NC-D-H. Adicionalmente, como base para comparação, esta tabela exibe os parâmetros utilizados pelo AG que utiliza NORND, representado por AG-MISA-NC-ND-H. Estes parâmetros foram retirados da tabela 5.8..

Tabela 5.12: Média dos Parâmetros para o AG-MISA-NC-D-I

Parâmetros	AG-MISA-NC-D-I	AG-MISA-NC-ND-I
Tam. da População	50	45
Tam. do Torneio	2	2
Prob. de Recombinação	0,8	0,8
Prob. de Mutação do Individuo	0,05	0,05
Num. de Gerações	300	200
Num. para ativar a Sub-Geração	60	50
Num. de Sub-gerações	60	30

A tabela 5.13 exibe os resultados obtidos com a utilização do sistema AG-MISA-NC-D-H para as bases de dados testadas. Esta tabela também exibe os resultados alcançados pelo sistema em que o NORND foi usado, ou seja, AG-MISA-NC-ND-H. Estes resultados foram expostos na tabela 5.9. A tabela 5.13 compara os resultados das duas versões do sistema.

Tabela 5.13: Comparação entre AG-MISA-NC-D-H e AG-MISA-NC-ND-H: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	AG-MISA-NC-D-H			AG-MISA-NC-ND-H			
Nome	#Reg	#Atr	Precisão	#Reg	#Atr	Precisão	T-Teste
bre	1,6	6	73,0	1	2,4	76,1	Sim
cr-a	2,2	8	90,8	1,4	3,2	92,4	Não
cr-g	4,2	13	71,5	2,2	6,6	74,4	Não
he-c	4	10,4	82,6	2,6	6,4	86,8	Sim
hep	2,8	6,2	82,9	1,8	3,8	88,4	Sim
h-col	3	7,6	83,8	1,8	5	84,8	Não

O NORND obtém melhores resultados em todas as 6 séries de dados; em 3 deles, os resultados são estatisticamente significativos. Para todas as séries de dados, o AG-MISA-NC-ND-H usou um menor número de regras e de atributos. Observe que o MISA é usado nas duas versões do sistema, entretanto, o que indiretamente adiciona atributos as teorias de AG-MISA-NC-D-H é o fato que, para estes testes, o uso do ORND implica no uso de mais regras para se atingir uma precisão comparável ao

sistema AG-MISA-NC-ND-H. Como o ORND trabalha como um operador de mutação, este operador parece ser incapaz de fazer o ajuste fino em algumas regiões promissoras do espaço de busca. Dessa forma, o algoritmo é "forçado" a adicionar regras extras para suprir esta falta de exploração das regras anteriormente adicionadas. Este parece ser um tipo de dilema de da exploração/explotação (Eiben e Schippers, 1998), onde o ORND possui capacidade para encontrar as boas regiões a serem exploradas no espaço de busca (exploração) enquanto não parece ter habilidade suficiente de explorar estas áreas de uma maneira mais refinada (explotação).

Verifica-se pela tabela 5.12, que o tamanho da população, o número das gerações e o número de sub-gerações para AG-MISA-NC-D-H em comparação com o AG-MISA-NC-ND-H, foram aumentados, respectivamente, em 11%, 50% e em 100%. Isso mostra que o ORND, além de possuir uma capacidade de exploração inferior ao NORND, necessita, ainda, de mais tempo para explorar o espaço de busca.

5.5.4 Avaliação da Abordagem Híbrida para Evolução de Teorias

No capítulo anterior, foi apresentado o modelo adotado pelo sistema para a evolução de uma teoria. Resumidamente, existem dois tipos de evolução, uma local e outra global, que trabalham respectivamente com a população principal e a população sub-geração.

A evolução local tem o objetivo de evoluir indivíduos que possuam uma única regra para que depois essas regras sejam adicionadas aos indivíduos da população principal. Esta evolução é executada sobre um conjunto reduzido de instâncias que é composto por todas as instâncias negativas e apenas pelas instâncias positivas que não foram classificadas pelo melhor indivíduo da população principal. A evolução global, por sua vez, tem o objetivo de evoluir os indivíduos compostos por várias regras sobre todo o conjunto de instâncias. Acredita-se, dessa forma, que a combinação destes dois tipos de evolução tenham sido de fundamental importância para a obtenção dos bons resultados alcançados pelo sistema.

Essa seção avalia cada uma destas duas evoluções com o objetivo de verificar até que ponto cada uma influencia no aprendizado de uma teoria. Dessa forma, duas novas versões do sistema foram construídas. A primeira versão é denominada AG-MISA-NC-ND-G, em que o componente de evolução local foi retirado, ou seja, apenas o procedimento de evolução global é utilizado. A segunda versão denomina-se AG-MISA-NC-ND-L, em que o componente de evolução global foi retirado, ou seja, apenas

o processo de evolução local é executado. As próximas seções fornecem mais detalhes sobre as novas versões do sistema e discutem os resultados obtidos por cada uma delas.

I. O Sistema AG-MISA-NC-ND-G

Como afirmado, essa versão do sistema trabalha apenas com o componente global de evolução. Dessa forma, essa versão aproxima da abordagem Pittsburgh, já que os indivíduos representam teorias a serem evoluídas.

O processo de adição de regras contínua o mesmo, porém, as regras adicionadas não são previamente evoluídas por uma nova instância do AG, ou seja, o processo de sub-geração não ocorre. Assim, o parâmetro Número para ativar a Sub-Geração, determina apenas que novas regras (uma para cada indivíduo da população principal) serão aleatoriamente criadas e adicionadas a cada indivíduo da população principal. O parâmetro Número de Sub-gerações não é mais utilizado, já que a sub-geração não ocorre. Adicionalmente, um operador de remoção de regras foi usado. Ele remove as regras dos indivíduos que não classificam nenhuma instância positiva. Esse operador é aplicado durante o cálculo das aptidões em todas as iterações do sistema.

Para este sistema, novos parâmetros foram determinados como explicado na seção 5.4 deste capítulo. A tabela 5.14 exibe a média dos parâmetros utilizados por essa configuração do sistema. Adicionalmente, como base para comparação, esta tabela exibe os parâmetros utilizados pelo AG que usa a abordagem híbrida de evolução, representado por AG-MISA-NC-ND-H. Estes parâmetros foram retirados da tabela 5.8.

Tabela 5.14: Média dos Parâmetros para o AG-MISA-NC-ND-G

Parâmetros	AG-MISA-NC-ND-G	AG-MISA-NC-ND-H
Tam. da População	70	45
Tam. do Torneio	3	2
Prob. de Recombinação	0,8	0.8
Prob. de Mutação do Individuo	0,05	0,05
Num. de Gerações	400	200
Num. para ativar a Sub-Geração	45	50
Num. de Sub-gerações	-	30

A tabela 5.15 exibe os resultados obtidos pelo sistema AG-MISA-NC-ND-G. Adicionalmente, como base para comparação, esta tabela exibe os resultados utilizados obtidos pelo AG-MISA-NC-ND-H. Estes resultados foram retirados da tabela 5.9.

Tabela 5.15: Resultados para o AG-MISA-NC-ND-G: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	AG-MISA-NC-ND-G			AG-MISA-NC-ND-H			
Nome	**#Reg.**	**#Atr.**	**Pre.**	**#Reg.**	**#Atr.**	**Pre.**	**T-Teste**
bre	2,3	5,8	73,1	1	2,4	76,1	Sim
cr-a	3,5	7,3	91,2	1,4	3,2	92,4	Não
cr-g	5,6	11,5	70,5	2,2	6,6	74,4	Sim
gls	18	65,8	75,5	9	27,8	78,5	Sim
he-c	6,3	15,4	84,3	2,6	6,4	86,8	Não
hep	4,8	7,9	87,4	1,8	3,8	88,4	Não
h-col	3,9	12,5	83,9	1,8	5	84,8	Não
ino	4,3	41,5	87,3	2,4	19,8	91,3	Sim
son	3,2	9,5	78,4	1,6	5	79,0	Não
vel	25,9	170,5	73,5	14	82,6	74,0	Não
w-bca	5	10,5	95,3	2	4,2	96,1	Não

Considerando as precisões obtidas pelos sistemas, em 4 séries de dados (bre, cr-g, gls, ino) os resultados obtidos pelo AG-MISA-NC-ND-H são estatisticamente significativos quando comparados aos resultados de AG-MISA-NC-ND-G. Para as outras 7 bases de dados, as precisões são equivalentes. Perceba, entretanto, que a complexidade das teorias desenvolvidas pelo AG-MISA-NC-ND-G é bem maior, chegando, na maioria dos casos, a possuir o dobro de regras e atributos quando comparadas as teorias desenvolvidas pelo AG-MISA-NC-ND-H.

É importante notar que em relação à configuração do AG-MISA-NC-ND-H, os parâmetros Tamanho da População e Numero de Gerações do AG-MISA-NC-ND-G, foram aumentados em respectivamente 55% e 100%. O aumento do tamanho da população mostra que essa versão do sistema não é capaz de explorar o espaço de busca da mesma maneira que o AG-MISA-NC-ND-H. Perceba, ainda, que o parâmetro Número para Ativar a Sub-geração teve o seu valor decrescido em 10% quando comparado ao AG-MISA-NC-ND-H. Esse decréscimo, juntamente com o aumento do número de gerações, implica no uso de uma maior quantidade de regras por indivíduo, o que faz, obviamente, com que a teoria possua mais atributos e dessa forma seja mais complexa.

Os fatos apresentados determinam que a fase de evolução local é muito importante para que teorias simples e precisas sejam desenvolvidas.

II. O Sistema AG-MISA-NC-ND-L

Esta versão do sistema possui apenas a etapa de evolução local, ou seja, não é utilizada a fase de evolução global que trabalha com indivíduos que constituem uma teoria. Assim, cada indivíduo da população representa uma única regra, e, por isso, essa versão do sistema se aproxima mais da abordagem Michigan, já que nesta, cada indivíduo da população representa uma única regra.

No AG-MISA-NC-ND-L, a teoria é construída passo a passo, sendo que a cada etapa uma nova regra é adicionada à teoria. Assim, o processo de sub-geração do AG é chamado várias vezes, e a cada execução da sub-geração, o melhor indivíduo da população (maior aptidão) é usado para construir a teoria final. As chamadas da sub-geração trabalham com cada vez menos instâncias, já que as instâncias cobertas pelo melhores indivíduos de ciclos anteriores são sucessivamente retiradas. Dessa forma, esta versão do sistema se comporta como um típico algoritmo de cobertura. É importante perceber que a teoria é construída passo a passo, e que as regras que já foram selecionadas para compor a teoria não sofrem mais nenhum tipo de evolução. O parâmetro Número de Gerações determina juntamente com o parâmetro Num. para ativar a Sub-Geração, quantas vezes o processo de sub-geração será chamado. A teoria, como afirmado, é construída pelos melhores indivíduos (com maior aptidão) selecionados de cada chamada da sub-geração. Esse tipo de evolução é idêntico ao processo utilizado pelo sistema HIDER (apresentado no capítulo 3).

Novamente, para este sistema, novos parâmetros foram determinados como 5.4 deste capítulo. A tabela 5.16 exibe a média dos parâmetros utilizados por essa configuração do sistema. Adicionalmente, como base para comparação, esta tabela exibe os parâmetros utilizados pelo AG que utiliza a abordagem híbrida de evolução, representado por AG-MISA-NC-ND-H. Estes parâmetros foram retirados da tabela 5.8.

Tabela 5.16: Média dos Parâmetros para o AG-MISA-NC-D-L

Parâmetros	AG-MISA-NC-D-L	AG-MISA-NC-ND-H
Tam. da População	65	45
Tam. do Torneio	2	2
Prob. de Recombinação	0,8	0,8
Prob. de Mutação do Individuo	0,05	0,05
Num. de Gerações	15	200
Num. para ativar a Sub-Geração	1	50
Num. de Sub-gerações	200	30

Perceba que o parâmetro Número para Ativar a Sub-geração possui o valor 1. Isso significa que o processo de sub-geração será sempre executado, pois essa é maneira usada para a construção da teoria. O parâmetro Número de Gerações controla o término do algoritmo, enquanto o parâmetro Num. de Sub-gerações determina quantas iterações serão executadas em cada chamada da sub-geração.

A tabela 5.17 exibe os resultados obtidos pelo sistema AG-MISA-NC-ND-L. Adicionalmente, como base para comparação, esta tabela exibe os resultados utilizados obtidos pelo AG-MISA-NC-ND-H. Estes resultados foram retirados da tabela 5.9.

Tabela 5.17: Resultados para o AG-MISA-NC-ND-L: Nome = Nome da Base de Dados; #Reg. = Número de Regras Utilizadas pela Teoria; #Atr. = Número de Atributos Utilizados pela Teoria; Precisão = Precisão (*acuraccy*) Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	AG-MISA-NC-ND-L			AG-MISA-NC-ND-H			
Nome	**#Reg.**	**#Atr.**	**Pre.**	**#Reg.**	**#Atr.**	**Pre.**	**T-Teste**
bre	3,8	12	74,5	1	2,4	76,1	Não
cr-a	6,1	15,8	92	1,4	3,2	92,4	Não
cr-g	5,3	20,5	71,5	2,2	6,6	74,4	Sim
gls	35	128,8	74,5	9	27,8	78,5	Sim
he-c	9,5	28,5	83,3	2,6	6,4	86,8	Sim
hep	7,2	16,5	85,4	1,8	3,8	88,4	Sim
h-col	8,5	26,7	84,8	1,8	5	84,8	Não
ino	10,5	70,5	88,5	2,4	19,8	91,3	Não
son	6,4	20,8	79,0	1,6	5	79,0	Não
vel	60,5	333,5	73,5	14	82,6	74,0	Não
w-bca	9	17,8	92,3	2	4,2	96,1	Não

Considerando as precisões obtidas pelos sistemas, em 4 séries de dados (cr-g, gls, he-c, hep) os resultados obtidos pelo AG-MISA-NC-ND-H são estatisticamente significativos quando comparados aos resultados de AG-MISA-NC-ND-L. Para as outras 7 bases de dados, as precisões são equivalentes. Note, entretanto, que a complexidade das teorias desenvolvidas pelo AG-MISA-NC-ND-L é bem maior, chegando, na maioria dos casos, a possuir o triplo de regras e atributos quando comparadas as teorias desenvolvidas pelo AG-MISA-NC-ND-H.

Observe na tabela 5.16 que o tamanho da população em relação ao AG-MISA-NC-ND-H, sofreu um acréscimo de 44%. É importante perceber que embora o número de gerações para o AG-MISA-NC-ND-H seja pequeno, o número de iterações de cada ciclo de sub-geração é elevado. Isso se deve ao fato de que o processo de evolução global não existe nesta configuração do sistema, ou seja, a evolução local é a única

responsável por realizar todo o aprendizado e, dessa forma, deve ser executadas mais vezes e com um maior número de iterações.

Pelos resultados apresentados nesta seção, fica claro que a abordagem hibrida utilizada é de imenso valor para o desenvolvimento de teorias simples e precisas. As outras versões do sistema adquirem resultados competitivos, porém a teoria se apresenta bem mais complexa. Esse fato demonstra que os processos de evolução global e local se complementam, e, dessa forma, o sistema é capaz de obter teorias precisas com um número reduzido de regras e de atributos.

A próxima seção dedica-se a comparar o sistema proposto com outros algoritmos de aprendizado de regras.

5.6 Avaliação Global do Sistema

Essa seção compara o sistema proposto com outros quatro algoritmos de aprendizado de regras. Primeiramente, o sistema é comparado com o algoritmo clássico de árvore de decisão C4.5 (Rivest, 1987) e com o RIPPER (Cohen, 1995). As duas seções seguintes comparam o sistema com o GAssist (Bacardit, 2000) e o XCS (Wilson, 1995), que são sistemas genéticos para aprendizado de regras e foram explicados no capítulo 3 deste trabalho. Finalmente, uma visão geral dos resultados obtidos por todos os sistemas é apresentada com objetivo de fornecer uma visão geral dos resultados obtidos pelo sistema proposto.

5.6.1 Comparação com o C4.5

A implementação do C4.5 no WEKA (Witten e Frank, 2005) foi usada para esta comparação. Os parâmetros deste sistema foram determinados seguindo a metodologia explicada na seção 5.4 deste capítulo. A tabela 5.18 exibe estes parâmetros. A tabela 5.19 apresenta a comparação entre os resultados obtidos pelo o C4.5 e o sistema proposto, i.e., o GA-MISA-NC-ND-H.

Tabela 5.18: Parâmetros para o C4.5

Parâmetros	Valor
fatConf:	0,20
numMinObj:	4
numFolds:	6
podaErroReduzido:	sim
crescimento:	sim

Tabela 5.19: Resultados para o AG-MISA-NC-ND-H e C4.5: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	C4.5			GA-MISA-NC-ND-H			
Nome	#Reg.	#Atr.	Pre.	#Reg.	#Atr.	Pre.	T-Teste
bre	7	18	74.7	1	2,4	76,1	Não
cr-a	15	72	89.3	1,4	3,2	92,4	Sim
cr-g	60	270	73.4	2,2	6,6	74,4	Não
gls	25	97	74.4	9	27,8	78,5	Sim
he-c	26	83	82.1	2,6	6,4	86,8	Sim
hep	7	30	84.4	1,8	3,8	88,4	Sim
h-col	13	49	83.7	1,8	5	84,8	Não
ino	24	71	89.8	2,4	19,8	91,3	Não
son	12	53	75.1	1,6	5	79,0	Sim
vel	140	699	71.3	14	82,6	74,0	Sim
w-bca	30	65	92.5	2	4,2	96,1	Sim

O sistema GA-MISA-NC-ND-H obteve melhores resultados em todas as 11 bases de dados testadas; em 7 delas, os resultados são estatisticamente significantes. As bases de dados em que os T-Testes não foram significantes foram bre, cr-g, h-col e ino. Perceba o pequeno número de regras e atributos usados pelo GA-MISA-NC-ND-H em comparação ao C4.5. Na base de dados son, a que possui a maior quantidade de atributos, a teoria obtida pelo GA-MISA-NC-ND-H usa apenas 5 deles, enquanto o C4.5 usa 53 atributos. Embora as precisões sejam equivalentes para as bases h-col e ino, é importante perceber que as teorias evoluídas pelo GA-MISA-NC-ND-H são bem mais simples do que as teorias evoluídas pelo C4.5. Para o caso h-col, por exemplo, a teoria do GA-MISA-NC-ND-H consta de 1,8 regras e 5 atributos enquanto que a teoria desenvolvida pelo C4.5 possui 13 regras e usa 49 atributos. Estes resultados mostram que o sistema proposto é capaz de evoluir teorias mais simples e mais precisas do que as teorias desenvolvidas pelo algoritmo de árvore de decisão C4.5.

A tabela 5.20 exibe o tempo gasto para ambos os sistemas para alcançar os resultados apresentados. Obviamente, o tempo usado pelo C4.5 é bem menor.

Tabela 5.20: Resultados de Tempo para C4.5 e AG-MISA-NC-ND-H

	C4.5	AG-MISA-NC-ND-H
Nome	Tempo (s)	Tempo (s)
bre	0,38	32,93
cr-a	0,03	97,21
cr-g	0,05	134,52
gls	0,02	123,87
he-c	0,02	44
hep	0,01	25,95
h-col	0,02	68,89
ino	0,02	117,64
son	0,02	46,89
vel	0,08	250,78
w-bca	0,02	81,11

5.6.2 Comparação com o RIPPER

A implementação do RIPPER no WEKA (Witten e Frank, 2005) foi usada para esta comparação. Os parâmetros deste sistema foram determinados seguindo a metodologia explicada na seção 5.4 deste capítulo. A tabela 5.21 exibe estes parâmetros. A tabela 5.22 apresenta a comparação entre os resultados obtidos pelo o RIPPER e o sistema proposto, i.e., o GA-MISA-NC-ND-H.

Tabela 5.21: Parâmetros para o RIPPER

Parâmetros	Valor
ChecarTaxaErro	sim
numFolds	5
pesoMin	2.0
numOt	5
usarPoda	sim

Tabela 5.22: Resultados para o AG-MISA-NC-ND-H e RIPPER: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	RIPPER			GA-MISA-NC-ND-H			
Nome	**#Reg.**	**#Atr.**	**Pre.**	**#Reg.**	**#Atr.**	**Pre.**	**T-Teste**
bre	3,2	3,2	75.5	1	2,4	76,1	Não
cr-a	3,6	7,6	89.3	1,4	3,2	92,4	Sim
cr-g	6	16	71,4	2,2	6,6	74,4	Sim
gls	7,4	14,2	71,3	9	27,8	78,5	Sim
he-c	6	10,2	84.2	2,6	6,4	86,8	Sim
hep	7	27	84.4	1,8	3,8	88,4	Sim
h-col	5	10	84,3	1,8	5	84,8	Não
ino	6	10	91,2	2,4	19,8	91,3	Não
son	4	8	77,3	1,6	5	79,0	Não
vel	18	56	70.3	14	82,6	74,0	Sim
w-bca	11	11	93,1	2	4,2	96,1	Sim

Para as 11 bases de dados testadas, o sistema GA-MISA-NC-ND-H obteve resultados estatisticamente significantes em 7 delas. O sistema RIPPER chama atenção pela simplicidade das teorias desenvolvidas, entretanto, o sistema proposto desenvolve teorias mais simples que o RIPPER na grande maioria dos testes efetuados. A exceção a este caso, considerando o número de regras, se faz para a base de dados gls, em que o RIPPER usa em média 7,4 regras e o GA-MISA-NC-ND-H utiliza 9. Entretanto, para este caso, o resultado do sistema proposto é estaticamente significante quando comparado ao RIPPER. Estes resultados mostram de modo geral, que o sistema proposto é capaz de evoluir teorias mais simples e mais precisas do que as teorias desenvolvidas pelo algoritmo RIPPER.

A tabela 5.23 exibe o tempo gasto para ambos os sistemas para alcançar os resultados apresentados. Claramente, o tempo usado pelo RIPPER é bem menor.

Tabela 5.23: Resultados de Tempo para RIPPER e AG-MISA-NC-ND-H

	RIPPER	AG-MISA-NC-ND-H
Nome	Tempo (s)	Tempo (s)
bre	0,02	32,93
cr-a	0,14	97,21
cr-g	0,8	134,52
gls	0,09	123,87
he-c	0,03	44
hep	0,02	25,95
h-col	0,06	68,89
ino	0,09	117,64
son	0,02	46,89
vel	0,53	250,78
w-bca	0,2	81,11

As próximas tabelas exibem os resultados obtidos com o RIPPER comparados com as variantes do sistema proposto (AG-MISA-NC-ND-G e AG-MISA-NC-ND-L). Essa comparação tem o objetivo de verificar até que ponto a proposta de evolução hibrida de teorias influência os resultados do sistema.

Tabela 5.24: Resultados para o AG-MISA-NC-ND-G e RIPPER: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	RIPPER			GA-MISA-NC-ND-G			
Nome	#Reg.	#Atr.	Pre.	#Reg.	#Atr.	Pre.	T-Teste
bre	3,2	3,2	75.5	2,3	5,8	73,1	Não
cr-a	3,6	7,6	89.3	3,5	7,3	91,2	Não
cr-g	6	16	71,4	5,6	11,5	70,5	Não
gls	7,4	14,2	71,3	18	65,8	75,5	Sim
he-c	6	10,2	84.2	6,3	15,4	84,3	Não
hep	7	27	84.4	4,8	7,9	87,4	Sim
h-col	5	10	84,3	3,9	12,5	83,9	Não
ino	6	10	91,2	4,3	41,5	87,3	Não
son	4	8	77,3	3,2	9,5	78,4	Não
vel	18	56	70.3	25,9	170,5	73,5	Sim
w-bca	11	11	93,1	5	10,5	95,3	Não

Os resultados do GA-MISA-NC-ND-G são estatisticamente significativos em apenas 3 bases de dados testadas, para as outras 7 os resultados são equivalentes e para uma base de dados (ino), o resultado do RIPPER é estatisticamente superior quando comparado ao GA-MISA-NC-ND-G. Para os casos gls, he-c, vel, considerando o número de regras, as teorias desenvolvidas pelo GA-MISA-NC-ND-G são mais complexas quando comparadas às teorias desenvolvidas pelo RIPPER. Para 7 casos, as

teorias desenvolvidas pelo GA-MISA-NC-ND-G usam uma maior quantidade de atributos. Estes resultados, quando comparados aos resultados expostos na tabela 5.22, mostram a importância vital da abordagem hibrida para a evolução da teoria.

Tabela 5.25: Resultados para o AG-MISA-NC-ND-L e RIPPER: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	RIPPER			GA-MISA-NC-ND-L			
Nome	**#Reg.**	**#Atr.**	**Pre.**	**#Reg.**	**#Atr.**	**Pre.**	**T-Teste**
bre	3,2	3,2	75.5	3,8	12	74,5	Não
cr-a	3,6	7,6	89.3	6,1	15,8	92	Não
cr-g	6	16	71,4	5,3	20,5	71,5	Não
gls	7,4	14,2	71,3	35	128,8	74,5	Sim
he-c	6	10,2	84.2	9,5	28,5	83,3	Não
hep	7	27	84.4	7,2	16,5	85,4	Não
h-col	5	10	84,3	8,5	26,7	84,8	Não
ino	6	10	91,2	10,5	70,5	88,5	Não
son	4	8	77,3	6,4	20,8	79,0	Não
vel	18	56	70.3	60,5	333,5	73,5	Sim
w-bca	11	11	93,1	9	17,8	92,3	Não

Os resultados do GA-MISA-NC-ND-L são estatisticamente significativos em apenas 2 bases de dados testadas, para as outras 9 os resultados são equivalentes. Para todos os casos, exceto para cr-g, e w-ba, a complexidade das teorias do GA-MISA-NC-ND-L, considerando o número de regras, é maior quando comparada às teorias obtidas pelo RIPPER. O número de atributos usados pelas teorias do GA-MISA-NC-ND-L é bem maior do que a quantidade de atributos usada pelas teorias do RIPPER. A tabela 5.25 quando comparada à tabela 5.22, mostra uma vez mais, o quão importante é a abordagem de evolução hibrida de teorias.

5.6.3 Comparação com o GAssist

A tabela 5.26 exibe os parâmetros utilizados pelo GAssit, enquanto a tabela 5.27 exibe os resultados obtidos por esse sistema. Adicionalmente, a tabela 5.27 exibe os resultados obtidos com o GA-MISA-NC-ND-H.

Tabela 5.26: Média dos Parâmetros para o GAssist

Parâmetros	Valor
popSize	300
numIterations	750
initialNumberOfRules:	20
probCrossover:	0,6
probMutationInd:	0,6
probOne:	0,9
initMethod:	random
numStrata:	4
sizePenaltyMinRules:.	4
tournamentSize:	3
defaultClass:	majority
adiKR:	true
Discretizer [1..10]:	uniform-width
maxIntervals:	5
probMerge:	0,05
probReinitializeBegin	0,03
probReinitializeEnd	0
iterationHierarchicalSelection	24
hierarchicalSelectionThreshold	0
useMDL	true
iterationMDL	25
initialTheoryLengthRatio	0,075
weightRelaxFactor	0,90
iterationRuleDeletion	5
ruleDeletionMinRules	9

Tabela 5.27: Resultados para o GAssist: Nome = Nome da Base de Dados; #Reg. = Número de Médio Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	GAssist (I)			GA-MISA-NC-ND-H (II)			
Nome	#Reg.	#Atr.	Pre.	#Reg.	#Atr.	Pre.	T-Teste
bre	4,2	10	73,9	1	2,4	76,1	Não
cr-a	5,4	7,6	88,7	1,4	3,2	92,4	Sim - II
cr-g	4,6	9,8	71,9	2,2	6,6	74,4	Não
gls	7,6	15,2	73,1	9	27,8	78,5	Sim - II
he-c	4,8	9	83,5	2,6	6,4	86,8	Sim – II
hep	4,8	8,4	86,4	1,8	3,8	88,4	Não
h-col	4,8	6,6	85,4	1,8	5	84,8	Não
ino	4,6	14	95,0	2,4	19,8	91,3	Sim - I
son	7,2	21	79,9	1,6	5	79,0	Não
vel	21,5	172	74,3	14	82,6	74,0	Não
w-bca	4	9,2	94,1	2	4,2	96,1	Não

Note que nesta tabela, os sistemas foram numerados para uma melhor interpretação do T-Teste. Dessa forma, o sistema GAssit foi numerado como I e o sistema AG-MISA-NC-ND-H, como II. Os resultados do T-Teste devem ser interpretados da seguinte forma: caso a coluna exiba o valor 'não', a diferença entre as precisões não é estatisticamente significativa. Caso a coluna possua o valor 'sim', deve-se verificar o número que aponta qual dos sistemas obteve a precisão estatisticamente significativa. Para a base cr-a, por exemplo, o sistema AG-MISA-NC-ND-H alcança uma precisão estatisticamente significativa segundo o T-Teste. Já para o caso ino, o GAssit é o sistema que alcança a precisão estatisticamente significativa. Este mesmo raciocínio deve ser usado para analisar os resultados do T-Teste para as outras bases de dados.

Os resultados são bastante competitivos: para as 11 bases de dados testadas, em 7 delas os resultados de precisão são equivalentes; em 3 bases o sistema AG-MISA-NC-ND-H alcança precisões estatisticamente significativas e para 1 base de dados o GAssist é quem alcança a precisão significativa. Note, entretanto, que para todos os casos, exceto para o gls, a quantidade de regras que compõem a teoria desenvolvida pelo AG-MISA-NC-ND-H é menor quando comparada as teorias desenvolvidas pelo GAssist. Já para o número de atributos usados em cada teoria, para todos os casos, exceto para gls e ino, a quantidade de atributos utilizado pelas teorias desenvolvidas pelo AG-MISA-NC-ND-H é menor que a quantidade de atributos usados pelas teorias desenvolvidas pelo GAssist. Embora a complexidade da teoria desenvolvida para gls pelo AG-MISA-NC-ND-H seja maior, é importante notar que sua precisão é estatisticamente significativa quando comparada a teoria desenvolvida pelo GAssit para essa mesma base de dados. Assim, embora mais complexa, ela possui uma maior capacidade de classificação. Para o caso ino, o GAssit obtém uma teoria com melhor precisão do que o sistema AG-MISA-NC-ND-H.

Como afirmado, essa tabela demonstra que ambos os sistemas são competitivos, embora as teorias desenvolvidas pelo AG-MISA-NC-ND-H sejam mais simples quando comparadas as teorias desenvolvidas pelo GAssit.

A tabela 5.28 exibe o tempo gasto para ambos os sistemas para alcançar os resultados apresentados.

Tabela 5.28: Resultados de Tempo para GAssist e AG-MISA-NC-ND-H

	GAssist	AG-MISA-NC-ND-H
Nome	Tempo (s)	Tempo (s)
bre	88,5	32,93
cr-a	175,5	97,21
cr-g	155,5	134,52
gls	320,8	123,87
he-c	210,58	44
hep	100,78	25,95
h-col	205,34	68,89
ino	425,8	117,64
son	330,78	46,89
vel	360,57	250,78
w-bca	230,41	81,11

Note que o tempo de execução usado pelo GAssist é maior em todos os casos do que o tempo usado pelo sistema AG-MISA-NC-ND-H. Em alguns casos, como gls, he-c, hep, son, essa diferença é muito grande, chegando a ser três vezes maior (he-c e son). É importante notar que os problemas utilizados neste trabalho são relativamente simples de serem resolvidos e os sistemas apresentam, de forma geral, um baixo tempo de execução. Entretanto, ao se abordar problemas mais complexos, a diferença de tempo entre o GAssist e o sistema proposto pode se tornar um ponto crucial na escolha do sistema a ser utilizado.

5.6.4 Comparação com o XCS

A tabela 5.29 exibe os parâmetros utilizados pelo XCS, enquanto a tabela 5.30 exibe os resultados obtidos por esse sistema. Adicionalmente, a tabela 5.30 exibe os resultados obtidos com o GA-MISA-NC-ND-H.

Tabela 5.29: Média dos Parâmetros para o XCS

Parâmetros	Valor
numGenerations	10000
N	4000
β	0,2
α	0,1
e_0	0,001
v	5
γ	0,1
pmut	0,04
prec	0,8
δ	0,1
rp_I	10,0
ep_I	0,0
apt_I	0,01
pal	0,2
θsub	50,0
θdel	50,0
θAG	50,0
θmna	2
tournamentSize	0,4

Tabela 5.30: Resultados para o XCS: Nome = Nome da Base de Dados; #Reg. = Número Médio de Regras Utilizadas pela Teoria; #Atr. = Número Médio de Atributos Utilizados pela Teoria; Precisão = Precisão Média Obtida pela Teoria; T-Teste = Compara se os valores das precisões são estatisticamente significativos.

	XCS (I)			GA-MISA-NC-ND-H (II)			
Nome	**#Reg.**	**#Atr.**	**Pre.**	**#Reg.**	**#Atr.**	**Pre.**	**T-Teste**
bre	1220,2	6100,9	75,5	1	2,4	76,1	Não
cr-a	1981,4	17829	89,6	1,4	3,2	92,4	Não
cr-g	2638,2	31656,1	74,4	2,2	6,6	74,4	Não
gls	581,8	2905,3	73,2	9	27,8	78,5	Sim – II
he-c	1824,8	12768,7	82,8	2,6	6,4	86,8	Sim – II
hep	789,9	8679,9	83,9	1,8	3,8	88,4	Sim – II
h-col	1798,6	23374,5	82,1	1,8	5	84,8	Não
ino	1405	23885,4	87,4	2,4	19,8	91,3	Sim – II
son	1367	47845,3	80	1,6	5	79,0	Não
vel	2509	27599,2	73,1	14	82,6	74,0	Não
w-bca	1730	6920	95,7	2	4,2	96,1	Não

A coluna T-Teste da tabela 5.30 deve ser interpretada na mesma forma como na tabela 5.27. Note que os resultados são, mais uma vez, bem competitivos. Para 4 bases de dados, os resultados do AG-MISA-NC-ND-H são estatisticamente significativos em relação ao XCS quando a precisão é comparada. Para as outras 7 bases, os resultados

são equivalentes, sendo que o XCS não possui nenhum resultado que seja superior ao do sistema AG-MISA-NC-ND-H. Note, ainda, que todas as teorias evoluídas pelo AG-MISA-NC-ND-H são muito mais simples do que as teorias evoluídas pelo XCS. De fato, como explicado no capítulo 3, o XCS trabalha com uma população de classificadores em que todos são usados para classificação através de votação. Dessa forma, o número de classificadores é imenso, o que leva a teorias complexas que usam uma grande quantidade de atributos.

A tabela 5.31 exibe o tempo gasto para ambos os sistemas para alcançar os resultados apresentados.

Tabela 5.31: Resultados de Tempo para GAssist e AG-MISA-NC-ND-H

	XCS	AG-MISA-NC-ND-H
Nome	**Tempo (s)**	**Tempo (s)**
bre	57,84	32,93
cr-a	128,84	97,21
cr-g	100,07	134,52
gls	139,45	123,87
he-c	106,1	44
hep	48,21	25,95
h-col	187,59	68,89
ino	196,86	117,64
son	125,31	46,89
vel	300,9	250,78
w-bca	86,98	81,11

De modo geral, a quantidade de tempo utilizada pelo XCS é maior que o tempo gasto pelo sistema AG-MISA-NC-ND-H. Entretanto, existem bases de dados em que os tempos são bem próximos, como w-ba, por exemplo.

5.6.5 Comparação Geral

Nesta seção, é apresentada uma comparação geral de todos os sistemas usados neste trabalho. As complexidades das teorias desenvolvidas por cada sistema, bem como as precisões obtidas por estas teorias, são comparadas uma a uma para que se obtenha uma visão geral da capacidade do sistema proposto. Com esse objetivo, a tabela 5.30 exibe a complexidade das teorias enquanto que a tabela 5.32 exibe as precisões obtidas por essas teorias para cada base de dados e cada sistema testado.

Tabela 5.32: Comparação da Complexidade das Teorias Desenvolvidas

	Número de Regras					Número de Atributos				
Nome:	**C4.5**	**RI**	**GAsssit**	**XCS**	**AG**	**C4.5**	**RI**	**GAsssit**	**XCS**	**AG**
bre	7	3,2	4,2	1220,2	1	15	3,2	10	6100,9	2,4
cr-a	15	3,6	5,4	1981,4	1,4	72	7,6	7,6	17829	3,2
cr-g	60	6	4,6	2638,2	2,2	273	16	9,8	31656,1	6,6
gls	25	7,4	7,6	581,8	9	97	14,2	15,2	2905,3	27,8
he-c	26	6	4,8	1824,8	2,6	83	10,2	9	12768,7	6,4
hep	7	7	4,8	789,9	1,8	27	27	8,4	8679,9	3,8
h-col	13	5	4,8	1798,6	1,8	49	10	6,6	23374,5	5
ino	24	6	4,6	1405	2,4	71	10	14	23885,4	19,8
son	12	4	7,2	1367	1,6	53	8	21	47845,3	5
vel	140	18	21,5	2509	14	692	56	172	27599,2	82,6
w-bca	30	11	4	1730	2	57	11	9,2	6920	4,2

A tabela 5.32 mostra a complexidade das teorias desenvolvidas por todos os sistemas testados para todas as bases de dados usadas. Nesta tabela, o sistema AG-MISA-NC-ND-H, foi simplesmente representado por AG e o sistema RIPPER é representado por RI. A tabela 5.32 consta de duas colunas principais: uma que exibe a quantidade de regras de cada teoria para cada base de dados e a outra exibe a quantidade de atributos usados por essas teorias. As células destacadas em cinza indicam qual sistema possui o menor número de atributos e regras para cada base de dados.

Fica claro pela tabela 5.32, que em geral, o AG possui as teorias mais simples considerando tanto o número de regras quanto o número de atributos que constituem cada teoria. Para todas as bases de dados, exceto gls, o AG possui teorias com um menor número de regras que as teorias dos outros sistemas. Considerando o número de atributos usados nas teorias, o AG usa um número inferior de atributos para todas as bases de dados exceto para gls e gls, ino e vel, em que o sistema RIPPER usa um número menor de atributos do que o AG. Essa tabela, de maneira geral, permite organizar os sistemas na ordem crescente de complexidade de teorias, assim, as teorias mais simples são desenvolvidas pelo AG, seguido do RIPPER, GAssist, C4.5 e finalmente o XCS que possui as teorias mais complexas quando comparadas aos outros sistemas.

Considere agora a tabela 5.33 que compara as precisões obtidas para cada sistema em cada uma das bases de dados testadas.

Tabela 5.33: Resultados dos T-Testes para Todos os Sistemas Utilizados

	T-Teste para C4.5					T-Teste para RIPPER (RI)					T-Teste para GAssist					T-Teste para XCS					T-Teste para AG				
Nome:	RI	GAsssit	XCS	AG	C4.5	GAsssit	XCS	AG	C4.5	RI	C4.5	RI	XCS	AG	GAsssit	C4.5	RI	GAsssit	AG	XCS	C4.5	RI	GAsssit	XCS	AG
bre	75,5	73,9	75,5	76,1	74,7	73,9	75,5	76,1	74,7	75,5	74,7	75,5	75,5	76,1	73,9	74,7	75,5	73,9	76,1	75,5	74,7	75,5	73,9	75,5	76,1
cr-a	89,3	88,7	89,6	92,4	89,3	88,7	89,6	92,4	89,3	89,3	89,3	89,3	89,6	92,4	88,7	89,3	89,3	88,7	92,4	89,6	89.3	89.3	88,7	89.6	92,4
cr-g	71,4	71,9	74,4	74,4	73,4	71,9	74,4	74,4	73,4	71,4	73,4	71,4	74,4	74,4	71,9	73,4	71,4	71,9	74,4	74,4	73,4	71,4	71,9	74,4	74,4
gls	71,3	73,1	73,2	78,5	74,4	73,1	73,2	78,5	74,4	71,3	74,4	71,3	73,2	78,5	73,1	74,4	71,3	73,1	78,5	73,2	74.4	71,3	73,1	73,2	78,5
he-c	84,2	83,5	82,8	86,8	82,1	83,5	82,8	86,8	82,1	84,2	82,1	84,2	82,8	86,8	83,5	82,1	84,2	83,5	86,8	82,8	82.1	84.2	83,5	82,8	86,8
hep	84,4	86,4	83,9	88,4	84,4	86,4	83,9	88,4	84,4	84,4	84,4	84,4	83,9	88,4	86,4	84,4	84,4	86,4	88,4	83,9	84.4	84.4	86,4	83,9	88,4
h-col	84,3	85,4	82,1	84,8	83,7	85,4	82,1	84,8	83,7	84,3	83,7	84,3	82,1	84,8	85,4	83,7	84,3	85,4	84,8	82,1	83,7	84,3	85,4	82,1	84,8
ino	91,2	95	87,4	91,3	89,8	95	87,4	91,3	89,8	91,2	89.8	91,2	87,4	91,3	95	89,8	91,2	95	91,3	87,4	89.8	91,2	95	87,4	91,3
son	77,3	79,9	80	79	75,1	79,9	80	79	75,1	77,3	75.1	77,3	80	79	79,9	75.1	77,3	79,9	79	80	75.1	77,3	79,9	80	79
vel	70,3	74,3	73,1	74	71,3	74,3	73,1	74	71,3	70,3	71.3	70.3	73,1	74	74,3	71,3	70.3	74,3	74	73,1	71.3	70.3	74,3	73,1	74
w-bca	93,1	94,1	95,7	96,1	92,5	94,1	95,7	96,1	92,5	93,1	92,5	93,1	95,7	96,1	94,1	92.5	93,1	94,1	96,1	95,7	92.5	93,1	94,1	95,7	96,1

Tabela 5.34: Resultados de Tempo para Todos os Sistemas Utilizados

Nome:	RIPPER (s)	C4.5 (s)	GAssit (s)	XCS (s)	AG (s)
bre	0,38	0,02	88,5	57,84	32,93
cr-a	0,03	0,14	175,5	128,84	97,21
cr-g	0,05	0,8	155,5	100,07	134,52
gls	0,02	0,09	320,8	139,45	123,87
he-c	0,02	0,03	210,58	106,1	44
hep	0,01	0,02	100,78	48,21	25,95
h-col	0,02	0,06	205,34	187,59	68,89
ino	0,02	0,09	425,8	196,86	117,64
son	0,02	0,02	330,78	125,31	46,89
vel	0,08	0,53	360,57	300,9	250,78
w-bca	0,02	0,2	230,41	86,98	81,11

A tabela 5.33 compara os resultados dos T-Testes realizados para todos os sistemas testados para cada base de dados usada. Esta tabela consta de cinco colunas principais, sendo cada uma relativa a cada um dos cinco sistemas avaliados. O AG-MISA-NC-ND-H foi simplesmente representado por AG nesta tabela e o RIPPER por RI. Cada coluna principal compara os resultados das precisões obtidas pelo sistema que dá nome a coluna com os sistemas representados nas colunas que constituem a coluna principal. Assim, a coluna T-Teste para C4.5 compara os resultados deste algoritmo com os sistemas RIPPER, GAssit, XCS e GA. Dada uma base de dados, o resultado da precisão para o sistema exposto na coluna principal é estatisticamente significativo do que o resultado obtido por outro sistema que compõem a coluna principal, se o valor da célula para o sistema que compõem a coluna principal estiver em tom de branco. Para todas as células em tom de cinza, o resultado da precisão não é estatisticamente significante. Dessa forma, para o sistema GAssist, (veja coluna T-Teste para GAssist), pode-se dizer que para a base de dados hep, a precisão obtida pelo GAssist é estatisticamente significativa do que as precisões obtidas pelo XCS e (célula em tom branco), mas a precisão do GAssist não é estatisticamente significativa do que a precisão obtida pelo o sistema C4.5, RIPPER e AG, já que essas células se apresentam em tom de cinza. Esse mesmo raciocínio deve ser seguido para todos os outros testes representados nesta tabela.

Note que cada sistema é comprado a quatro outros em cada uma das onze bases de dados. Assim, no total, para cada sistema, 44 comparações são efetuadas.

Para os T-Testes para o C4.5 (coluna T-Teste para C4.5), verifica-se que para nenhuma base de dados, as precisões obtidas por esse sistema são estatisticamente significativas quando comparadas a precisões das teorias obtidas por outros sistemas.

Para os T-Testes para o RIPPER (coluna T-Teste para RIPPER), verifica-se que para nenhuma base de dados, as precisões obtidas por esse sistema são estatisticamente significativas quando comparadas a precisões das teorias obtidas por outros sistemas.

Para os T-Testes efetuados para o sistema GAssist (coluna T-Teste para GAssist), verifica-se que as precisões obtidas por esse sistema são estatisticamente significativas em 8 casos. Em 3 deles, a precisão é significativa em relação ao C.45, em 2 deles, a precisão é significativa em relação ao RIPPER, em 2 casos a precisão é significativa em relação ao XCS e em 1 caso a precisão é significativa em relação ao AG. Para todos os outros testes, as precisões são equivalentes. Dessa forma, verifica-se

que em 18,1% dos casos testados, o GAssist possui uma precisão estatisticamente significativa.

Para o sistema XCS, os T-Testes efetuados indicam que a precisão deste sistema é estatisticamente significativa em 5 casos, sendo 2 deles relativos aos resultados do C4.5, 2 casos relativos ao RIPPER, e 1 caso relativo ao sistema GAssist e nenhum relativo ao AG. Dessa forma, nota-se que em 11,4% dos testes efetuados, a precisão do XCS é estatisticamente significativa.

Para o AG, os T-Testes realizados demonstram que a precisão deste sistema é estatisticamente significativa em 21 casos, sendo 7 deles relativos aos resultados do C4.5, 7 relativos ao RIPPER, 3 relativos ao GAssist e 4 relativos ao XCS. Para todos os outros casos as precisões obtidas são equivalentes. Assim, verifica-se que em 47,8% dos casos testados, o GA possui uma precisão estatisticamente significativa quando comparadas aos sistemas C4.5, RIPPER, XCS e GAssist.

A tabela 5.34 exibe o tempo (em segundos) utilizado por cada sem cada base de dados. Note que, considerando apenas os sistemas genéticos, o sistema proposto neste trabalho possui o menor tempo de execução.

As tabelas 5.32, 5.33 e 5.34 mostram não somente que o sistema proposto se apresenta bastante competitivo quando comparado aos outros sistemas de aprendizado de regras testados, mas que ele é também capaz de desenvolver teorias bem mais simples quando comparadas às outras teorias desenvolvidas. Dessa forma, o sistema proposto obtém teorias com alta precisão e que possuem um bom nível de interpretação das regras que a compõem.

6. Conclusão e Trabalhos Futuros

Neste trabalho, vários tipos de contribuições foram apresentas e independentemente testadas e avaliadas. Posteriormente, o sistema de aprendizado genético como um todo foi também avaliado em comparação a outros sistemas de aprendizado de regras. Primeiramente, este capítulo apresenta as conclusões relativas a cada uma das quatro contribuições deste trabalho. Em seguida, algumas conclusões gerais sobre o sistema proposto são apresentadas. Finalmente, apresentam-se algumas diretrizes de trabalhos futuros.

6.1 Conclusões sobre o Mecanismo Implícito de Seleção de Atributos (MISA)

Em primeiro lugar, foi demonstrado o que o MISA, em comparação a abordagem padrão de seleção de atributos que utiliza variáveis boolenas, apresenta uma enorme economia de memória. Em seguida, o MISA foi independentemente avaliado mostrou-se de fundamental importância para a obtenção de teorias que usam uma quantidade reduzida de atributos e por isso são mais simples de serem interpretadas.

6.2 Conclusões sobre a Abordagem Hibrida para Evolução de Teorias

A abordagem hibrida para a evolução de teorias mostrou-se de essencial importância para a obtenção de teorias com alta precisão e que utilizem um número reduzido de regras e de atributos. Adicionalmente, a abordagem proposta alcança, em geral, melhores resultados em menos tempo de execução quando comparado a outras abordagens para a evolução de teorias.

6.3 Conclusões sobre o Novo Operador de Recombinação Natural Contínuo (NORNC)

O NRNC apresenta uma maior capacidade de exploração da matriz de discretização quando comparado ao operador natural de recombinação contínuo proposto (ORNC) por Aguilar-Ruiz et al. Como conseqüência, o operador proposto necessita de menos tempo de execução para obter os mesmos resultados obtidos pelo ORNC.

6.4 Conclusões sobre o Novo Operador de Recombinação Natural Discreto (NORND)

Foi demonstrado que o NORND é capaz de explorar o espaço de busca como o operador de recombinação binário de dois pontos. Os resultados permitem concluir que o NORND é mais eficiente tanto no sentido de obtenção de teorias com um menor número de regras e atributos, quanto no quesito de obtenção de teorias com uma maior precisão e, ainda, no sentido de economia de tempo de execução quando comparado ao operador de recombinação natural discreto (ORND) proposto por Aguilar-Ruiz et al.

6.5 Conclusões sobre o Sistema

Os resultados obtidos pelo sistema proposto foram competitivos quando comparados a outros algoritmos para geração de teorias. De modo geral, pode-se dizer que as teorias desenvolvidas pelo sistema proposto são mais simples, i.e., utilizam um menor número de atributos e de regras quando comparado às teorias obtidas pela utilização dos outros sistemas. Os resultados demonstram ainda, que as quatro contribuições propostas pelo trabalho apresentam vantagens quando usadas isoladamente, e, que, quando usadas em conjunto, são capazes de gerar resultados ainda melhores.

6.6 Trabalhos Futuros

As prioridades para o aprimoramento e desenvolvimento do sistema proposto incluem os seguintes tópicos:

- **Utilização do sistema em problemas mais complexos:** Dessa forma, pretende-se avaliar o comportamento do sistema em problemas mais complexos que apresentem de alta dimensionalidade e uma pequena quantidade de instâncias.
- **Inclusão de Módulo de Busca Local Estocástica:** Este módulo teria o objetivo de otimizar de forma local alguns indivíduos da população, combinando, dessa forma, a capacidade de exploração global dos AGs com a força de exploração local das buscas locais (Hoos e Stützle, 2004). Resultados preliminares já mostram que a inclusão do método de busca local estocástica é capaz de evoluir teorias ainda mais simples e mais precisas.

Apêndice A: Validação Cruzada

O processo de validação cruzada (Mitchell, 1997) é utilizado com o objetivo de determinar os parâmetros do sistema de aprendizado, bem como de maneira a avaliar de forma mais eficaz uma teoria obtida. De maneira geral, o processo de validação cruzada divide o conjunto de dados em k partições, definindo desta forma k conjuntos de treinamento e k conjuntos de teste, i.e., um conjunto de teste para cada conjunto de treinamento. Os k conjuntos de treinamento são então particionados novamente em n partes de maneira a formar n conjuntos de treinamento e n conjuntos de validação. Assim, para cada conjunto k de tese, existem n conjuntos de treinamento e n conjuntos de validação. A partir deste ponto, o processo de aprendizado é dividido em duas etapas, a saber: a etapa de treinamento, em que a teoria é desenvolvida, e a etapa de teste, em que a teoria desenvolvida é avaliada.

A etapa de treinamento, como anteriormente explicado, é a responsável pela construção da teoria. Contudo, antes que esta fase possa ser realizada, é necessária a determinação dos parâmetros do sistema. Os n conjuntos de treinamento e validação são utilizados para se determinar os parâmetros do sistema para cada subdivisão k do conjunto de dados. Assim, para cada conjunto n de treinamento, executa-se o sistema com uma configuração arbitrária de parâmetros e analisa-se a precisão da teoria desenvolvida em cada conjunto n de validação. Caso a precisão da teoria seja satisfatória no conjunto de validação, os parâmetros utilizados são armazenados. Caso a precisão seja insatisfatória, os parâmetros do sistema são ajustados com objetivo de fazer com que a precisão da teoria se torne satisfatória para cada conjunto de validação. É importante notar que existem, para cada k, n conjuntos de treinamento e n conjuntos de validação. Assim, para cada n, o procedimento acima descrito é utilizado, ou seja, são determinadas n listas de parâmetros do sistema para cada k. Uma vez que a lista de n parâmetros seja determinada, calcula-se a média dos parâmetros sobre estes n conjuntos. Esse valor médio dos parâmetros é então utilizado na fase de treinamento. Assim, para cada conjunto de treinamento k, a determinação dos parâmetros é realizada sobre os n subconjuntos de treinamento e n conjuntos de validação. Como afirmado, a média dos parâmetros é utilizada na etapa de treinamento.

A etapa de teste avalia a precisão do modelo nos conjuntos de teste. Uma vez que a etapa de treinamento sobre um conjunto k tenha sido realizada, a teoria desenvolvida é avaliada sobre o conjunto de teste k. De forma geral, o resultado do

sistema para cada base de dados é a média das precisões obtidas sobre os k conjunto de tese.

Referências Bibliográficas

AGUILAR-RUIZ, J. S. GIRALDEZ, R. RIQUELME, J. C., "*Natural Encoding for Evolutionary Supervised Learning*", Evolutionary Computation, Vol. 11, Issue: 4 Aug. 2007, pp. 466-479

AGUILAR-RUIZ J. S., RIQUELME J. C. "*Improving the Evolutionary Coding for Machine Learning Tasks*", European Conference on Artificial Intelligence, IOS Press, Lyon, France 2002, pp 173-177, 2002.

AGUILAR–RUIZ J. S., RIQUELME J. C., TORO M. "*Evolutionary Learning of Hierarchical Decision Rules*", IEEE Transactions on Systems, Man and Cybernetics, Part B, 33(2), pp. 324–331, 2003.

AMABIS E MARTHO, *Biologia das Populações: Genética, Evolução e Ecologia*, São Paulo, Editora Moderna, 1997.

AHA, D., E D. KIBLER, "Instance-based learning algorithms", Machine Learning, vol.6, pp. 37-66, 1991.

BACARDIT, J., "*Creació d'um assitent digital per la preccció del francàs acadèmic basat en conputació evolutiva*", Porjecte Final de Carrera, Enginyeria i Arquitectura La Salle, Universitar Ramom Llull, 2000

BACARDIT, J. E BUTZ, M.V., "*Data Mining in Learning Classifier Systems: Comparing XCS with GAssist*", 7th Inter. Workshop on Learning Classifier Systems, IWLCS 2004, Seattle, USA, 2002.

BACARDIT J., GARRELL J. M.. "*Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system*", Sixth International Workshop on Learning Classifier Systems (IWLCS-2003), Chicago, July 2003.

BACK, T. *Evolutionary algorithms in theory and practice, USA,* Oxford University Press, 1995.

BAKER J. E. "*Adaptive Selection Methods for Genetic Algorithms*", First International Conference on Genetic Algorithms and their Applications, San Mateo: Morgan Kaufmann, pp. 101-111, 1985.

BESSAOU M., Siarry P. "*A genetic algorithm with real-value coding to optimize multimodal continuous functions*", Struct Multidisc Optim pp. 63-74, 2001.

BLAKE, C.L. AND MERZ, C.J., "*UCI Repository of machine learning databases*", Irvine, CA: University of California, Department of Information and Computer Science, 1998, http://www.ics.uci.edu/#mlearn/MLRepository.html

BOSER B., GUYON I., VAPNIK V. *"A training algorithm for optimal margin classifiers"*, In Fifth Annual Workshop on Computational Learning Theory, pages 144--152, Pittsburgh, ACM. 1992

BRINDLE A. *Genetic Algorithms for Function Optimization*, PhD thesis, University of Alberta, Edmonton, Canada, 1981.

CANO J. R., HERRERA F, LOZANO M., *"Evolutionary Stratified Instance Selection applied to Training Set Selection for Extracting High Precise-Interpretable Classification Rules"*, IEEE ICDM 2004 Workshop on Alternative Techniques for Data, 2004.

CHIH-HSUN C.; JOU-NAN C. *"Genetic algorithms: initialization schemes and genes extraction"*, The Ninth IEEE International Conference on Fuzzy Systems, FUZZ IEEE. Volume 2, Page(s):965 - 968 vol.2, 2000

COHEN W. *"Efficient pruning methods for separate-and-conquer rule learning systems"*, in IJCAI 1993: 988-994, 1993.

COHEN W. *"Fast Effective Rule Induction"*, Twelfth International Conference on Machine Learning, 115-123, 1995.

DE JONG, K.. *"Learning with genetic algorithms: An overview"*, Machine. Learning., 3(2-3), 121–138., 1988.

DEJONG, K.A., SPEARS W.M. *"An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms"*, First Workshop Parallel Problem Solving from Nature, Springer-Verlag, pp. 38-47, Berlin, 1990.

DEJONG, K. A. SPEARS, W. M. *"Learning Concept Classification Rules Using Genetic Algorithms"*, *12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann., Sydney, Australia, 1991., pp. 51-56, 1991.

DEJONG K. A, SPEARS W. M, GORDON D. F, *"Using genetic algorithms for concept learning"*, Machine. Learning, vol. 1, no. 13, pp. 161-188, 1993.

EIBEN A. E., SCHIPPERS A., *"On Evolutionary Exploration and Exploitation"*, Fundamenta Informaticae, IOS Press, Amsterdam, The Netherlands, Vol. 35, Issue 1-4, pp. 35-50, 1998.

ENDOU T., ZHAO Q. *"Generation of comprehensible decision trees through evolution of training data"*, Proceedings of the Congress on Evolutionary Computation, pp. 1221-1225, 2002.

ESHELMAN L. J., CARUANA R. A., SCHAFFER J. D. *"Biases in the crossover landscape"*, International Conference on Genetic algorithms, Morgan Kaufmann Publishers Inc , pp 10-19, San Francisco, CA, USA, 1989.

FAYYAD U.M, IRANI K.B. "Multi-interval discretization of continuous valued attributes for classification learning", Proceedings of the 13th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Chambery, France,, pp. 1022-1027, 1993.

FOGEL, D. B. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence* IEEE Press, New York, 2000.
FÜRNKRANZ, J. "*Separate-and-conquer rule learning*", Artificial Intelligence Review, vol. 13, pp. 3–54, 1999.

FÜRNKRANZ, J., WIDME, F., "*Incremental reduced error pruning*", Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann. pp. 70-77, 1994.

GHOSH A, TSUTSUI S., TANAKA H. E CORNE D, "*Genetic Algorithms with Substitution and Re-entry of Individuals*", International Journal of Knowledge-Based Intelligent Engineering Systems, Vol. 4, No. 1, pp. 64-71, 2000.

GIRÁLDEZ R., AGUILAR-RUIZ J. S. , RIQUELME J. C., "Natural Coding: A More Efficient Representation for Evolutionary Learning". GECCO 2003, String Verlag Berlin Heidelberg, Chicago, USA, pp. 279 - 290, 2002.

GOLDBERG, D. E., *Genetic Algorithms in Search, Optimization & Machine Learning. Reading,* Addison Wesley, 1989.

GOLDBERG, D. E., KARGUPTA, H., HORN, J., CANTU-PAZ, E. "Critical *deme size for serial and parallel genetic algorithms*", IlliGAL Report 95002 (The Illinois GA Lab, University of Illinois). 24 pp, 1995.

GUYON I., ELISSEEFF A., "*An Introduction to Variable and Feature Selection*" Journal of Machine Learning Research, 157-1182, 2003.

HOLLAND, J. H. *Adaptation in natural and artificial systems* MIT Press, Cambridge, 1975.

HOLLAND, J. H., & REITMAN, J. S. "*Cognitive systems based on adaptive algorithms*", Hayes-Roth, D., & Waterman, F. (Eds.), Pattern-directed Inference Systems pp. 313–329, New York Academic Press 1978.

HOOS H., STÜTZLE T., *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann, San Francisco, CA, USA 2004

KAELBLING L, LITTMAN M, MOORE A. "*Reinforcement Learning: A Survey*", Journal of Artificial Intelligence "Research", vol 4, pp 237-285, 1996.

KOHAVI R., JOHN G., "*Wrappers for Feature Subset Selection*", Artificial Intelligence journal, special issue on relevance, Vol. 97, Nos 1-2, pp. 273-324, 1997.

MCCULLOCH, W., & PITTS, W. "*A logical calculus of ideas immanent in neural activity*", Bulletin of Mathematical Biophysics, 5, 115–133, 1943.

MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs.* Springer-Verlag, 1996.

MICHALSKI, R. "*A theory and methodology of inductive learning*" In Machine Learning: An Artificial Intelligence Approach. Palo Alto: Tioga. 1983

MITCHELL, T. *Machine Learning*, McGraw Hill, 1997.

NADEAU C., BENGIO Y., "*Inference for the Generalization Error*", Machine Learning 52(3) pp. 239-281, 2003.

OCHOA G., HARVEY I., BUXTON H. "*Optimal Mutation Rates and Selection Pressure in Genetic Algorithms*", Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann, pp. 93-101 San Francisco, CA, 2000.

PEARL, J. *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufmann Publishers, 1988.

PLATT J. "*Using Analytic QP and Sparseness to Speed Training of Support Vector Machines*", Advances in Neural Information Processing Systems 11, pp. 557-563, 1999.

PUNCH W. F., GOODMAN E. D., MIN PEI, "*Further Research on Feature Selection and Classification Using Genetic Algorithms*", Computer Science Dep., Michigan State University, USA, pp 557-564, 1998.

QUINLAN R. "*Induction of decision trees*", Machine Learning. 1(1):81-106, 1986.

QUINLAN R. "*C4.5: Programs for Machine Learning*", Morgan Kaufmann Publishers, San Mateo, CA, 1993

QUINLAN R. CAMERON-JONES M."*{FOIL}: {A} Midterm Report*", Machine Learning: {ECML}-93, European Conference on Machine Learning, Proceedings, vol. 667, Springer-Verlag, pp. 3-20, 1993.

RISSANEM, J. "*Modeling by Shortest data Description*" Automatica, vol. 14, pp. 465-471, 1978

RIVEST R. L. "Learning decision lists", Machine Learning , vol.2 , pp. 229-246, 1987.

RUSSEL S., NORVIG P., *Artificial Intelligence: A Modern Approach*, 2nd Edition, Prentice Hall, 1995.

SMITH, S. F., *A learning system based on genetic algorithms*, Doctoral dissertation, University of Pittsburgh, 1980

SMITH, S. F. "*Flexible learning of problem solving heuristics through adaptive search*", Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 421–425, 1983.

SYSWERDA G. *"Uniform Crossover in Genetic Algorithms"*, Third International Conference on Genetic Algorithms and their Applications, San Mateo, Morgan Kaufmann, pp. pages 2-9, 1989.

VENTURINI, G. *"Sia: A supervised inductive algorithm with genetic search for learning attributes based concepts"* ECML-93 - Proc. of the European Conference on Machine Learning, Heidelberg: Springer-Verlag, .pp. 280–296, Berlin, 1993.

VOSE M. D., WRIGHT A. H., ROWE J. E., Proceedings of GECCO 2003, vols. 2723-2724), Springer-Verlag, 2003.

WILSON, S.W.: *"Classifier fitness based on accuracy."* Evolutionary Computation, vol. 3, pp. 149-175, 1995.

WITTEN I. H., FRANK E. *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

YANG J., *"Feature Subset Selection Using A Genetic Algorithm"*, Vasant Honavar, IEEE Intelligent Systems, pp 44-49, 1998.