Aprendizado Genético: Operadores de Crossover Naturais Aprimorados

Cristiano Pitangui[*], Gerson Zaverucha[*]

Universidade Federal do Rio de Janeiro (UFRJ)/COPPE / Rio de Janeiro – RJ – Brasil

{gerson, cris_pi}@cos.ufrj.br

Abstract. *Aguilar-Ruiz et al proposed crossover operators, both discrete and continuous, for the natural representation. They showed advantages in accuracy and also in efficiency compared to the binary ones. However, they do not explore the search space like the two points crossover when the binary coding is being used. In order to do so, in our previous work we proposed a new natural discrete crossover operator which gave very good results compared to C4.5 in several UCI databases. Nonetheless, it was not experimentally compared to Aguilar-Ruiz et al natural operator. So, in this work, we perform this comparison in the same datasets and define a new natural continuous crossover operator, which is also evaluated. Results shows that our operators achieves better accuracy and simpler concepts using less time*

Resumo. *Aguilar-Ruiz et al propuseram operadores de crossover para dados contínuos e discretos utilizado a representação natural. Eles apresentaram vantagens na precisão e na eficiência em comparação com a representação binária. Entretanto, estes operadores não exploram o espaço de busca como o operador binário de dois pontos. A fim de obter essa exploração, nosso trabalho anterior propôs um novo operador natural discreto que resultou em bons resultados quando comparado com o C4.5 em dados do UCI. Contudo, esse operador não foi comparado com o operador natural de Aguilar-Ruiz et al. Assim, neste trabalho, nós apresentamos essa comparação e definimos um novo operador natural contínuo o qual é também avaliado. Resultados mostram que nossos operadores obtêm melhor precisão e conceitos mais simples usando menos tempo.*

1. Introdução

Aprendizado Genético é o nome dado à aplicação de Algoritmos Genéticos (AG) [Goldberg 1989] a problemas de aprendizado. Um tópico importante sobre Aprendizado Genético se relaciona à codificação usada para representar a regra. Tradicionalmente, a representação binária [DeJong, Spears e Gordon 1993] é utilizada; entretanto, Aguilar-Ruiz et al propuseram a Codificação Natural [Aguilar-Ruiz, Riquelme e Carmelo 2002] que apresenta uma maneira mais compacta para se codificar o problema. Os operadores de *crossover* naturais de Aguilar-Ruiz et al obtiveram conceitos com precisão mais elevada, menos regras, usando um número menor de gerações e tamanho de população quando comparados à representação binária dos atributos. Entretanto, eles não exploram o espaço de busca como o operador de *crossover* binário de dois pontos. Com este objetivo, em [Pitangui e Zaverucha, 2006] apresentou-se um operador de *crossover* natural discreto que obteve bons resultados quando comparados ao C4.5. Entretanto, este operador não foi experimentalmente comparado ao operador natural de Aguilar-Ruiz et al. Neste trabalho, executa-se esta comparação e define-se um novo *crossover* natural contínuo, que é comparado ao operador de Aguilar-Ruiz et al.

O artigo é estruturado como segue. A seção 2 resume o sistema proposto em [Pitangui e Zaverucha, 2006]. A seção 3 descreve o operador de *crossover* natural discreto de Aguilar-Ruiz et al e expõe as razões que levaram a desenvolver nosso operador de *crossover* para dados discretos. Similarmente, a seção 4 descreve o operador natural contínuo de Aguilar-Ruiz et al e explica as razões levaram a

[*] O primeiro autor é financiado pela CAPES e o Segundo pelo CNPq.

desenvolver um novo operador natural para dados contínuos, apresentado na seção 5. Os resultados experimentais são mostrados na seção 6. Finalmente, a seção 7 conclui o trabalho e apresenta alguns trabalhos futuros.

2. O Sistema: Uma Rápida Revisão

2.1 Codificação

Na codificação binária [DeJong, Spears e Gordon 1993], um bit é usado para cada valor de cada atributo discreto. Para atributos contínuos, uma das abordagens é representar os limites inferiores e superiores do atributo [Bacardit e Butz 2004].

Exemplo 1: Considere a classificação de um cão como bonito ou feio. O cão é descrito por três características: pêlo (curto, médio, ou longo); tamanho (pequeno, médio, ou grande) e peso, que varia entre 10 kg e 50 kg. Assim, a seguinte regra tem a representação binária:

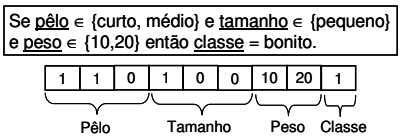

Figura 1. Representação Binária.

O sistema usa a Codificação Natural [Aguilar-Ruiz Riquelme e Carmelo 2002], que reduz a quantidade memória gasta para alocar a regra. A codificação usa um número natural para representar cada atributo; os atributos contínuos devem ser discretizados.

Exemplo 2: A regra da figura 1, supondo que a variável peso foi discretizada e o intervalo [10, 20] foi codificado como o número natural 7, seria representada com a codificação natural como:

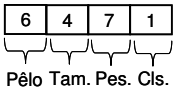

Figura 2. Representação Natural.

2.2 Integrando Pittsburgh e Michigan

Nossa abordagem adiciona novas regras somente após regras previamente adicionadas terem sido amplamente evoluídas. Isto significa que o sistema, antes de adicionar uma nova regra, tenta explorar o potencial máximo de uma regra previamente adicionada. A figura 3 apresenta o algoritmo bem como uma simples explicação do mesmo.

Entrada: **maxn**: numero máximo de gerações; **k**: número que determina quando a sub-geração deve ocorrer; **pp**: população principal; **sp**: população da sub-geração; **e+**: conjunto de exemplos positivos na base de dados; **e-**: conjunto de exemplos negativos na base de dados; **enc+**: conjunto de exemplos positivos não cobertos pelo melhor indivíduo em pp;
Saída: População principal (**pp**), onde cada um de seus indivíduos possui um número de regras determinado pelo valor de k;
Inicio:

```
0 - Inicialize pp aleatoriamente;          //Criação da população principal de forma aleatória.
1 - Enquanto (geração < maxn)              //Teste para finalização da execução do algoritmo.
2 - Evolua pp em e+ ∪ e-;                  //Evolução global. Evolução em todo o conjunto de exemplos
3 - Se ( (geração % k) == 0 ) então        //Valor de k determina quando a sub-geração deve ocorrer.
        4 – Inicialize sp aleatoriamente;  //População que fará a sub-geração é criada de forma aleatória.
        5 - Evolua sp em enc+ ∪ e-;        //Evolução local. sp é evoluída em todos os exemplos negativos e
                                           //nos exemplos positivos que o melhor indivíduo em pp não cobre.
        6 - pp ← pp concatenada com sp;    //Concatenação das duas populações.
    fim-se;
7- geração ← geração + 1;                  //Incrementa o número de gerações.
fim-enquanto;
retorne pp;                                //Retorno da população principal evoluída.
Fim;
```

Figura 3. Integração de Michigan e Pittsburgh

Este processo é simples e reduz o número das regras em um indivíduo explorando o potencial de classificação para cada regra adicionada (passo 5). O procedimento usado é semelhante ao processo usado por HIDER [Aguilar–Ruiz, Riquelme e Toro 2003]. Entretanto, em HIDER as regras adicionadas e a regra previamente adicionada não são evoluídas globalmente (passo 2), como em nossa abordagem.

2.3 O Mecanismo Implícito de Seleção de Características (MISC)

Nosso mecanismo de seleção é codificado diretamente na representação da regra, i.e., não existe a necessidade de variáveis externas.

2.3.1 MISC para Dados Discretos

Considere um atributo N e que |N| é a cardinalidade de seu domínio. Para um atributo discreto N, um bit, o de posição |N| + 1, será usado como o mecanismo da seleção de características. Se o valor deste bit for 1, a característica é usada; caso contrário, a mesma não é usada. Para a característica pêlo, do exemplo 1, |N| = 3, se o número natural que codifica o atributo for maior de 8 (2^3), a característica é usada, caso contrário, ela não é usada. O mecanismo é muito econômico e simples de ser testado.

2.3.2 MISC para Dados Contínuos

Para dados contínuos, o mecanismo é baseado no número de intervalos retornados por algum algoritmo de discretização. Considere um atributo contínuo N. Para θ que representa o número de intervalos, existe um conjunto $\mu = \{n_1, n_2.., n_\theta\}$ de números naturais que representam os intervalos obtidos. Assuma que n_θ é o número natural atribuído ao último intervalo. Assim, o número representado por $\alpha = n_\theta + 1$ constitui o número que verifica se o atributo será ou não utilizado. Assim, para um dado n_i ($1 \leq i \leq \theta$), sua seleção é dada pelo número $n_i + \alpha$. Portanto, se n_i é maior que α, a característica, que ele representa, é usada. Caso contrário, a característica que ele representa não é usada. O mecanismo é muito econômico e bem simples de ser testado.

3. Operadores de *Crossover* Naturais para Dados Discretos

Esta seção descreve as razões que levaram ao desenvolvimento do novo *crossover* natural discreto [Pitangui e Zaverucha, 2006]. Antes de rever este operador, uma breve análise do *crossover* natural discreto de Aguilar-Ruiz et al é realizada.

3.1 O *Crossover* Natural Discreto de Aguilar-Ruiz et al

Este operador é baseado em mutações. A mutação para atributos discretos trabalha de maneira idêntica à mutação binária. A mutação do k-ésimo bit de um numero natural n é dada por:

$$mut_k(n) = (n + 2^{k-1})\%2^k + 2^k \left\lfloor \frac{n}{2^k} \right\rfloor \quad (1)$$

Exemplo 3: Uso da equação (1) para n = 5 (101):

$$mut_1(5) = (5 + 2^0)\%2^1 + 2^1 \left\lfloor \frac{5}{2^1} \right\rfloor = 4 \rightarrow (100)$$

O *crossover* natural discreto de Aguilar-Ruiz et al é baseado em duas definições, a saber: Conjunto de Mutação e Classe de Mutação de Ordem-j.

O conjunto de mutação, definido como Mut(n), contém todos os números gerados a partir da mutação de cada um dos bits de n. Assim, para o número 1 codificado com 3 bits, i.e., 001, o conjunto de Mutação é dado por: Mut(1) = {0, 3, 5}.

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 4: Considere n = {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}; Cl-Mut(1)^0 = {1} (Por definição); Cl-Mut(1)^1 = Mut(1) $\cup$ {1} = {1, 0, 3, 5}; Cl-Mut(1)^2 = Mut(1) $\cup$ Mut(0) $\cup$ Mut(3) $\cup$ Mut(5) $\cup$ {1}, assim: Cl-Mut(1)^2 = {1, 0, 3, 5, 2, 4, 7};

O *crossover* natural de Aguilar-Ruiz et al [Aguilar-Ruiz, Riquelme e Carmelo 2002] 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.

Este operador envolve no mínimo três operações: cálculo da classe de mutação, cálculo da interseção e sorteio da prole. O número de bits que representa um número influencia a complexidade deste operador. A tarefa de encontrar a interseção entre conjuntos, torna-o ainda mais lento. Este operador não explora o espaço de busca como o *crossover* binário de dois pontos. Para ilustrar o fato, considere o próximo exemplo:

Exemplo 5: Deseja-se realizar o *crossover* nos números n_1 = 29 (11101) e n_2 = 6 (00110). A primeira interseção não vazia entre as classes de mutação dos números n_1 e n_2, é encontrada para a Classe de Mutação de Ordem 2. Portanto, tem-se:
Cl-Mut(29)^2 $\cap$ Cl-Mut(6)^2 = {30, 20, 12, 15, 5, 23}. Assim, todos os possíveis pares de filhos para serem escolhidos aleatoriamente são: {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}. Usando o *crossover* binário de dois pontos, e os números 29 e 6 como pais, têm-se os seguintes pares de filhos possíveis: {28-7, 30-5, 22-3, 31-4, 29-6, 21-14, 23-12, 13-22}.

A quantidade de pares de filhos criados pelo operador natural é maior do que a criada com o *crossover* binário. Entretanto, o *crossover* de Aguilar-Ruiz et al cobre apenas um dos casos (30-5) do operador binário. Isso ocorre porque o operador de Aguilar-Ruiz et al é baseado na mutação e não respeita a troca de material genético. Embora existam mais filhos gerados por este *crossover*, a sua busca, além de mais lenta, não se apresenta como um operador de troca genética entre dois indivíduos, e sim uma mutação entre ambos, inserindo, portanto, novo material genético na população.

3.2 Nosso Operador de *Crossover* Natural Discreto

Nosso operador de *crossover* natural discreto é baseado em operadores *bit-wise* presentes na maioria das linguagens de programação. Os operadores usados são os de deslocamento de bit para direita e esquerda, representados respectivamente por >> e <<. Como exemplo, considere o número 6 representado em uma variável de 4 bits. O número 6 será deslocado em dois bits para direita e, posteriormente, em um bit para esquerda. Assim: 6 (0110) >> 2 = 1 (0001), 6 (0110) << 1 = 12 (1100)

Exemplo 6: Considere o *crossover* binário de dois pontos entre os pais P1 = 6 e P2 = 29. Assuma que os pontos de corte foram 2 e 5. Assim, o *crossover* seria dado por:

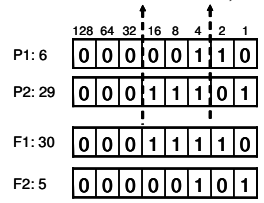

Figura 4. Crossover Binário de Dois Pontos.

Observe que a parte do P1 que será trocada é equivalente ao número natural 4, e que a parte de P2 que será trocada é equivalente ao número natural 28. Assim, o F1 pode ser calculado como:

Filho1 = Pai1- parte_Pai1_trocada + parte_Pai2_trocada. (2)

De fato, F1 = 30 (6 - 4 + 28). Assim, os operadores bit-wise têm o objetivo de encontrar o valor natural do material genético a ser trocado. Assim que este valor é encontrado, usa-se a equação 2 para executar nosso *crossover* natural que explora o espaço da busca como *crossover* binário de dois pontos. Assim, dado um pai P1 e dois pontos de corte X e Y (X < Y), usa-se o seguinte algoritmo para se realizar o *crossover*.

Entrada: Pontos de *Crossover* **X, Y** (X < Y); Pais **P1** e **P2**;
Saída: Filhos **F1** e **F2**
Inicio:
1- AuxP1 ← P1 >> X;
2- AuxP1 ← AuxP1 << X;
3- AuxP1' ← P1 >> Y;
4- AuxP1' ← AuxP1' << Y;
5- Parte_Trocada_P1 = AuxP1 - AuxP1'; *//Calcula valor do material genético do P1 a ser trocado.*
6- AuxP2 ← P2 >> X;
7- AuxP2 ← AuxP2 << X;
8- AuxP2' ← P2 >> Y;
9- AuxP2' ← AuxP2' << Y;
10- Parte_Trocada_P2 = AuxP2 - AuxP2'; *//Calcula valor do material genético do P2 a ser trocado.*
11- F1 ← P1 - Parte_Trocada_P1 + Parte_Trocada_P2; *//Cálculo do F1;*
12- F2 ← P2 - Parte_Trocada_P2 + Parte_Trocada_P1; *//Cálculo do F2;*
Fim;

Figura 5. Algoritmo do Nosso *Crossover* Natural Discreto

4. O *Crossover* Natural Contínuo de Aguilar-Ruiz et al

Esta seção descreve o *crossover* natural contínuo de Aguilar-Ruiz et al e explica as razões que nos levaram a desenvolver um novo *crossover* natural contínuo. Os atributos contínuos discretizados são apresentados em uma tabela que é usada para executar a mutação e o *crossover* entre os intervalos. Cada intervalo é representado por um número natural.

Exemplo 7: Assuma que o algoritmo de discretização retornou o vetor com 5 cortes: {1.1, 3.7, 4.8, 5.2, 6.2}. A tabela usada para o operador de Aguilar-Ruiz et al é dada por:

Tabela 1. Intervalos de um Atributo Contínuo.

Pontos de Corte	3.7	4.8	5.2	6.2
1.1	**1**	**2**	**3**	**4**
3.7	-	**6**	**7**	**8**
4.8	-	-	**11**	**12**
5.2	-	-	-	**16**

Os números naturais na tabela representam os intervalos. Assim, o intervalo [1.1, 3.7] é representado pelo número natural 1. Considere as seguintes definições:

Linha e Coluna: Considere n um número natural em uma tabela de discretização e k o número dos cortes. A linha (l) e a coluna (c) de n são dadas por:

$$l = (n-1)/(k-1)+1\,,\quad c = (n-1)\%(k-1)+1 \tag{3}$$

Exemplo 8: Para tabela 1, e $n_i = 2$ e $n_j = 8$. Tem-se: l(2) = 1, c(2) = 2, l(8) = 2, c(8) = 4.

Limites: Os limites de um número natural n são aqueles valores que limitam n nos quatro sentidos possíveis: acima, para baixo, de esquerdo, de direito. Assim, estes valores são respectivamente dados por:

$$lci(n) = c,\ \ lba(n) = (k-1)(c-1)+c\,,\ les(n) = (k-1)(l-1)+l,\ ldi(n) = (k-1)l \tag{4}$$

Exemplo 9: Do exemplo 8: lci(2) = 2, lba(2) = 6, les(2) = 1, ldir(2) = 4, lci(8) = 4, lba(8) = 16, les(8) = 6, ldi(8) = 8.

Deslocamentos: Os deslocamentos adjacentes, para cima, baixo, esquerda e direita são respectivamente dados por:

$$cim(n) = \max(lci(n), n-k+1),\ bai(n) = \min(lba(n), n+k-1),\ esq(n) = \max(les(n), n-1),\quad dir(n) = \min(ldi(n), n+1) \tag{5}$$

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

$$hor(n) = \bigcup_{i=1}^{k-1} \max(les(n), (k-1)(l-1)),\quad ver(n) = \bigcup_{i=1}^{k-1} \max(lba(n), (k-1)(i-1)) \tag{6}$$

Exemplo 10: Do exemplo 9, 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}.

Observe que quase todas estas definições são facilmente calculadas. Entretanto, os deslocamentos horizontais e verticais dependem do número de cortes k, assim este cálculo necessita possivelmente uma estrutura da repetição. A mutação de Aguilar-Ruiz et al e o seu *crossover* natural contínuo podem ser definidos como:

Mutação Natural: Dado um número natural n, a Mutação Natural de n, denotado por Mut (n), é qualquer valor próximo a n usando todos os deslocamentos e distinto de n.

Exemplo11: Mut(2) ∈ { {1, 2 ,3 ,6} – {2} }, i.e., Mut(2) ∈ {1, 3, 6}. Qualquer valor pode ser escolhido.

Crossover Natural: O *crossover* natural entre n_i e n_j, denotados por Cross (n_i, n_j), é obtido como:

$$Cross(n_i, n_j) \in ((hor(n_i) \cap ver(n_j)) \cup (hor(n_j) \cap ver(n_i))) \tag{7}$$

Exampo 12: Cross(2, 8) ∈ { { {1,2,3,4}∩{4,8,12,16} }∪{ {6,7,8}∩{2,6} } }, i.e. Cross(2, 8) ∈ {4, 6}.

O *crossover* natural de Aguilar-Ruiz et al pode ser entendido como um operador que troca o limite superior dos intervalos que são cruzados. De fato, para o exemplo 6, os pais são 2 = [1.1, 4.8] e 8 = [3.7, 6.2] e os filhos são 4 = [1.1. 6.2] e 6 = [3.7, 4.8]. Mas o cruzamento para 1 = [1.1, 3.7] e 16 = [5.2, 6.2] retorna somente o intervalo 4 = [1.1, 6.2], i.e., há somente um filho, e o intervalo 7 = [3.7, 5.2] falta. 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.

5. Novo Operador Natural Contínuo

Nosso *crossover* natural contínuo trabalha com o mesmo vetor de cortes retornado por um algoritmo de discretização, porém, a primeira diferença se faz na numeração dos intervalos. Considere o mesmo vetor dos cortes usado pelo operador de Aguilar-Ruiz et al no exemplo 7, i.e, {1.1, 3.7, 4.8, 5.2, 6.2}. A numeração da matriz para nosso *crossover* é:

Tabela 2. Intervalos de um Atributo Contínuo.

	1.1	3.7	4.8	5.2	6.2
1.1	1	2	3	4	5
3.7	6	7	8	9	10
4.8	11	12	13	14	15
5.2	16	17	18	19	20
6.2	21	22	23	24	25

A matriz é simétrica se considerarmos os valores que limitam cada intervalo. Para este tipo de representação, têm-se duas equações que serão usadas pelo nosso operador natural de *crossover*. A primeira equação retorna a posição (linha e coluna) na tabela de um número natural, visto que a segunda retorna o número natural da tabela, dado sua posição (linha e coluna). Considere, portando, as definições:

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

$$lin = q + ceil(r/(r+1)),\ col = r + floor((k-r)/k)*k,\ \text{onde q e r são dados por:}$$
$$q = floor(n/k),\ r = n\%k \quad (8)$$

Onde: ceil(x) retorna o menor inteiro não menor que x, floor(x) retorna o maior inteiro que é menor ou igual a x e o operador % retorna o resto da divisão inteira.

Exemplo 13: para a tabela 2, com $n_i = 2$ e $n_j = 20$, tem-se:

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

Posição do número natural: Dada uma linha, uma coluna e o número de cortes k, o número natural n que ocupa a posição (linha, coluna) é dado por:

$$n = k(linha - 1) + coluna \quad (9)$$

Exemplo 14: Deseja-se encontrar o número natural n que está na linha = 3 e coluna = 4, para k = 5 (tabela 2). Assim: n = 5 * 2 + 4 = **14**

Nosso operador de *crossover* é baseado nas duas definições acima e no fato de que a matriz de busca é simétrica. Nosso operador retorna sempre dois filhos produzidos por dois pais. Às vezes, é necessário corrigir a posição do segundo filho gerado. Em seguida, apresentamos o algoritmo de nosso *crossover* natural contínuo.

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

```
Inicío
1- Ordene_Pais(p1, p2);                    //Coloca o p1 como o menor pai e p2 como maior.
2- Pega_Linha_Coluna(p1, k, linP1, colP1); //Pega a linha e coluna de p1 e  os guarda respectivamente em linP1 e colP1.
3- Pega_Linha_Coluna(p2, k, linP2, colP2); //Pega a linha e coluna de p2 e  os guarda respectivamente em linP2 e colP2.
4- f1 ← p1 + (colP2 – colP1);              //Calcula o valor do filho1 (f1).
5- f2 ← (p1 + p2) – f1;                    //Calcula o valor do filho2 (f2).
6- Pega_Linha_Coluna (f2, k, linF2, colF2); //Pega linha e coluna de f2 os guarda respectivamente em linF2 e colF2.
7 – Se (linF2 > colF2) então:              //Caso o valor de F2 esteja abaixo da diagonal principal:
      o2 ← Pega_Elemento(k, colF2, linF2); //f2 recebe o valor simétrico à sua posição.  (Correção da posição de f2)
   fim-se;
Fim;
```

Figura 6. Nosso Crossover Natural Contínuo

Exemplo 15: Considere, ainda para k = 5 (exemplo 7) e o vetor dos cortes = {1.1, 3.7, 4.8, 5.2, 6.2}, um dos casos em que o operador de Aguilar-Ruiz et al não retorna todos os filhos possíveis. O caso escolhido é representado pelos intervalos [1.1, 3.7] e [5.2, 6.2], representados, respectivamente, pelos números 2 e 20 (tabela 2). Sabe-se que a prole produzida trocando os limites superiores dos intervalos deve ser f1 = [1.1, 6.2] e f2 = [3.7, 5.2], representado, respectivamente, pelos números naturais 5 e 9 (tabela 2). Dessa forma, apresenta-se a execução do algoritmo.

Entrada: p1 ← 2, p2 ← 20, k ← 5;
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.*

Observa-se que nosso operador tem uma melhor capacidade de explorar o espaço da busca. O *crossover* proposto não depende do número de cortes k para ser realizado e não tem que encontrar uma interseção entre dois conjuntos para obter os filhos.

6. Experimentos

Na tabela 3 apresentamos as bases de dados selecionadas do UCI [Blake e Merz 1998].

Tabela 3. Base de dados. #Inst. = Número de Instâncias, #Atr. = Número de Atributos, #Real = Número de Atributos Reais, #Nom. = Número de Atributos Nominais, #Cla. = Número de Classes.

Name	#Inst.	#Atr.	#Real	#Nom.	#Cla.
bre	286	9	-	9	2
cr-g	1000	20	7	13	2
gls	214	9	9	-	6
he-c	303	13	6	7	2
hep	155	19	6	13	2
h-col	368	22	7	15	2
son	208	60	60	-	2
vehi	846	18	18	-	4

Tabela 4. Parâmetros do AG[1].

Parâmetro	Valor
Tamanho da População	45
Algoritmo de Seleção	Torneio
Tamanho do Torneio	2
Probabilidade de Crossover [2]	0.8
Probabilidade de Mutação do Individuo [2]	0.05
Número de Gerações	200
Número para ativar a Sub-Geração [3]	50
Número de Sub-gerações	30

Os resultados 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). Para atributos contínuos, foi usado o método do discretização proposto em [Fayyad e Irani 1993] e sua implementação no Weka [Witten e Frank 2005].

Tabela 5. Resultados: #Reg. = Número Médio de Regras, #Atr. = Número Médio de Atributos Usados, Acc = Média de Precisão, T-Teste = T-Teste é estatisticamente significante (sim/não).

	Algoritmo Genético			Algoritmo C4.5			
Nome	#Reg	#Atr	Acc	#Reg	#Atr	Acc	T-teste
bre	1	2,4	76.1	7	15	72.7	Sim
cr-g	2.2	6.6	74.4	60	273	72.4	Sim
gls	9	27.8	78.5	25	97	74.4	Sim
he-c	2.6	6.4	86.8	26	83	82.1	Sim
hep	1.8	3.8	88.4	7	27	82.4	Sim
h-col	1.8	5	84.8	13	49	83.7	Não
son	1.6	5	79.0	12	53	75.1	Sim
vehi	14	82.6	74.0	140	692	70.0	Sim

Os resultados obtidos com o AG foram comparados com a árvore de decisão (C4.5) [Quinlan 1993] executada no Weka. T-testes de Student foram usados a fim analisar e comparar os resultados dos testes, usando um intervalo da confiança de 95%. Os parâmetros do AG (tabela 4) foram ajustados usando o conjunto de validação de cada "fold" para cada série de dados. A função da aptidão é a porcentagem dos exemplos classificados corretamente. Os parâmetros padrão de C4.5 foram usados. Os resultados apresentados são as médias de 20 rodadas pra cada conjunto de teste.

Nosso algoritmo obteve melhores resultados (tabela 5) em todas as 8 séries de dados testadas; em 7 deles a diferença foi estatisticamente significativa. Observe o baixo número de regras e de características usadas pelo AG, mostrando que o MISC combinado com o método proposto para adicionar regras, evolui conceitos bem simples.

6.1 Comparando Nosso *Crossover* Discreto com o *Crossover* de Aguilar-Ruiz et al

Com o objetivo de se comparar nosso *crossover* natural discreto com o operador natural de Aguilar-Ruiz et al, executamos o sistema usando o *crossover* discreto natural de Aguilar-Ruiz et al em todas as séries de dados que possuem ao menos um atributo discreto. Para atributos contínuos, nosso *crossover* natural contínuo foi usado. Os

[1] Claramente, para cada série de dados testada, o sistema foi ajustado com um valor particular de parâmetros. O que se apresenta é média dos parâmetros calculados sobre as series de dados usadas. Isto deve ser considerado para todos os parâmetros mostrados neste artigo.

[2] Os operadores de mutação e de crossover que manipulam o mecanismo da seleção de características têm os mesmos parâmetros.

[3] Quando o algoritmo alcança um número de gerações que é múltiplo de 50, uma regra nova é evoluída até o número de sub-gerações. Esta regra é então é adicionada a população principal, como visto na figura 3.

parâmetros (determinados da mesma forma como na seção anterior) são mostrados na tabela 6.

Tabela 6. Parâmetros do AG[1] para o Crossover Natural Discreto de Aguilar-Ruiz et al.

Parâmetro	Valor
Tamanho da População	50
Algoritmo de Seleção	Torneio
Tamanho do Torneio	2
Probabilidade de *Crossover* [2]	0.8
Probabilidade de Mutação do Individuo [2]	0.05
Número de Gerações	300
Número para ativar a Sub-Geração [3]	60
Número de Sub-gerações	60

Tabela 7. Resultados do AG* com uso do Crossover Natural Discreto de Aguilar-Ruiz et al.

	AG			AG*			
Nom	#Reg	#Atr	Acc	#Reg	#Atr	Acc	T-t
bre	1	2,4	76.1	1.6	6	73.0	Sim
cr-g	2.2	6.6	74.4	4.2	13	71.5	Sim
he-c	2.6	6.4	86.8	4	10.4	82.6	Sim
hep	1.8	3.8	88.4	2.8	6.2	82.9	Sim
h-col	1.8	5	84.8	3	7.6	83.8	Não

Os resultados obtidos usando o operador de Aguilar-Ruiz et al são descritos na tabela 7. Nesta tabela, AG representa os resultados obtidos usando o sistema com nosso operador natural, enquanto AG* representa o sistema com o operador de Aguilar-Ruiz et al. A última coluna da tabela informa os resultados do t-teste com um intervalo da confiança de 95%.

Nosso operador de *crossover* obtém melhores resultados em todas as 5 séries de dados; em 4 deles, os resultados são estatisticamente significativos. Para todas as séries de dados, o AG usou um menor número de regras e de características. Observe que o MISC é usado para AG e AG*. O que indiretamente adiciona características aos conceitos de AG*, é o fato que, para estes testes, o uso do *crossover* natural de Aguilar-Ruiz et al implica no uso de mais regras para se atingir uma precisão comparável. Como o operador de Aguilar-Ruiz et al trabalha como um operador de mutação, este operador parece ser incapaz de fazer o ajuste fino em algumas regiões promissoras do espaço de busca. Assim, 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 operador de *crossover* de Aguilar-Ruiz et al para dados discretos, possui capacidade para encontrar as boas regiões a serem exploradas no espaço da busca (exploração) enquanto não parece ter habilidade suficiente de explorar estas áreas em uma maneira mais refinada (explotação). Verifica-se na tabela 6, em comparação com a tabela 4, que o tamanho da população, o número das gerações e as sub-gerações para o AG*, em comparação com AG, foram aumentados, respectivamente, em 12.5%, em 50% e em 100%. Isso mostra que o operador de Aguilar-Ruiz et al precisa mais tempo para explorar o espaço de busca.

6.2 Comparando Nosso *Crossover* Contínuo com o *Crossover* de Aguilar-Ruiz et al

Selecionaram-se as séries de dados que têm somente atributos contínuos e o AG foi executado usando o *crossover* natural contínuo de Aguilar-Ruiz et al. O MISC foi usado também nestas experiências para ambos os AGs. Os parâmetros (determinados como na seção 6) são mostrados na tabela 8.

Tabela 8. Parâmetros do AG[1] para o Crossover Natural Contínuo de Aguilar-Ruiz et al.

Parâmetro:	Valor:
Tamanho da População	50
Algoritmo de Seleção	Torneio
Tamanho do Torneio	2
Probabilidade de *Crossover* [2]	0.8
Probabilidade de Mutação do Individuo [2]	0.05
Número de Gerações	250
Número para ativar a Sub-Geração [3]	60
Número de Sub-gerações	40

Para todas as séries de dados examinadas, o AG com o operador de Aguilar-Ruiz et al obteve a mesma precisão que quando nosso operador foi usado. Observe que, em comparação com a tabela 4, o tamanho da população aumentou 12.5%, o número de gerações aumentou 25% e o número de sub-gerações aumentou 33.3%. Estes resultados mostram que nosso operador natural contínuo tem uma maior capacidade exploração.

7. Conclusão

Este trabalho avaliou o operador natural discreto proposto em nosso trabalho anterior [Pitangui e Zaverucha, 2006]. Adicionalmente, propusemos um operador natural para dados contínuos e comparamos sua eficiência com o *crossover* natural de Aguilar-Ruiz et al.

Os resultados mostram que nosso operador natural discreto obteve melhores resultados em todas as séries de dados usando menos tempo, regras e atributos, quando comparado ao operador natural discreto de Aguilar-Ruiz et al. Mostrou-se que o operador natural discreto de Aguilar-Ruiz et al gera mais filhos quando comparado aos *crossover* binário de dois pontos, mas é incapaz de produzir todos os filhos gerados pelo *crossover* binário de dois pontos. O operador natural discreto de Aguilar-Ruiz et al comporta-se como um operador de mutação, possuindo boa força de exploração, mas não explora as áreas promissoras de forma mais refinada. Mostrou-se ainda que nosso *crossover* natural discreto explora o espaço da busca como o *crossover* binário de dois pontos.

Nosso *crossover* natural contínuo reduziu o tempo usado para se obter os mesmos resultados alcançados com o *crossover* natural contínuo de Aguilar-Ruiz et al.

As prioridades para um trabalho futuro são: verificar a eficiência de nossa abordagem hibrida que funde os estilos de Pittsburgh e de Michigan e analisar o MISC.

8. Referências

Aguilar-Ruiz Jesús S. , Riquelme J.C. e Carmelo D. V. "Improving the Evolutionary Coding for Machine Learning Tasks". *European Conference on Artificial Intelligence*, IOS Press, Lyon, France 2002, pp 173-177

Aguilar–Ruiz Jesús. S., Riquelme J. C. e Toro M.. "Evolutionary Learning of Hierarchical Decision Rules." *IEEE Transactions on Systems, Man e Cybernetics, Part B*, 33(2), 2003, pp. 324–331.

Bacardit, J. e Butz, M.V., "Data Mining in Learning Classifier Systems: Comparing XCS with GAssist", *7th Inter. Workshop on Learning Classifier Systems*, Springer-Verlag, Seattle, USA, 2004, pp. 381-387.

Blake, C.L. e Merz, C.J., "UCI Repository of machine learning databases*",* Irvine, CA: University of California, Department of Information e Computer Science, 1998, http://www.ics.uci.edu/#mlearn/MLRepository.html

DeJong, K. A. e Spears, W. M. 1991. "Learning Concept Classification Rules Using Genetic Algorithms." *12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann., Sydney, Australia, 1991., pp. 51-56.

DeJong K. A, W. M. Spears, e D. F. Gordon, "*Using genetic algorithms for concept learning*," Machine. Learning, vol. 1, no. 13, 1993, pp. 161-188.

Eiben A.E. e Schippers A., "On Evolutionary Exploration e Exploitation**".** *Fundamenta Informaticae*, IOS Press, Amsterdam, The Netherles, Vol. 35, Issue 1-4, 1998, pp. 35-50

Fayyad U. M. e 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, 1993, pp. 1022-1027.

Goldberg., David. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts, 1989.

Pitangui, C., Zaverucha, G. "Genetic Based Machine Learning: Merging Pittsburgh and Michigan, an Implicit Feature Selection Mechanism e a New Crossover Operator". *6th International Conference on Hybrid Intelligent Systems.* Auckland, New Zealand, 2006.

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

Witten Ian H. e Frank Eibe (2005) *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.