UNIVERSIDADE FEDERAL DE PELOTAS
INSTITUTO DE FÍSICA E MATEMÁTICA
CURSO DE BACHARELADO EM INFORMÁTICA

Algoritmos Genéticos: um estudo de seus conceitos fundamentais e aplicação no problema de grade horária

por
Diogo Correa Lucas

Orientador: Luiz Antônio Moro Palazzo
Co-orientador: Anderson Priebe Ferrugem

Pelotas, Dezembro, 2000

Agradecimentos

A meus pais que me incentivaram e apoiaram ao longo de toda esta jornada, aos professores: Anderson Ferrugem e Luiz Antônio Palazzo sem a ajuda dos quais este trabalho não poderia ser realizado, João Artur de Souza, por sua prestatividade e constante boa vontade e Sérgio Cardoso, por todas observações que auxiliaram na melhora deste trabalho.

Sumário

Lista de Abreviaturas e siglas

1PX	*One-Point Crossover*
2PX	*Two-Point Crossover*
AG	Algoritmo Genético
CE	Computação Evolutiva
CSF	*Classifier System*
CX	*Cycle Crossover*
e.g.	*exempli gratia*
FM	*Flip Mutation*
HBI	*Heuristic Block Initializer*
HSI	*Heuristic Slot Initializer*
IA	Inteligência Artificial
MBX	*Multi-Point Block Crossover*
MPX	*Multi-Point Crossover*
PG	Programação Genética
PMX	*Partial Match Crossover*
SBM	*Swap Block Mutation*
SM	*Swap Mutation*
SX	*Segmented Crossover*
URI	*Uniform Random Initializer*

Lista de Figuras

Lista de Tabelas

Resumo

Os algoritmos genéticos utilizam conceitos provenientes do princípio de seleção natural de Darwin para abordar uma série ampla de problemas, em especial de otimização. Robustos, genéricos e facilmente adaptáveis, consistem de uma técnica amplamente estudada e utilizada em diversas áreas.

Neste trabalho é feita uma descrição dos elementos básicos da teoria de algoritmos genéticos, bem como a implementação de um aplicativo experimental para a resolução de um problema de grade horária, com o objetivo de medir alguns índices de desempenho entre várias técnicas desta metodologia.

Palavras-chave:

algoritmos genéticos, híbridos, gerenciamento de horários, grade horária

Abstract

Genetic algorithms have their logic based on Darwin's concept of natural selection and are used to solve a large number of problems, specially in the optimization field. Robust, generic and easily adaptable, they consist of a fairly known and used technique.

In this text we make a description of genetic algorithms basic concepts, as well as the implementation of an experimental application for solving the time table problem (accompanied by the analysis of various possible techniques involving this method).

Keywords: *genetic algorithms*, *timetabling*, *hybridization*.

1. Introdução

1.1. Apresentação

Os algoritmos genéticos constituem uma técnica bastante utilizada em problemas de otimização, e baseiam a lógica de seu funcionamento nas leis de evolução natural propostas por Charles Darwin – as possíveis respostas geradas para o problema são vistas como indivíduos que competirão entre si pela oportunidade de se reproduzirem. Neste processo, os mais aptos, *i*.e., que representam uma melhor solução, têm maior chance de perpetuar parte de suas características, aumentando assim a probabilidade de se obter uma maior adaptação da população em geral.

O termo genético diz respeito à maneira como as possíveis soluções para o problema são codificadas em genomas, uma estrutura composta por uma cadeia finita de elementos, os genes. Seguindo ainda a analogia com a natureza, é comum encontrar autores que classifiquem os indivíduos da população como genótipos e sua resposta resultante de sua decodificação como fenótipo. Os AGs são então capazes de resolver problemas cuja solução possa ser codificada numa *string* de valores pertencentes a um alfabeto pré-definido.

A base de funcionamento dos algoritmos genéticos pode facilmente ser adaptada à solução de uma grande variedade de problemas, e em muitos casos a parte de codificação efetivamente dependente do domínio (*domain dependant*) destes pode ser limitada à escolha de representação (um ponto crítico para seu bom funcionamento, como veremos mais adiante) e da função de avaliação dos indivíduos (também chamada de função objetivo).

Os algoritmos genéticos são amplamente utilizados como otimizadores de funções, nos casos em que os métodos de busca exaustiva falham e é necessário o uso de busca controlada em espaço. Sua vantagem sobre outros métodos neste aspecto é seu alto grau de adaptabilidade, robustez e paralelismo.

1.2. Características Gerais

Por causa da maneira particular como os AGs operam, neles se destacam as seguintes características:

a) busca codificada – Segundo [PER96], "os AGs não trabalham sobre o domínio do problema, mas sim sobre representações de seus elementos". Tal fator impõe ao seu uso uma restrição: para resolver um problema é necessário que o conjunto de soluções viáveis para este possa ser de alguma forma codificado em uma população de indivíduos;

b) generalidade – Os algoritmos genéticos simulam a natureza em um de seus mais fortes atributos: a adaptabilidade. Visto que a representação e a avaliação das possíveis soluções são as únicas partes (de um considerável conjunto de operações utilizadas em seu funcionamento) que obrigatoriamente requisitam conhecimento dependente do domínio do problema abordado [WHI00], basta a alteração destas para portá-los para outros casos. A preocupação de um programador de AGs não é então *de que forma* chegar a uma solução, mas sim *com o que ela deveria se parecer*;

c) paralelismo explícito – o alto grau de paralelismo intrínseco aos AGs pode ser facilmente verificado se considerarmos o fato de que cada indivíduo da população existe como um ente isolado e é avaliado de forma independente. Se na natureza todo processo de seleção ocorre de forma concorrente, nos AGs essa característica se repete. Os modelos de ilha, descritos na seção 4.2:Modelo de Ilhas e em [COS99] e [WAL96a], foram criados para explorar tal característica;

d) busca estocástica – segundo [GEY97] e [MIC99], ao contrário de outros métodos de busca de valores ótimos, os algoritmos genéticos não apresentam um comportamento determinístico. Não seria correto, no entanto, afirmar que tal busca se dá de forma completamente aleatória, pois na verdade o fator aleatório opera no nível local da busca servindo como orientação para o comportamento do programa;

e) busca cega – de acordo com [GEY97] e [PER96], um algoritmo genético tradicional opera ignorando o significado das estruturas que manipula e qual a melhor maneira de trabalhar sobre estas. Tal característica lhe confere o atributo de

não se valer de conhecimento específico ao domínio do problema, o que lhe traz generalidade por um lado, mas uma tendência a uma menor eficiência por outro;

f) eficiência mediana – por constituir um método de busca cega, um algoritmo genético tradicional tende a apresentar um desempenho menos adequado que alguns tipos de busca heurística orientadas ao problema. Para resolver tal desvantagem, a tática mais utilizada é a hibridação [DAV91],[GEY97],[BUR95], onde heurísticas provenientes de outras técnicas são incorporadas;

g) paralelismo implícito: a partir do teorema dos esquemas de Holland, tem-se que ao fazer uma busca por populações, a evolução de um algoritmo genético tende a favorecer indivíduos que compartilhem determinadas características, sendo assim capaz de avaliar implicitamente determinadas combinações ou esquemas como mais ou menos desejáveis, efetuando o que chamamos uma busca por hiperplanos, de natureza paralela [GOL89];

h) facilidade no uso de restrições: ao contrário de muitos outros métodos de busca, os AGs facilitam a codificação de problemas com diversos tipos de restrição, mesmo que os últimos apresentem graus diferentes de importância [BAR96]. Neste caso, se dois indivíduos violam restrições, é considerado mais apto aquele que viola as mais flexíveis (*soft constraints*) em detrimento do que viola as mais graves (*hard constraints*);

2. Funcionamento

Como descrito anteriormente, um algoritmo genético opera sobre uma população, fazendo com que esta evolua de acordo com uma função de adaptação. Seu funcionamento começa com a inicialização da população, e depois o processo de seleção, reprodução e mutação ocorre a cada geração até que seja atingido algum critério para o fim da evolução. Sua estrutura de funcionamento é a seguinte:

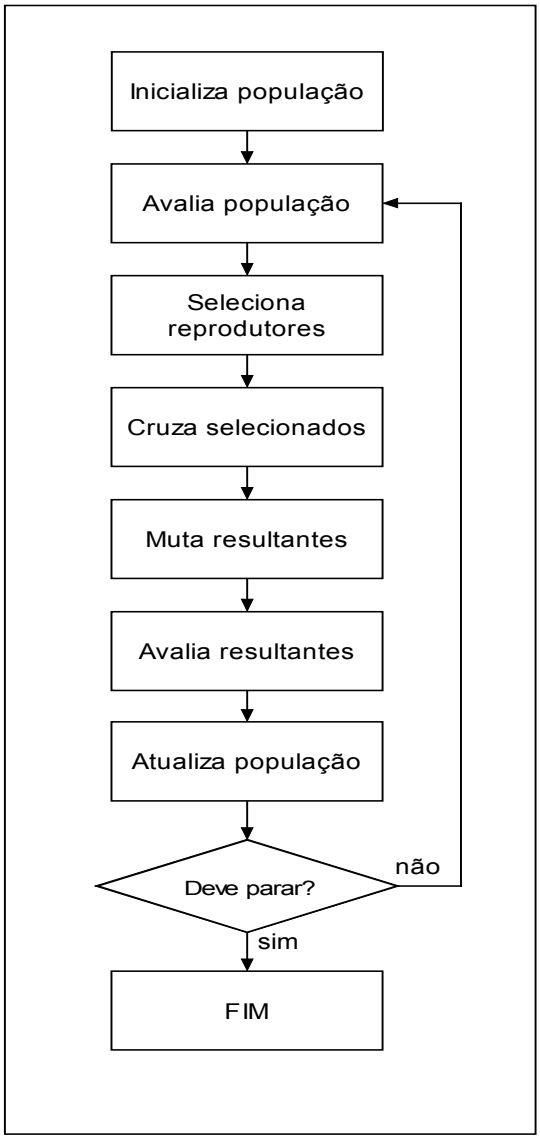

Figura 1: A estrutura de funcionamento de um AG tradicional.

2.1. Inicialização

A inicialização básica de um algoritmo genético clássico se resume à síntese de uma população inicial, sobre a qual serão aplicadas as ações dos passos subseqüentes do processo. Tipicamente se faz uso de funções aleatórias para gerar os indivíduos, sendo este um recurso simples que visa a fornecer maior "biodiversidade*", fundamental para garantir uma boa abrangência do espaço de pesquisa. Existem várias alternativas ao método randômico, destinadas a suprir deficiências no que diz respeito a dificuldades existentes quanto à criação aleatória de indivíduos de representação mais complexa e, fator mais considerado, a uma melhora na performance. Podemos citar como exemplo o uso de algoritmos de busca heurística como geradores de populações iniciais, especialmente em casos que apresentem um alto grau de restrições, onde o AG recebe uma população que ainda não possui indivíduos ótimos, mas que apresentam pelo menos algumas das características desejadas. Os operadores de inicialização mais tradicionais são, segundo [GOL89] e [GEY97]:

a) Inicialização randômica uniforme: cada gene do indivíduo receberá como valor um elemento do conjunto de alelos†, sorteado de forma aleatoriamente uniforme.
b) Inicialização randômica não uniforme: determinados valores a serem armazenados no gene tendem a ser escolhidos com uma freqüência maior do que o restante.
c) Inicialização randômica com "dope": indivíduos otimizados são inseridos em meio à população aleatoriamente gerada. Esta alternativa apresenta o risco de fazer com que um ou mais super indivíduos tendam a dominar no processo de evolução e causar o problema de convergência prematura‡.

---

* o termo biodiversidade aqui empregado pertence à tradição existente no meio da computação evolutiva de utilizar, com certa liberdade, termos da biologia.

† também pertencente a biologia, o termo conjunto de alelos diz respeito ao conjunto de valores que um gene pode assumir.

‡ a definição do termo convergência prematura, assim como a de vários outros importantes para o estudo dos AGs podem ser encontrados no Anexo 1: Glossário. Maiores informações sobre o problema da convergência prematura podem ser encontrados ao longo do texto, em especial na seção 6.8.1: Dificuldades encontradas

d) Inicialização parcialmente enumerativa: são inseridos na população indivíduos de forma a fazer com que esta comece o processo de evolução possuindo todos os esquemas possíveis de uma determinada ordem.

2.2. Avaliação

Nesta etapa, o primeiro passo da seleção em si é dado: o universo da população sofre, indivíduo a indivíduo, um processo de avaliação, que visa a representar seu grau de adaptação. Na verdade, este é, em conjunto com a escolha da representação, o ponto do algoritmo mais dependente do problema em si – é necessário aqui que o AG seja capaz de responder sobre quão boa uma resposta é para o problema proposto. Atualmente, várias formas de avaliação são utilizadas: em casos de otimização de funções matemáticas, tende a ser escolhido o próprio valor de retorno destas aplicado ao indivíduo, e em problemas com muitas restrições, funções baseadas em penalidades são mais comuns. A função de avaliação também é chamada de função objetivo em um grande número de trabalhos.

2.3. Seleção

É no estágio de seleção que os indivíduos são escolhidos para posterior cruzamento. Neste ponto, fazendo uso do grau de adequação de cada um, é efetuado um sorteio onde os mais aptos possuem maior probabilidade de se reproduzirem. Este grau é calculado a partir da função de avaliação de cada indivíduo, e determina quão apto ele está para a reprodução em relação à população a que pertence.

2.3.1. *Scaling*

De acordo com [GOL89], [WAL96b] e [GEY97], o uso de um método de *scaling* sobre o valor de adaptação de cada indivíduo pode ser útil por reduzir a probabilidade de convergência prematura. Alguns dos métodos mais utilizados são:

a) *no scaling*: O valor de retorno da função objetivo é usado sem nenhum tipo de alteração.

b) *linear scaling*: O valor de retorno da função objetivo ($f_a(x)$) sofre a seguinte alteração:

$$f_{ls}(f_a(x)) = f_a(x) \times a + b$$

onde $a = \dfrac{(c-1) \times média}{\Delta}$ e $b = \dfrac{média \times (máximo - c * média)}{\Delta}$ se $\Delta \neq 0$,

$a = 1$ e $b = 0$ se $\Delta = 0$,

máximo representa o maior valor de adaptação encontrado; *média* a média de adaptação da população e c é uma constante pré-definida e é um número real maior do que 1.

Valores negativos de adaptação não são aceitos por este método.

c) *sigma truncation scaling*: um múltiplo do desvio médio (denotado por *desvio*) é subtraído dos valores de adaptação. Todos valores negativos são alterados para zero. A fórmula para seu cálculo é a seguinte:

$$f_{sts}(f_a(x)) = f_a(x) - (média - c \times desvio)$$

onde c é uma constante pré-definida pertencente ao conjunto dos reais e maior do que zero.

d) *power law scaling*: o valor de adaptação do indivíduo é elevado a uma constante pré-definida *k*.

$$f_{pls}(f_a(x)) = f_a(x)^k$$

Este método não aceita valores negativos de adaptação.

2.3.2. Métodos de seleção

Existem várias formas para efetuar a seleção, dentre as quais destacam-se:

a) seleção por *ranking* (*Rank Selection*): ps indivíduos da população são ordenados de acordo com seu valor de adequação e então sua probabilidade de escolha é atribuída conforme a posição que ocupam;

b) seleção por giro de roleta (*Roulette Wheel Selection*): de acordo com [DAV91] o funcionamento do método de seleção por giro de roleta é o seguinte:

calcula $total = \sum_{i=1}^{popsize} adequação(x)$

sorteia um valor s tal que $s \in [0; total]$

seleciona o indivíduo x tal que $s \in \left[\sum_{i=1}^{x-1} adequação\ (x); \sum_{i=1}^{x} adequação\ (x) \right]$;

c) seleção por torneio (*Tournament Selection*): Grupos de soluções são escolhidos sucessivamente e as mais adaptadas dentro de cada um destes são selecionadas[GOL89][WAL96a][GEY97];

d) seleção Uniforme: todos indivíduos possuem a mesma probabilidade de serem selecionados [WAL96a]. Obviamente, esta forma de seleção possui uma probabilidade muito remota de causar a evolução da população sobre a qual atua;

e) *remainder Stochastic Selection*: a partir da fórmula:

$$s(x) = \frac{adequação(x)}{média}$$

se assume que a parte inteira da função s(x) determina quantas cópias do indivíduo são selecionadas diretamente. O restante de indivíduos a serem selecionados é escolhido aplicando um método de seleção como o giro de roleta sobre a parte decimal do valor s(x) de cada indivíduo [WHI00][GOL89][GEY97];

2.4. Reprodução

Uma vez selecionados os indivíduos, estes passam com uma probabilidade pré-estabelecida pelo processo de cruzamento (*crossover*), onde partes dos genes dos pais são combinadas para geração de filhos.

2.4.1. Escolha de pares

Alguns dos principais métodos de escolha dos pares de reprodutores são, segundo [GEY97]:

a) escolha aleatória: os pares de indivíduos que devem se reproduzir são escolhidos ao acaso;

b) *inbreeding*: parentes são combinados;

c) *line breeding*: um indivíduo de alta performance é cruzado com uma sub-população de indivíduos e os filhos são selecionados como pais;

d) *outbreeding*: indivíduos que codificam fenótipos diferentes são combinados;

e) *self-fertilization*: o indivíduo é combinado consigo mesmo;

f) *positive assortive mating:* indivíduos semelhantes são combinados;

g) *negative assortive mating*: indivíduos diferentes são combinados.

2.4.2. Operadores de cruzamento

Na verdade, é possível representar o funcionamento dos operadores de cruzamento como uma seleção por máscara: esta seria representada por um vetor cujos elementos possam assumir valores binários e que possua um comprimento igual ao dos cromossomas a serem combinados*. Seu uso pelo operador se daria segundo o seguinte algoritmo:

se $\text{máscara}_i = 0$
  então
    $\text{irmã}_i = \text{mãe}_i$
    $\text{irmão}_i = \text{pai}_i$
  senão
    $\text{irmão}_i = \text{mãe}_i$
    $\text{irmã}_i = \text{pai}_i$

Os principais operadores tradicionais de cruzamento para genomas de comprimento fixo são [GOL89]:

a) cruzamento de um ponto (*1PX*): dados dois genomas x e y de comprimento lg, sorteia-se um número p qualquer tal que $0 < p < lg$, o primeiro filho f_0 receberá todos os genes x de 1 até p e todos os genes y de p+1 até l_y, e o segundo filho o inverso. A máscara de cruzamento seria uma série de 0s (zeros) sucedidas de 1s (uns) pertencente ao conjunto $\{0^n1^m \mid n \geq 0, m \geq 0, n + m = lg\}$.

---

* [COS99] faz referência ao uso de uma máscara de cruzamento para a aplicação do operador UX.

Um exemplo de funcionamento do operador de cruzamento de 1 ponto pode ser observado na Figura 2, onde, dois genomas de comprimento 20 (os superiores na figura) são cruzados para gerar filhos. Considerando que a coloração dos quadrados indica o valor contido no gene aos quais eles correspondem (claro para 0 e escuro para 1), um ponto de corte é sorteado de forma a cair entre o início e o fim do genoma (no exemplo tal ponto é na posição 11). Caso se faça uso de uma máscara de cruzamento, esta é preenchida com zeros até o elemento de índice igual a posição de corte, e a partir daí com uns até seu fim. É possível observar na figura o resultado final da operação: o filho 1 recebe os genes do pai 1 de índice 1 a 11 e do pai 2 de 12 a 20, e o filho 2 os genes restantes, ou seja, os de 1 a 11 do pai 2 e os de 12 a 20 do pai 1.

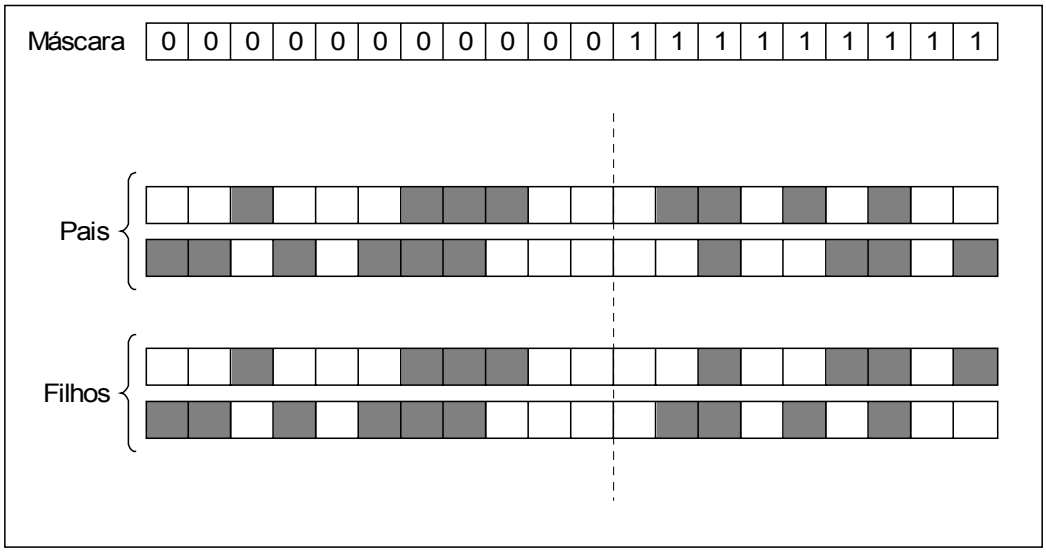

Figura 2: O operador de cruzamento de um ponto (1PX).

Dentre os operadores de cruzamento tradicionais o de um ponto é o que normalmente apresenta o pior desempenho.

b) cruzamento por 2 pontos (2PX): semelhante ao operador de cruzamento por um ponto, neste caso são escolhidos dois pontos de corte. A máscara de cruzamento seria pertencente ao conjunto $\{0^n1^m0^o \mid n \geq 0, m \geq 0, o \geq 0, n + m + o = lg\}$;

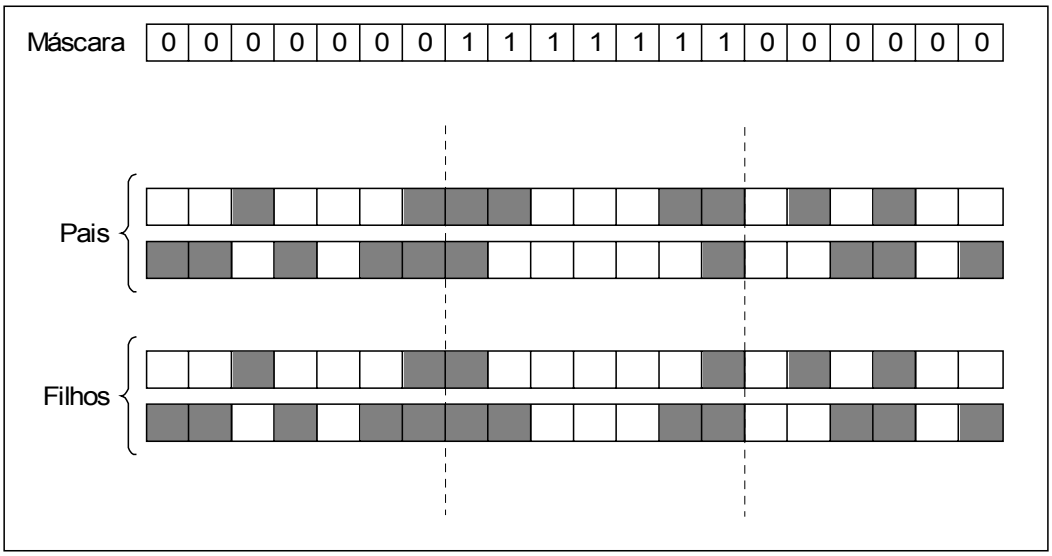

Figura 3: O operador de cruzamento de dois pontos(2PX).

c) cruzamento Multi-Ponto (MPX): o cruzamento multi-ponto é uma generalização dos operadores apresentados anteriormente. Nele são sorteados um número fixo *n* de pontos de corte. Como regra geral, uma operação de cruzamento de *n* pontos pode ser vista como um caso especial de uma de $m > n$ pontos em que um ou mais $(m - n)$ pontos de corte selecionados se localizam no final do cromossoma (o ponto de índice $i = lg$). Um operador com *n* pontos de cruzamento apresentaria uma máscara de cruzamento com *n* mudanças em sua seqüência de zeros e uns;

d) cruzamento Segmentado (SX): o cruzamento segmentado funciona de maneira semelhante ao multi-ponto, com a exceção de que sorteia o número de pontos de corte toda vez que é executado;

e) cruzamento Uniforme (UX): para cada gene a ser preenchido nos cromossomas filhos, o operador de cruzamento uniforme sorteia de qual dos pais este deve ser gerado. A máscara de cruzamento de tal operador é uma seqüência qualquer de zeros e uns;

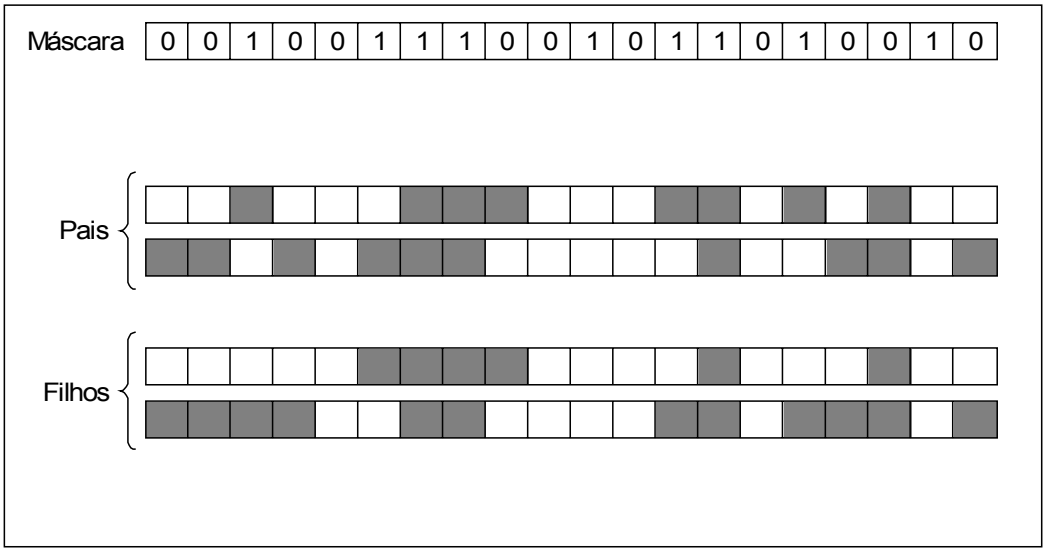

Figura 4: O operador de cruzamento uniforme (UX).

f) cruzamento por combinação parcial (PMX): sorteia dois pontos de corte e faz com que os indivíduos filhos recebam na íntegra os genes do pai ou da mãe (mãe para um e pai para outro) situados entre os pontos de corte, depois disso preenche os genes restantes com os valores considerados mais adequados para cada filho;

2.5. Mutação

A mutação opera sobre os indivíduos resultantes do processo de cruzamento e com uma probabilidade pré-determinada efetua algum tipo de alteração em sua estrutura.

a) mutação aleatória (*Flip Mutation*): cada gene a ser mutado recebe um valor sorteado do alfabeto válido;

b) mutação por troca (*Swap Mutation*): são sorteados *n* pares de genes, e os elementos do par trocam de valor entre si;

c) mutação *creep*: um valor aleatório é somado ou subtraído do valor do gene;

2.6. Atualização

Neste ponto, os indivíduos resultantes do processo de cruzamento e mutação são inseridos na população segundo a política adotada pelo AG. Na forma mais tradicional deste (chamada corriqueiramente de algoritmo genético simples), a população mantém um tamanho fixo e os indivíduos são criados em mesmo número que seus antecessores e os substituem por completo. Existem, porém, alternativas a essa abordagem: o número de indivíduos gerados pode ser menor[*], o tamanho da população pode sofrer variações e o critério de inserção pode ser variado (como, por exemplo, nos casos em que os filhos substituem os pais, ou em que estes só são inseridos se possuírem maior aptidão que o cromossoma a ser substituído), ou o conjunto dos *n* melhores indivíduos pode sempre ser mantido[†].

2.7. Finalização

A finalização não envolve o uso de nenhum operador genético: ela simplesmente é composta de um teste que dá fim ao processo de evolução caso o AG tenha chegado a algum ponto pré-estabelecido de parada. Os critérios para a parada podem ser vários, desde o número de gerações já criadas até o grau de convergência da atual população (por convergência entende-se o grau de proximidade dos valores da avaliação de cada indivíduo da população).

---

[*] para uma abordagem mais profunda do assunto, veja a seção 4.1:Populações

[†] chamamos essa característica de *elitismo*. Esta opção de funcionamento é amplamente utilizada, normalmente com apenas um indivíduo compondo a elite, mesmo em algoritmos genéticos simples. Deve ser aplicada com cuidado para se evitar o problema da *convergência prematura*.

3. Considerações

3.1. Esquemas

Consideremos um algoritmo genético que opere sobre uma população de soluções formadas por uma combinação qualquer de *n* bits. A linguagem que constrói tais indivíduos poderia usar como alfabeto S= $\{0, 1\}^n$. Utilizando-se o * como um coringa para designar situações "*don't care"*, é possível definir, por exemplo, para *n*=4 o conjunto C de todos indivíduos gerados a partir S que começam com o número 0 e terminam com o 1 como sendo {0,*,*,1}. Chamamos então esse *string* que define o conjunto, formado a partir de um alfabeto S' (equivalente a um alfabeto básico S básico acrescido do símbolo *), de *esquema*.

3.1.1. Ordem de um esquema

A ordem de um esquema Q, expressa como O(Q), representa o número de posições fixas (elementos pertencentes ao alfabeto original) presentes neste. Por exemplo, sejam os seguintes esquemas:

Q_1={0,1,0,0}
Q_2={*,*,1,*}
Q_3={*,*,*,*}
Q_4={0,*,0,*}

Temos então $O(Q_1) = 4$, $O(Q_2) = 1$, $O(Q_3) = 0$, $O(Q_4) = 2$. Obviamente, esquemas de menor ordem abrangem uma quantidade maior de indivíduos.

3.1.2. Tamanho de um esquema

O tamanho de um esquema Q, denotado por T(Q), é a distância entre o primeiro e o último número fixo deste, para o exemplo anterior temos $T(Q_1) = 4 – 1 = 3$, $T(Q_2) = T(Q_3) = 0$, $T(Q_4) = 3 – 1 = 2$. O tamanho de um esquema é importante para determinar a

probabilidade com que ele se manterá após submetido aos operadores de mutação e cruzamento, como veremos adiante.

O teorema dos esquemas de Holland diz que bons esquemas tendem a se reproduzir de forma exponencial nas gerações subseqüentes.

3.2. Mutação e Cruzamento

Os operadores genéticos de mutação e cruzamento são os responsáveis por todas as transformações sofridas pela população, mas possuem funções bastante distintas no que diz respeito a seu impacto na evolução.

O operador de cruzamento tem como objetivo propagar os esquemas mais adequados na população (ver seção 3.3:Busca por hiperplanos). Para isto, os pontos de corte são fundamentais, pois vão determinar quais esquemas sobreviverão ao processo de reprodução. Em problemas altamente combinatórios um operador cego de cruzamento pode facilmente gerar filhos menos adequados a partir dos cromossomas pais, no fenômeno conhecido como epístase. Veremos mais adiante metodologias utilizadas para evitá-lo.

A mutação é fator fundamental para garantir a biodiversidade, assegurando assim que o espaço de busca provavelmente será explorado em uma parte significativa de sua extensão. Apesar de normalmente aplicada com uma probabilidade bastante inferior à de cruzamento, ela é tida por uma série de autores como o operador mais importante para o processo evolutivo, chegando em alguns casos extremos a ser utilizada como o único operador de transformação no curso de evolução do AG.

O operador de mutação possui também um papel fundamental no que diz respeito à necessidade de evitar a convergência prematura, que ocorre quando a população se estabiliza com uma média de adaptação pouco adequada por causa da pressão evolutiva e baixa diversidade. Isto geralmente se dá com o surgimento de um super-indivíduo que domina o processo seletivo e, uma vez incapaz de gerar filhos melhores, transmite suas características por toda população. Uma análise deste caso será feita na seção 6.8.1:Dificuldades encontradas.

3.3. Busca por hiperplanos

Uma das implicações da prova de Holland, baseada no teorema dos esquemas, é de que os algoritmos genéticos efetuam uma busca por hiperplanos.

Se trabalharmos com um problema cuja solução pode ser codificada por uma *string* binária de tamanho 3, nosso espaço de soluções pode ser representado pelo cubo na Figura 5. Este cubo representa planos em que os indivíduos podem estar localizados. Por exemplo, nos vértices de sua face frontal localizam-se apenas indivíduos que tenham 0 como valor no gene de *lócus*[*] 1, *i*.e., que contenham o esquema {0**}, e nos da face superior apenas os com o esquema {*1*}. Esquemas de ordem 2 representam sempre vértices dos segmentos que compoem o cubo: o esquema {1*0} representa os vértices do segmento que é intersecção dos lados esquerdo e traseiro do cubo.

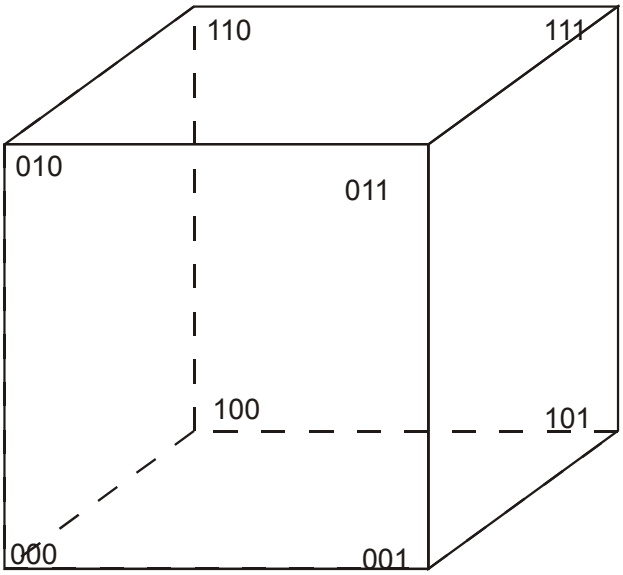

Figura 5: Um cubo de busca [WHI00].

Esta representação torna claro o fato de que esquemas de menor ordem abrangem um número maior de indivíduos. Uma representação de um hiper-cubo de busca em um espaço de 4 dimensões com um alfabeto binário pode ser vista na Figura 6. Nela o gene de *locus* 1 designa a que cubo pertence o esquema, e os demais funcionam de maneira análoga à representação tri-dimensional.

[*] o termo *locus* designa a posição de um gene dentro do cromossoma.

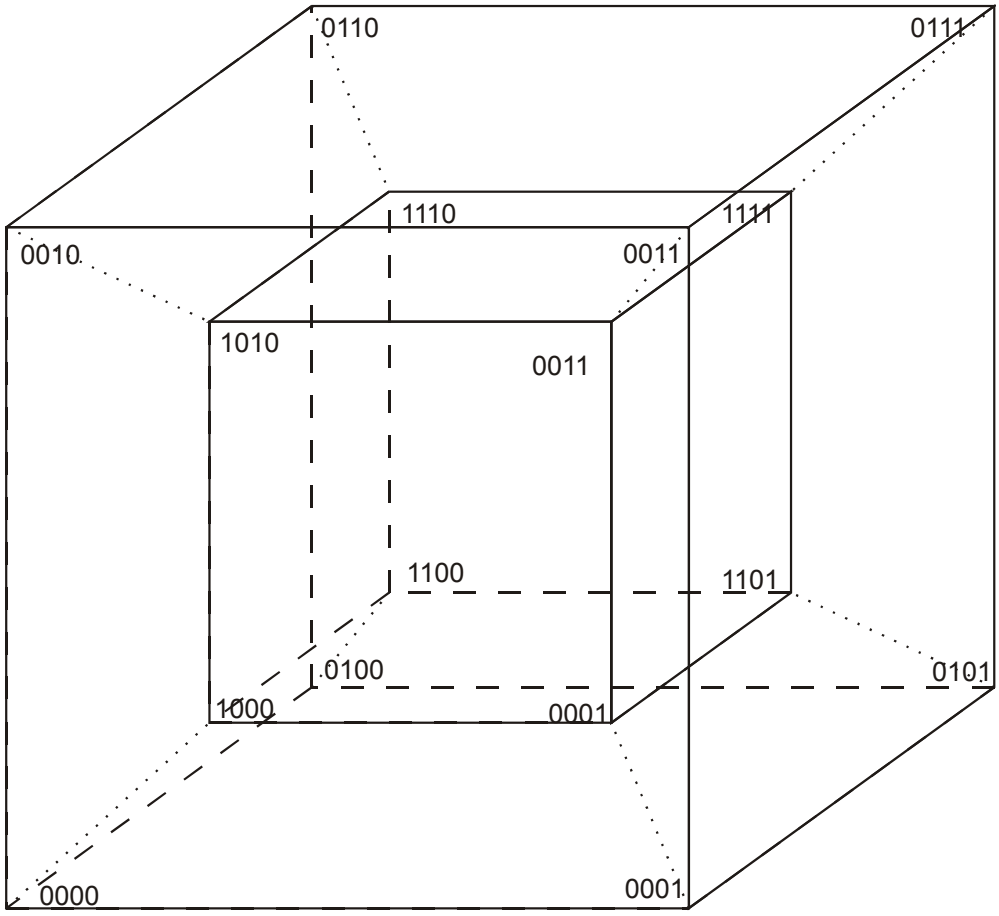

Figura 6: um hiper-cubo de busca 4 dimensões [WHI00].

Segundo [GOL89][WHI00], cada indivíduo gerado a partir de um alfabeto binário está presente em $2^L - 1$ hiperplanos diferentes (onde L é o comprimento do cromossoma e a *string* formada apenas por símbolos * representa todo espaço de busca e não é considerada uma partição do espaço) e é possível gerar 3^L partições do hiperplano.

A busca por populações realizada por um algoritmo genético na verdade faz uma tomada de amostra dos hiperplanos, e a partir desta efetua uma estimava do grau de adaptação de todos indivíduos pertencentes a eles.

Considerando-se que determinados hiperplanos possuem indivíduos com melhor grau de adaptação do que a média, estes tendem a representar parcelas cada vez maiores da população conforme a execução se dá. Em um algoritmo genético que funcione de maneira adequada, o número de indivíduos pertencentes a cada hiperplano após o processo de seleção deveria estar próximo de um valor que pode ser estimado[WHI00].

3.4. Algoritmos Genéticos X Escalada de Montanha

3.4.1. O problema dos ótimos locais

Algoritmos de escalada de montanha (*hill-climbing*) possuem dificuldades de lidar com funções que apresentem grande oscilação em seus valores, por causa de sua tendência a parar sua busca quando encontram um valor ótimo local. Estes últimos se localizam em picos locais, pontos onde a função atinge seu máximo em um intervalo reduzido, sem, contudo, que este seja o pico de maior altura de todo seu domínio.

Por causa de sua busca por hiperplanos, os AGs conseguem evitar este problema sem grandes dificuldades.

3.4.2. Escalada de montanha em um AG

O esquema de busca por hiperplanos foi demonstrado para o algoritmo genético clássico ou canônico e tem seu funcionamento baseado no operador de cruzamento. Para que este tipo de busca funcione, é necessário que a população apresente uma alta biodiversidade, fator que propicia uma amostragem razoável do espaço de busca. É sabido que algumas heurísticas tendem a reduzir esta característica, assim como afetar a maneira como os esquemas vão se propagar. Em alguns casos, tais métodos (muitas vezes combinados com uma inversão na probabilidade de aplicação dos operadores de transformação, *i*.e., com baixa probabilidade de cruzamento e alta de mutação) apresentam um comportamento de escalada de montanha, e são defendidos por uma série de autores (para maiores referências, ver [WHI00]).

4. Técnicas em voga

Existe uma ampla gama de técnicas de implementação de AGs que servem como alternativa à tradicional. É importante ressaltar aqui que mesmo uma combinação de diversas técnicas geralmente é possível, e em muitos casos, é até mesmo recomendável.

4.1. Populações

Diversas abordagens alternativas à maneira tradicional com que a população evolui a cada geração. Analisaremos a seguir algumas das mais conhecidas.

4.1.1. Estado fixo

Um algoritmo genético de estado fixo (*steady-state*) mantém o tamanho de sua população (*popsize*) constante por todo o processo de evolução. A cada geração são criados indivíduos que serão inseridos na população, e, uma vez feito isso, os menos aptos são retirados de maneira a fazer com que esta retorne a seu tamanho original. A taxa de sobreposição desta população é definida por uma constante, que representa o percentual desta substituído a cada geração. Os novos indivíduos são inseridos antes da exclusão para garantir que os mais fracos da totalidade sejam excluídos.

Algumas abordagens utilizam como constante para orientar na substituição o número *n* de indivíduos a serem inseridos ao invés do percentual.

4.1.2. *Incremental*

Segundo [WAL96a], neste modelo de AG um número pequeno de filhos é criado a cada geração, e o esquema de superposição deste pode ser variável. Algumas das formas mais utilizadas são:

a) pais: os pais são substituídos;

b) randômico: os indivíduos a serem substituídos são escolhidos ao acaso;

c) pior: os piores indivíduos da população são substituídos.

4.2. Modelo de Ilhas

O modelo de ilhas é na verdade mais uma forma de comportamento da população de um AG, mas possui um grau tão considerável de importância que deve ser tratado separadamente. Ele tem como objetivo explorar o paralelismo explícito dos algoritmos genéticos visto que o alto grau de independência entre os indivíduos torna facilmente viável induzir sua evolução em paralelo.

Nesta abordagem a população total é dividida em um conjunto de sub-populações ou *ilhas* (normalmente uma sub-população para cada processador disponível) e estas evoluem paralelamente, geralmente fazendo uso de um AG simples ou de estado fixo.A cada geração um número pré-determinado de indivíduos é copiado ou trocado entre as populações, em uma operação conhecida como *migração*. A taxa de migração define este número e deve ter um valor que evite a evolução completamente independente das sub-populações (no caso de a taxa ser igual a zero), onde o problema de convergência prematura tende a ocorrer com maior freqüência, assim como a supressão das diferenças locais entre ilhas. Outro fator a ser considerado no momento de escolha da taxa de migração é o custo que esta operação oferece.

Segundo [COS99], as principais formas de comunicação utilizadas pelo modelo de ilhas são:

a) Comunicação em rede: com a disposição em rede todas as ilhas se encontram inter-conectadas e efetuam um processo de troca de seus melhores indivíduos.

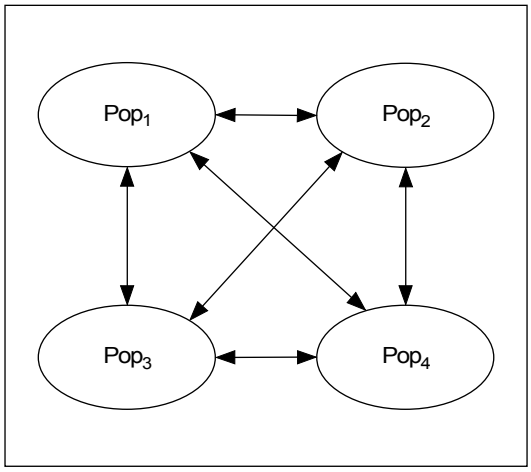

Figura 7: Modelo de ilhas em rede [COS99].

b) Comunicação em estrela: utilizado em [WAL96b], na disposição em estrela uma das ilhas (em geral a que apresenta a melhor média de adaptação após a inicialização) se torna a ilha mestre, a única a se comunicar com todas as outras. As restantes são comumente chamadas de escravas e a cada geração efetuam as operações de envio e recepção de indivíduos ótimos apenas com a mestre.

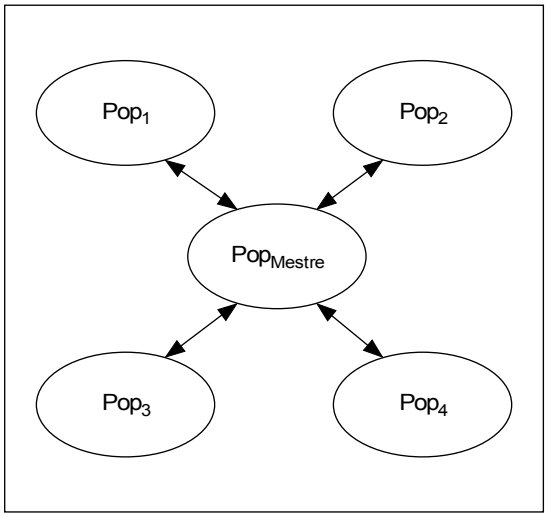

Figura 8: Modelo de ilhas em estrela [COS99].

c) Comunicação em anel: com esta disposição cada ilha envia seus melhores indivíduos para a ilha a seguir e recebe os da anterior, num esquema de comunicação de sentido único.

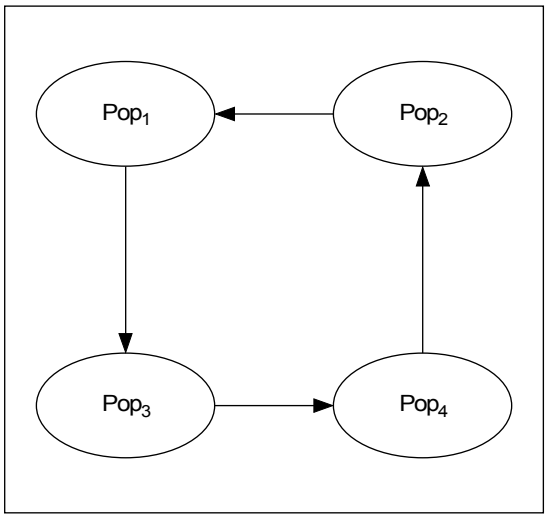

Figura 9: Modelo de ilhas em anel [COS99].

4.3. Representação poliplóide

A representação genética mais utilizada nos AGs é a haplóide, onde o genótipo de cada indivíduo é representado por apenas um cromossoma. Tal escolha se justifica pelo fato de evitar a redundância trazida por representações diplóides e poliplóides, onde o fenótipo relacionado a um certo gene é determinado por uma relação de dominância: uma vez que cada cromossoma a mais oferece um gene extra que codifica a característica desejada e possivelmente um valor (ou alelo) diferente neste, será selecionado aquele considerado dominante em detrimento dos recessivos.

Levando-se em conta que a representação haplóide é usada na natureza apenas para formas de vida mais simples é necessário questionar o porquê do uso de genótipos diplóides ou poliplóides por parte de organismos mais complexos. A justificativa para isto é encontrada no fato de que as relações de dominância entre os alelos não é estática, mas também sujeita à evolução, e que os genes redundantes servem como recipientes de informações que possibilitam uma espécie de "memória" do processo de evolução, reduzindo assim a probabilidade de que determinados alelos que codificam características até o momento pouco desejáveis sejam extintos da população.

4.4. Algoritmos miméticos

Os algoritmos miméticos[*] tem seu funcionamento baseado em uma evolução por pressão social, e não pela tradicional seleção natural[†]. O melhor e o pior indivíduo (o vencedor e o perdedor, respectivamente) localizados durante sua execução são memorizados e servem como modelos para a criação do restante da população. Os operadores de cruzamento e mutação são aqui substituídos por um de mutação social, que torna os indivíduos sobre os quais opera mais próximos ou distantes de seus modelos. Existem diversas estratégias de "comportamento" que definem que atitudes um indivíduo pode assumir com relação a seus modelos (por exemplo: imitar seria

[*] O termo mimético vem do grego *mimetes*, que quer dizer imitador.

[†] é importante ressaltar que o termo natural não está sendo aqui utilizado de maneira *strictu sensu*, isto é, a pressão social é um fato derivado da natureza e, por conseqüência, pode ser vista como um fenômeno natural.

equivalente a tentar copiar o maior número possível de características do modelo, rejeitar equivaleria à tentativa de se distanciar delas e ignorar simplesmente corresponderia a não as levar em conta), entre elas estão, segundo [PEY93]:

a) empreendedor – imita o vencedor e ignora o perdedor;

b) rebanho – imita o vencedor e rejeita o perdedor;

c) fóbico – rejeita o perdedor e ignora o vencedor;

d) ignorante – ignora ambos modelos, serve como referência para checar a relevância dos modelos;

4.5. Híbridos

Os algoritmos genéticos tradicionais geralmente não são as melhores técnicas de busca na maioria dos domínios aplicáveis, devido à ignorância que possuem de como gerar indivíduos melhores a partir da população sobre a qual atuam. Visto que esse método apresenta entretanto uma série significativa de atributos desejáveis, uma considerável carga de pesquisa foi dedicada à resolução deste problema. A idéia por trás dos algoritmos híbridos se resume a explorar em um AG as principais vantagens pertencentes a seus concorrentes. Esta tarefa se dá importando dos últimos o conhecimento específico sobre o domínio do problema que tratam. De acordo com [DAV91], as diretivas mais comuns para hibridização consistem em:

a) basear-se na forma de codificação da técnica concorrente. Desta forma, se faz uso do conhecimento de domínio presente na técnica, assim como se facilita o uso por parte de especialistas no assunto;

b) utilizar o algoritmo concorrente como operador de inicialização – ao se fazer uso do concorrente para gerar a população inicial, garante-se que na pior hipótese o híbrido possuirá um desempenho semelhante a ele. Esta abordagem só é recomendável quando se trata de técnica que forneça respostas em curto espaço de tempo, e é tida como “perfeitamente segura”, visto que no pior caso possível o último apresenta um desempenho semelhante ao primeiro;

c) converter operações aplicadas com freqüência em operadores genéticos – tais operações podem ser adaptadas para funcionar como um operador genético já

conhecido. Por exemplo, considerando-se que a mutação é unária (atua sobre apenas um objeto) e causa variações no cromossoma, uma operação do concorrente que apresente tais características pode ser convertida neste operador. Também é possível gerar um operador com uma nova lógica de funcionamento para substituir ou trabalhar em conjunto com os restantes;

d) utilizar o método de decodificação – se o algoritmo concorrente apresenta um método de decodificação eficiente e este é importante para o funcionamento do AG, a melhor alternativa é incorporá-lo ao híbrido;

4.6. Programação genética

Ao invés de manipular possíveis genótipos de soluções para o problema a ser tratado, a lógica de funcionamento dos algoritmos de programação genética é de manipular representações de programas que o resolvam. Para isto, os indivíduos que manipula são na realidade árvores sintáticas como a da Figura 10 e é tomando um cuidado especial com as alterações feitas nestes, visto que facilmente elas podem gerar programas não funcionais.

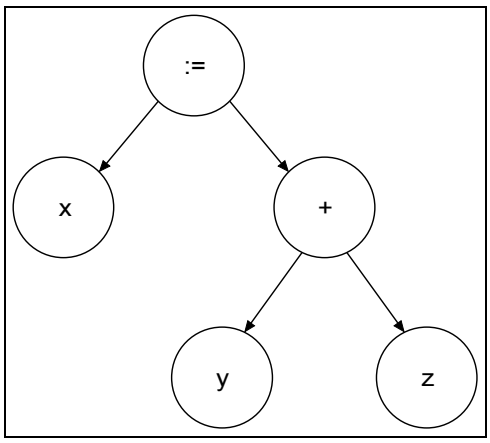

Figura 10: Uma árvore sintática usada como indivíduo de um algoritmo de PG.

Um operador de cruzamento por *swap* de sub-árvores escolhe um nodo de cada um dois pais e gera os filhos como resultado da troca destes dois nós (seus eventuais filhos os acompanham no processo), e um operador de mutação *flip* substitui uma chamada de função por outra de mesmo tipo ou um símbolo terminal por outro do

conjunto válido [MIC99]. Outros exemplos de operadores de mutação e cruzamento para genomas em forma de árvore podem ser encontrados em [WAL96a].

O cálculo da adaptação de um indivíduo se dá através do teste de sua execução: dada uma entrada pré-definida, a saída do programa representado é avaliada, assim como tamanho deste.

5. Aplicações

Por se tratar de uma técnica bastante difundida e geralmente oferecer bons resultados, os AGs são usados em uma ampla série de domínios de problemas. A seguir, ilustramos sua utilização em alguns deles. Mais comentários podem ser encontrados em [HEI95].

5.1. Escalonamento e grade horária

Segundo [SOA97], existem diversas definições conflitantes para o problema de escalonamento (*scheduling*)mas uma definição possível é a de Fox: "*scheduling selects among alternative plans, and assigns resources and times to each activity so that they obey the temporal restrictions of activities and the capacity limitations of a set of shared resources*". De acordo com [DAV91], problemas de *scheduling* apresentam grande dificuldade em seu tratamento pelos seguintes motivos:

a) complexidade computacional: os problemas de escalonamento são da classe NP-Completo, o que quer dizer que a partir de um tamanho razoável dos dados de entrada torna-se impossível explorar todo espaço de busca. Além disso, métodos que façam uso de heurísticas para limitar este espaço não podem garantir que sempre encontrarão uma resposta ótima;

b) conhecimento de domínio: problemas de *scheduling* devem levar em consideração restrições que com freqüência estão fortemente vinculadas a conhecimento específico ao domínio de que fazem parte.

Os problemas de grade horária, ou *timetabling*, (classe à qual pertence o abordado na parte prática deste trabalho) podem ser vistos como um subconjunto dos de *scheduling*.

5.2. Jogos

De acordo com [HEI95], os algoritmos genéticos podem ser relacionados ao método de teoria dos jogos: para isso, geralmente a evolução de sua população consiste de um processo onde indivíduos são selecionados para jogar um jogo com um número limitado de jogadas. Cada jogador interage com um grupo uma série de vezes para

depois jogar com outros oponentes. Dependendo das combinações efetuadas por este em cada jogo será determinado um valor de recompensa, e do somatório destes ao final resultará seu grau de adaptação.

Um indivíduo nesta abordagem codifica a estratégia de um jogador, seja por exemplo representando a probabilidade de escolha de uma jogada A em detrimento de uma B (onde a população poderia ser representada por um conjunto de números reais) ou através de Autômatos Finitos que indicariam a sucessão de passos a serem efetuados durante o jogo.

5.3. Sistemas de classificação

De acordo com [HEI95], uma das aplicações propostas por Holland para os AGs seria seu uso em sistemas de classificação (*classifier system*, CFS), capazes de interpretar mudanças em seu meio ambiente e decidir sobre qual a melhor maneira de se adaptar a elas.

Para isto, é necessário que se defina um meio ambiente e a forma como o CFS interage com ele, ou seja, que funcionalidades serão implementadas para que o último seja informado a respeito de alterações e para que possa efetuar alterações no primeiro.

A lógica de funcionamento de um CFS pode ser definida como "*... start from scratch, i.e., from tabula rasa (without any knowledge) using a randomly generated classifier POPULATION, and let the system learn its program by induction, (cf Holland et al. 1986), this reduces the input stream to recurrent input patterns, that must be repeated over and over again, to enable the animat to classify its current situation/context and react on the goings on appropriately."*[HEI95].

5.4. Uma breve consideração

Os exemplos aqui citados não têm como objetivo apresentar o panorama total de aplicações nas quais os AGs foram ou estão sendo usados com sucesso, pois sua alta popularidade e generalidade fazem deles um método utilizado em um espectro de domínios extremamente abrangente. Optou-se, então, por escolher áreas mais significativas, seja por sua alta importância, complexidade, sucesso no uso da técnica ou potencial para pesquisa.

6. Implementação

6.1. Definição do problema

6.1.1. *Status quo* (locação)

O problema de elaboração de grades horárias para cursos é bastante conhecido no meio acadêmico, se não por estudos a ele dedicados, então por ser vivido a cada semestre. Uma típica resolução manual para tal projeto envolve a composição de alguns horários [COR93][COR94a] que respeitem as regras mais fundamentais, e sua remodelagem conforme neles são descobertos problemas. Este processo envolve muitas vezes a submissão de uma alternativa considerada mais adequada aos indivíduos envolvidos (professores e alunos) para que através do *feedback* proporcionado por estes se encontre uma solução que minimize conflitos. Tal processo efetuado manualmente pode levar vários dias para uma conclusão.

Vários métodos de otimização foram tentados, mas a grande maioria encontra problemas ao lidar com o grande número de restrições e a alta variedade de seu grau de importância, assim como falsos pontos ótimos.

6.1.2. Formalização

O problema de alocação de grade horária em um curso consiste em, para cada hora-aula h pertencente ao conjunto H de *horas-aula* de todas as disciplinas do curso, ministrada por um docente *p* a uma turma t e em uma sala s, escolher o melhor *time slot* [*]s de forma a minimizar problemas como:

a) colisões (*clashes*): por motivos óbvios, o mais importante critério para a avaliação de um horário proposto é que este evite que para um determinado *time slot* s mais de uma disciplina envolvendo o mesmo professor, turma ou sala ocorra;
b) semi colisões (*near-clashes*): por preocupações didáticas, é importante evitar a alocação adjacente de blocos de horas-aula do mesmo gênero para a mesma turma,

[*] um *time slot* representa um determinado horário da semana em que uma hora-aula pode ser alocada, por exemplo, terça-feira às 10h 00min.

ou seja, evitar que para uma turma *t* ocorra, após ministrada uma aula de uma disciplina de um gênero g_d no *time slot s*, outra com mesmo tipo no *slot s*+1;

c) tempo de deslocamento: é preciso considerar o tempo gasto no deslocamento entre duas aulas sucessivas que ocorram em salas distantes, tanto para a turma quanto para o professor envolvido. Um horário que também minimize o número de idas e vindas entre locações distantes para aulas é considerado mais adequado;
d) *slots* inadequados: determinados *slot*s são menos adequados para aulas que outros. Por exemplo, a alternativa de se ministrar aulas no período de 12h 00min às 14h 00min é viável, mas pouco indicada; ou um determinado turno de um dia da semana de um ou mais professores pode estar reservado a atividades que não lhe permitam dar aulas;
e) alocação ótima: a vizinhança em que as disciplinas são alocadas deve ser considerada . Para melhor rendimento das aulas são sugeridos "blocos" de duas horas para cada disciplina.

O sistema mais tradicionalmente utilizado para a avaliação dos indivíduos é o de penalidades, no qual utiliza-se a fórmula:

$$objetivo(x) = \frac{1}{1 + \sum penalidades(x)},$$

onde *objetivo(x)* representa o valor de retorno da função objetivo aplicada ao indivíduo *x*, e *penalidades(x)* as penalidades a ele associadas.

6.1.3. Necessidades fundamentais a suprir

Criar um sistema de auxílio na tomada de decisão – uma vez alimentado com os requisitos fundamentais para escolha dos horários, este gera uma população de soluções final para ser avaliada pelo encarregado.

6.1.4. Expectativa de retorno

Por se tratar de um estudo de viabilidade, não existe a expectativa de criar neste trabalho uma ferramenta completa, cuja entrada seja a totalidade de restrições às quais está sujeito o horário do curso e cuja saída seria uma população de indivíduos ótimos. A justificativa para isso se dá devido ao fato de ser de proporções consideráveis a

quantidade de dados que seriam necessários na entrada e de processamento a ser efetuado sobre eles, assim como a natureza extremamente dinâmica de tais restrições.

6.2. Escolha de ferramenta

Diversas ferramentas para o auxílio na implementação de uma proposta baseada em AGs se encontram disponíveis atualmente, e a própria Internet oferece um considerável repositório destas. Visto que o objetivo deste TCC é uma análise da técnica, optou-se pelo emprego de um software de base para auxílio na avaliação.

A ferramenta de base escolhida para a implementação do aplicativo foi a GALib ([WAL96b]), por apresentar as seguintes características:

a) licença: Livre para aplicações não comerciais, com partes GNU;

b) código: aberto;

c) linguagem: C++;

d) documentação: manual da API em *postscript* ([WAL96a]) e HTML, bastante completa e com vários exemplos;

e) origem: MIT;

f) autor: Matthew Wall;

g) aplicação: biblioteca básica para implementação de funções de AGs;

h) recursos: Orientação a Objetos, exemplos, possibilidade de rodar com PVM, sobreposição populacional;

i) sistemas operacionais: DOS, *Win95*, *WinNT*, *MacOS*, Unix.

6.3. Representação

A escolha da representação dos indivíduos é um ponto crucial para os AGs, ela é o fator mais importante no seu desenvolvimento e posterior desempenho. Ao optar por uma representação é necessário avaliar, segundo [PER96], características como:

a) completude: determina se é possível representar todos os fenótipos possíveis;

b) coerência: indica se a partir do esquema de representação é possível gerar um genótipo que codifique um fenótipo não pertencente ao espectro de soluções do problema;

c) uniformidade: numa representação uniforme o número de genótipos correspondentes deve ser o mesmo para todo fenótipo;

d) simplicidade: representa o grau de complexidade dos atos de codificação e decodificação;

e) localidade: pequenas alterações no genótipo acarretam pequenas alterações em seu fenótipo correspondente;

Considerando-se as características apresentadas, podemos fazer uma avaliação de dois possíveis esquemas de representação:

a) representação de *time slot*s: mais intuitiva, onde cada gene representa um *time slot* (ex.: segunda-feira às 10h 00min) para um determinado semestre. Um genoma seria então uma matriz de duas ou três dimensões (uma ou duas para representar determinado horário na semana e outra representando o semestre). Apresenta a vantagem de não permitir, graças à sua codificação, que duas disciplinas do mesmo semestre ocupem o mesmo horário. Infelizmente, não garante por si só a não ocorrência de colisões de professor e pode facilmente gerar horários com uma alocação errônea do número de horas-aula por disciplina, como mostra a Figura 11:

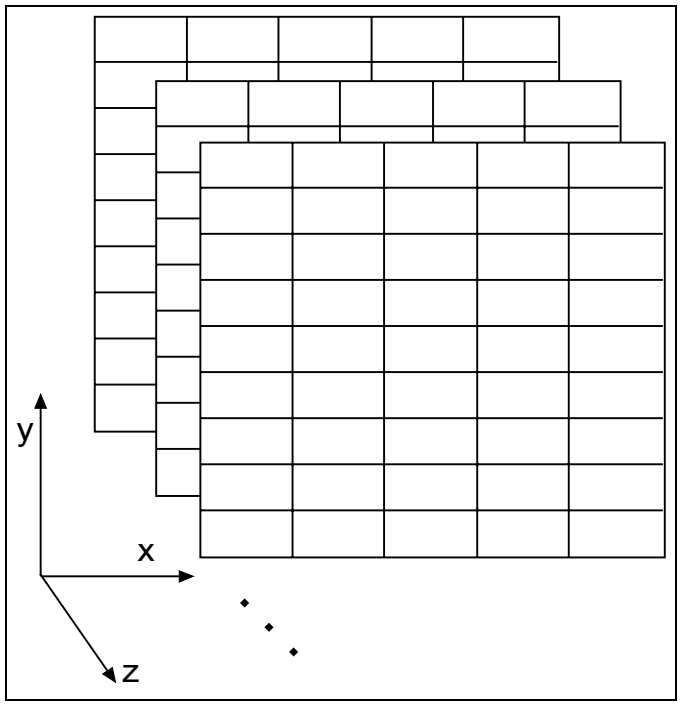

Figura 11: Possível representação de uma grade horária.

Neste exemplo, *x* designa o dia, *y* uma determinada hora deste (ex.: 10h 00min) e z o semestre ao qual pertence a grade horária;

b) representação de horas-aula: neste caso, o genoma representa o conjunto de todas as *horas-aula* de todas as disciplinas. Por motivos de simplicidade, as *horas-aula* da mesma disciplina são alocadas em blocos adjacentes, e por sua vez os blocos são alocados ordenadamente segundo o semestre da disciplina a que pertencem. Apesar de permitir todos os tipos de colisão em sua codificação, esta representação apresenta a vantagem considerável de sempre ter como solução horários que respeitem a carga horária de todas as disciplinas.

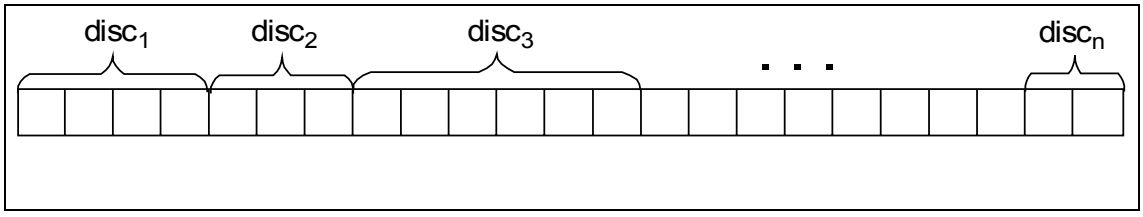

Figura 12: A estrutura escolhida para o genoma.

6.4. Função objetivo

A função objetivo faz uso do cálculo de penalidades associadas a um cromossoma. Na realidade seu funcionamento se baseia em verificar para cada gene quais penalidades estão associadas à sua combinação com todos os demais.

soma_das_penalidades ← 0
para cada $g_i \in$ cromossoma
  //corta iterações desnecessárias visto que graças ao caminhamento
  //sabemos que todo gene já foi testado com seus anteriores
  para cada (g_j posterior a g_i) $\in$ cromossoma
    //impede que uma única infração seja penalizada múltiplas vezes
    se g_j ainda não causou nenhuma penalidade
      então
        soma_das_penalidades ← penalidade(g_i, g_j)
    fim se
  fim para
fim para
// soma-se 1 à soma das penalidades para evitar uma divisão por zero
retorne $\frac{1}{1 + soma\_das\_penalidades}$

Como se pode notar pelo código, o valor possível de retorno da função objetivo se localiza no intervalo] 0 ; 1]

6.5. Estruturas básicas

Em função dos dados fornecidos pelo usuário são geradas as seguintes estruturas:

a) lista de disciplinas: contém toda informação necessária para representar e tratar as disciplinas – nome, código, mneumônico, tipo e carga horária, assim como o SIAPE do professor que a ministra, e o semestre para o qual está programada. A estrutura de um nodo da lista é a seguinte:

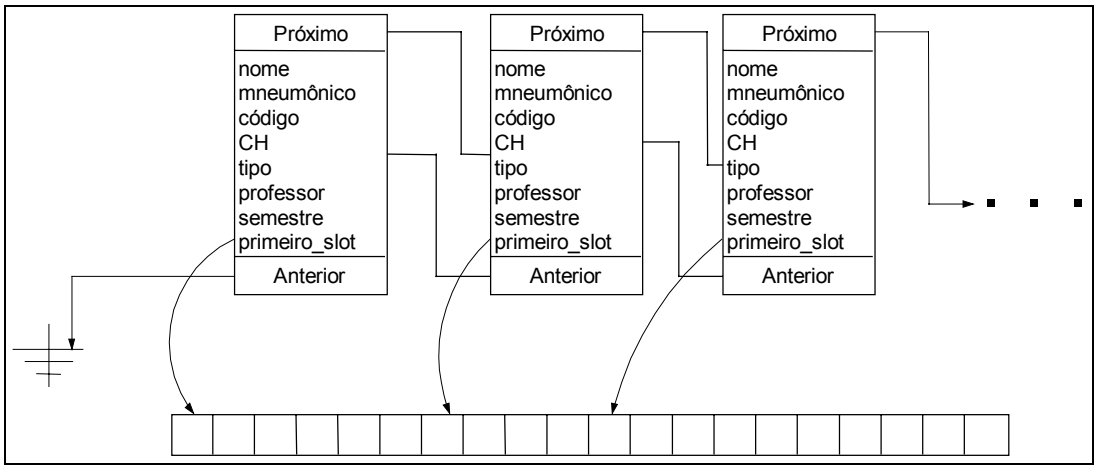

Figura 13: Estrutura da lista de disciplinas.

O campo *primeiro_slot* diz respeito à primeira hora-aula da disciplina alocada no genoma*.

É a partir da criação da lista que se pode calcular o tamanho do genoma, através da fórmula:

$$tamanho(g) = \sum_{i=1}^{ndiscs} ch_i \ ,$$

onde ndiscs é o número de disciplinas envolvidas e ch_i é a carga horária da *i*-ésima disciplina.

b) tabela de penalidades por *slot* (*Slot Penalty List* ou SPList na representação interna do programa): na verdade é um *array* bidimensional de dimensões [$tamanho_g$ X (*ndias x nhoras*)], onde ndias é o número de dias em que podem ser alocadas aulas e *nhoras* o número máximo de horas-aula por dia.

c) tabela de penalidades por colisão (*Clash Penalty List* ou CPList): a partir da informação disponível na lista de disciplinas (semestre e professor), é gerada a lista de penalidades por colisão. Esta lista é representada por um *array* bidimensional de dimensões [tamanho(g) X tamanho(g)].

O preenchimento das células na tabela é feito a seguinte fórmula:

* cabe aqui salientar mais uma vez que as horas-aula de cada disciplina são alocadas contiguamente no genoma.

se $\text{semestre}_i = \text{semestre}_j$

então CPList[*i*, j]← penalidade_por_semestre

se $\text{professor}_i = \text{professor}_j$

então CPList[*i*, j]← penalidade_por_professor

Onde *i* e j são coordenadas de genes no interior do genoma, semestre_n e professor_n são, respectivamente, o semestre e o professor da disciplina correspondente ao *n*-ésimo gene.

A escolha do uso de tabelas deve ser explicada: o cálculo do valor das penalidades associadas a um determinado gene *i* do genoma poderia ser feito sem o uso das primeiras caso as restrições fossem implementadas diretamente no código (*hard-coded*). Tal solução apresentaria um desempenho semelhante no que diz respeito à velocidade e uma melhora considerável no quesito uso de memória (o tamanho da tabela de penalidades, por exemplo, cresce em uma ordem n^2 conforme o número total de horas-aula). Sua deficiência, contudo, é a perda do grau de dinamismo no uso das restrições – seria provavelmente necessário alterar um volume significativo de código em caso de mudança de requisitos, assim como se dificultaria o tratamento de alguns tipos de exceção. Ao se fazer uso das estruturas tabelares é possível, por exemplo, apresentá-las ao usuário após sua criação e antes da execução do AG propriamente dita, para que este possa revisar e/ou modificá-las, assim como criar mais facilmente sub-rotinas específicas que configurem valores para determinados tipos de exceção.

6.6. Penalidades

Para o uso por parte da função objetivo, os tipos de penalidade foram agrupados em classes conforme seu grau de importância. É necessário ressaltar que este grupamento é flexível: os valores *default* de cada um deles pode ser alterado mesmo durante a execução do programa. A escolha dos valores atribuídos para cada grau foi baseada apenas em bom senso, como defendido por [COR93] e [COR94a]. Inferiu-se que determinados tipos de restrições deveriam possuir um caráter de exclusão: indivíduos que infrinjam as de nível altíssimo deveriam receber uma penalidade tão grande que sua probabilidade de seleção se tornaria ínfima.

Tabela 1: Tipos de restrição e seu grau de importância

critério	Função	grau
semestre	Impedir colisão entre disciplinas do mesmo semestre	altíssimo
professor	Impedir que disciplinas ministradas pelo mesmo docente sejam alocadas no mesmo horário	altíssimo
deslocamento *	Considerar o respeito ao tempo de demora no deslocamento de um prédio a outro (ex: prédios localizados em dois campi distantes).	altíssimo
sala *	Respeitar o limite de uma turma por sala durante as aulas.	altíssimo
slots tabu	Evitar alocar aulas em horários pouco adequados (ex: das 12h 00min às 14h 00min)	médio
bloco	Dispor as horas-aula das disciplinas em blocos de tamanho que facilitem no estudo. O tamanho tipicamente escolhido para um bom bloco é de duas horas-aula.	médio
dia	Evitar a alocação de mais de um bloco de aulas de uma disciplina no mesmo dia	médio
tipo	Inibir que duas disciplinas do mesmo tipo sejam alocadas em sucessão para a mesma turma.	baixo
slots ótimos *	Tentar alocar na primeira hora da tarde ou na segunda da manhã	baixo
evitar horários esparsos *	Tentar evitar que turnos não tenham a totalidade de seus *time-slots* alocados	baixo

Na Tabela 1 são sugeridas restrições (marcadas com *) que por motivos de viabilidade não foram implementadas. Elas se encontram aqui como uma forma de ilustração de outros fatores que poderiam ser considerados no momento de confecção de uma grade horária e não foram tratadas no aplicativo por não serem consideradas absolutamente fundamentais (como por exemplo a restrição de alocar na primeira hora da tarde ou na segunda da manhã), ou, como no caso da importante colisão de sala, apresentarem um considerável complicador no que diz respeito à dificuldade de se obter os dados de entrada necessários e por ilustrar um conceito (a alta necessidade de evitar colisões) abordado em outros critérios.

6.7. Operadores Genéticos

6.7.1. Operadores GALib

Operadores da GALib – Os operadores genéticos presentes em [WAL96b] já tiveram sua base explicada na seção 2.4: Reprodução. Limitamo-nos aqui a uma avaliação dos resultados constatados com o uso de alguns deles:

a) cruzamento por um ponto (OPX): Apresentou, contrariamente ao normal nas aplicações, o melhor desempenho dentre os operadores fornecidos pela GALib. A justificativa para este fato é que por apresentar apenas um ponto de corte, este operador tende a conservar esquemas importantes com maior facilidade;
b) cruzamento multipontos(MPX): Com um desempenho pouco inferior ao cruzamento por um ponto, os testes realizados confirmam que, quanto menor o número de pontos de corte empregados na operação, melhores os resultados;
c) cruzamento uniforme(UX): apresentou péssimos resultados, até 1000% inferiores ao cruzamento por um ponto;

6.7.2. Operadores criados de inicialização

a) *Uniform* r*andom initializer* (inicializador aleatório simples): Esta sub-rotina atribui um valor aleatório para cada gene do indivíduo. Tem como vantagem a facilidade de implementação, o alto grau de biodiversidade da população que gera e o baixo custo de seu desempenho (neste caso pouco significativo para os tamanhos de população e indivíduos testados neste trabalho). Sua principal desvantagem é a de gerar com uma probabilidade ínfima indivíduos que satisfaçam as restrições por *time slot* e alocação ótima;

b) *heuristic slot initializer* (inicializador heurístico por *slot*): Faz uso de uma função aleatória com distribuição não uniforme, que sorteia *time slots* indesejáveis com uma probabilidade menor. Seu custo operacional é praticamente igual ao do inicializador randômico simples e possui sobre este a vantagem de raramente alocar

horas-aula em *time* slots tabu[*]. Testes comparativos revelaram uma melhora de 100% no desempenho em relação a seu antecessor. Não se preocupa com uma alocação racional de blocos;

c) *heuristic discipline initializer* (inicializador heurístico por disciplina): faz uso da mesma função de sorteio de *time* slots do inicializador de slots, mas aloca preferencialmente as disciplinas em blocos que respeitem as restrições de alocação ótima. Seu desempenho é comparável dos operadores anteriores, porém proporciona uma melhora considerável na performance do AG como um todo.

6.7.3. Operadores de cruzamento criados

a) *Multi point block crossover* (MBX): O cruzamento por blocos e em múltiplos pontos é um operador de cruzamento multiponto que efetua o corte dos cromossomas pais de forma a não partir um bloco de uma mesma disciplina.

6.7.4. Operadores de Mutação criados

a) *Swap Block Mutation* (SWM): Semelhante ao SM, faz a troca de blocos entre os pares de disciplinas que sorteia. Para executar sua tarefa, precisa extrair do genoma uma representação em blocos das disciplinas escolhidas para o *swapping* e uma vez feito isso escolher quais os melhores blocos para a troca. Este operador apresentou uma melhora de 300% em relação aos operadores de mutação tradicionais, e ajuda a manter a biodiversidade, sem contudo apresentar o mesmo efeito negativo que seus antecessores sobre a convergência.

6.8. Resultados

6.8.1. Dificuldades encontradas

a) Em um problema de *timetabling* com a representação escolhida, bons esquemas tendem a possuir um grande comprimento, pois se de um lado é comum que os

[*] O termo tabu aqui utilizado é emprestado de outra técnica de otimização conhecida: a busca tabu. Um indivíduo ou valor tabu é possível mas inadequado, indesejável.

valores ao redor de um gene sejam importantes para a qualidade final do cromossoma, visto que é isto que determina de que forma as disciplinas foram alocadas em blocos, por outro é muito comum que genes tenham grande importância um para o outro apesar de fisicamente[*] distantes, como no caso de genes pertencentes a disciplinas de semestres diferentes mas de mesmo professor.

Visto que o problema é altamente combinatório, ou seja, que o valor de um determinado gene tende a possuir uma considerável importância para os demais, podemos também afirmar que o grau de um bom esquema geralmente é bastante elevado.

Considerando-se estas duas características se torna claro que operadores de recombinação mal direcionados podem gerar com alta probabilidade filhos menos adequados que os pais. Dados o alto número de restrições a que está sujeito um horário e a maneira cega como operam o cruzamento e a mutação tradicionais, o AG baseado no uso destes apresentou um desempenho muito pouco satisfatório, principalmente no que diz respeito ao requisito de grupamento ótimo em blocos. Apesar de este requisito poder ser otimizado por esquemas de pequeno comprimento, graças à alocação contígua das horas-aula da mesma disciplina, ele dificilmente é respeitado com o uso dos operadores tradicionais, visto que a probabilidade de que o AG aloque as disciplinas em blocos ótimos é bastante pequena;

b) convergência prematura – o problema de convergência prematura tende a ocorrer com facilidade em AGs para *timetabling*. Isto se dá pelo fato de que no momento em que se restringe a maneira como os operadores genéticos alteram os indivíduos, também se restringe a biodiversidade da população à qual eles pertencem. Se aliado a isto tivermos o aparecimento de superindivíduos (fenômeno bastante típico) e uma política inadequada de seleção e *scaling* é possível observar que durante a evolução um grupo reduzido de cromossomas tende a dominar e com isso diminuir o escopo da busca.

[*] o termo fisicamente aqui se refere à disposição dos genes no genoma.

Em termos de características da população, o que se observa é um aumento gradual nas médias conforme esta evolui acompanhado de uma queda na diversidade e globalmente no ritmo de adaptação dos melhores indivíduos.

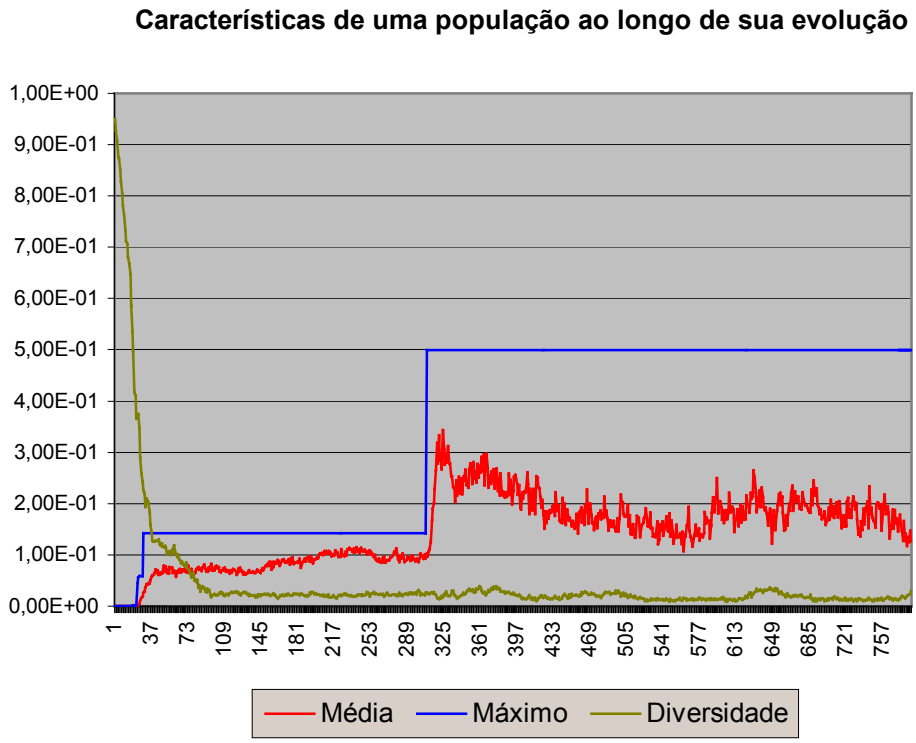

Figura 14: Gráfico de evolução de uma população.

Conforme já descrito, a solução tradicional para este problema é um uso racional do operador de mutação: é necessário que este consiga manter um nível adequado de biodiversidade sem incorrer com grande freqüência em outro problema já citado, o de epístase. Os operadores tradicionais de mutação deram uma contribuição bastante limitada a estes dois critérios, pois um uso mais freqüente destes (necessário para garantia da diversidade) tende a baixar o grau de adaptação da população.

c) alto desvio no resultado das execuções – A importância do elemento aleatório na evolução de um algoritmo genético se faz notar por um fato negativo: os resultados finais de execuções diferentes com os mesmos dados de entrada podem apresentar um desvio extremamente significativo, dependendo do problema tratado. Isto

significa que, para determinadas classes de problemas (da qual o de grade horária faz parte), execuções ótimas e medíocres tendem a aparecer intercaladas, e implica em uma dificuldade considerável de previsão de comportamento do algoritmo, fundamental para sua depuração e *benchmarking*.

No protótipo apresentado neste trabalho, variações de 1000% chegaram a ser registradas, fenômeno que requer uma investigação mais profunda do grau de importância da base aleatória para a evolução do AG: através de uma análise dos piores resultados obtidos, é possível observar que seus melhores indivíduos não são ótimos por violarem um pequeno número de restrições de prioridade média ou baixa ligadas à maneira como as horas-aula são alocadas em blocos.

Duas abordagens para a resolução deste problema seriam, primeiramente, limitar a maneira como são gerados os indivíduos iniciais, para reduzir a probabilidade de que apresentem tais características indesejáveis, através da criação de uma nova função heurística de sorteio de *time-slots* para cada bloco de horas-aula, ou, em segundo lugar, fazer uso de uma heurística de escolha que não o simples sorteio uniforme para os pontos do genoma sobre os quais deve atuar o *Swap Block Mutator*, visto que este tende a eliminar o problema quando aplicado a genes que violem estas restrições.

6.8.2. Análise do grau de sucesso

O aplicativo *Timetabler* desenvolvido neste TCC ainda é um protótipo: ele não leva em consideração todas as restrições desejáveis para o problema e não apresenta tampouco uma interface adequada para um possível usuário final (para mais informações a respeito, veja 6.8.3:Perspectiva). O objetivo no momento de sua criação era puramente experimental: através da abordagem de um problema do mundo real pelo método de algoritmos genéticos obter uma visão mais prática e profunda de seu funcionamento.

Pelo alto grau de complexidade do problema tratado e conseqüentes dificuldades encontradas, tornou-se possível não só analisar algumas das características intrínsecas aos AGs como também lidar com os problemas mais conhecidos que podem ocorrer em sua execução e experimentar diversas abordagens para resolvê-los.

6.8.3. Perspectiva

Acreditamos que a demonstração da viabilidade de tratamento do problema de grade horária através do aplicativo desenvolvido incentive a transformação deste último em uma ferramenta completa voltada para o usuário final. Algumas das características desejáveis para ele seriam:

a) Uso de uma base de dados: considerando-se que o aplicativo opera sobre dados a respeito das disciplinas de um curso e que nos dias de hoje tais informações estão geralmente localizadas em bases de dados, o processo se tornaria mais prático se o acesso fosse feito diretamente a estas.

b) Uso de restrições complementares: restrições como as de sala de aula e tempo de deslocamento são importantes quando consideramos uma grade horária de um curso. O aplicativo *Timetabler* deveria então ser expandido para tratá-las.

c) Auxílio no processo de matrícula: a mentalidade de uso do aplicativo poderia ser alterada, fazendo com que ele recebesse como dados as intenções de matrícula de todos alunos do curso e a partir destes atribuísse como penalidade pela colisão entre duas disciplinas um valor proporcional ao número de alunos matriculados nas duas[*]:

$$penalidade(x, y) = n\_alunos(x, y) * penalidade\_por\_aluno$$

O aplicativo teria como saída os horários sugeridos e uma tabela de sugestão de matrícula para todos os alunos.

[*] esta forma de cálculo de penalidade de colisão por alunos suprimiria o uso da restrição de penalidade por semestre.

7. Conclusão

7.1. Potencial

Processo inteligente – os AGs requerem do programador informações sobre como avaliar seus indivíduos, isto é, quão boa é a resposta que este representa, mas não exige informações sobre como deve proceder para encontrar pontos ótimos. Esta independência pode ser encarada como um atributo que confere maior inteligência ao processo

7.2. Viabilidade

Os AGs clássicos apresentam um desempenho em muitos casos pouco satisfatório uma vez comparados com outros algoritmos de busca. Para resolver tal problema, o uso de heurísticas nos operadores genéticos tem sido a solução mais universalmente adotada. Tal uso não deve ser indiscriminado: ao implementar funções dependentes de domínio, aproximamos os AGs de seus concorrentes, mesmo no que diz respeito a desvantagens, e em casos mais extremos vamos de encontro à sua própria filosofia de funcionamento.

Um mau uso de heurísticas pode, ainda, suprimir a biodiversidade da população e causar problemas como o da convergência prematura.

7.3. Últimas considerações

Existem bons argumentos para justificar a popularidade alcançada pelos algoritmos genéticos na comunidade científica: sua robustez, adaptabilidade, generalidade e alto grau de paralelismo, entre outras qualidades, fazem deles uma escolha natural quando se deseja implementar uma aplicação de busca e otimização. Os problemas mais significativos que lhe são associados, e.g., o da convergência prematura, têm sido objeto de um número expressivo de estudos, que comumente demonstram abordagens para sua resolução.

A mais forte crítica que se pode fazer aos AGs diz respeito ao seu desempenho ou à necessidade de descaracterização da técnica para resolver este ponto fraco. Os

resultados demonstram que híbridos bem implementados apresentam uma performance que na maioria dos casos nada deixa a desejar com relação às suas alternativas concorrentes, normalmente com o já discutido sacrifício da generalidade, trazido pelo uso de conhecimento específico da área de problema abordado para os operadores genéticos.

Uma saída adequada para o dilema que a hibridação impõe consiste no puro uso de bom senso para equilibrar todas as considerações que ele envolve. Alternativamente, resta o uso de AGs muito semelhantes ao canônico, também chamados de *puros*, em abordagens onde os potenciais de adaptação e generalidade sejam os fatores mais importantes, como é o caso dos CFS e outros sistemas pertencentes às áreas de aprendizado de máquina ou de vida artificial. Através da análise realizada ao longo deste trabalho é possível concluir que o ponto mais forte da técnica de algoritmos genéticos reside em simular a natureza em de seus aspectos mais importantes: o princípio de que toda robustez e diversidade que apresenta são conseqüência direta do dinamismo existente em sua organização interna.

Bibliografia

[BAR96] BARBOSA, Hélio J. C.. Algoritmos Genéticos para Otimização em Engenharia: Uma Introdução. In: IV Seminário sobre Elementos Finitos e Métodos Numéricos em Engenharia. **Anais...** Juiz de Fora, MG, 1996.

[BUR94] BURKE, Edmund; ELLIMAN, David; WEARE, Rupert. A Genetic Algorithm Based University Timetabling System. In: 2nd East-West International Conference on Computer Technologies in Education (Crimea, Ukraine, 19th-23rd Sept 1994), vol 1, pp 35-40. **Proceedings.** 1994. Disponível via WWW em http://www.asap.cs.nott.ac.uk/ASAP/papers/pdf/crimea94.pdf (29 de Agosto de 2000).

[BUR95] BURKE, Edmund; ELLIMAN, David; WEARE, Rupert. A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. In: 6th International Conference on Genetic Algorithms (ICGA'95, Pittsburgh, USA, 15th-19th July 1995), pp 605-610. **Proceedings...** Morgan Kaufmann, San Francisco, CA, USA, 1995. Disponível via WWW em http://www.asap.cs.nott.ac.uk/ASAP/papers/pdf/icga95.pdf. (29 de Agosto de 2000).

[COR93] CORNE, Dave; FANG, Hsiao-Lan; MELLISH, Chris. Solving the Modular Exam Scheduling Problem with Genetic Algorithms. In Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. **Proceedings...** Gordon and Breach Science Publishers, 1993**.** Disponível via FTP anônimo em ftp://ftp.dai.ed.ac.uk/pub/user/ga/93-001.ps.Z. (29 de Agosto de 2000).

[COR94a] CORNE, Dave; FANG, Hsiao-Lan; ROSS, Peter. Fast Practical Evolutionary Timetabling. In: AISB Workshop 1994. **Proceedings...** Springer Verlag Lecture Notes in Computer Science , 1994. Disponível via FTP anônimo em ftp://ftp.dai.ed.ac.uk/pub/user/ga/94-004.ps.Z. (29 de Agosto de 2000).

[COR94b] CORNE, Dave; FANG, Hsiao-Lan; ROSS, Peter. Evolutionary Timetabling: Practice, Prospects and Work in Progress. In: UK

Planning and Scheduling SIG Workshop, Strathclyde, September 1994. Disponível via FTP anônimo em ftp://ftp.dai.ed.ac.uk/pub/user/ga/94-005.ps.Z (29 de Agosto de 2000).

[COS99] COSTA Jr, Ilaim. Introdução aos Algoritmos Genéticos. In: VII Escola de Informática da SBC Regional Sul. **Anais...** SBC, 1999.

[DAV91] DAVIS, L. D. **Handbook of Genetic Algorithms.** Van Nostrand Reinhold, 1991.

[GOL89] GOLDBERG, David E. **Genetic Algorithms in Search, Optimization & Machine Learning.** Addison-Wesley, 1989.

[GEY97] GEYER-SCHULTZ, Andréas. **Fuzzy Rule-Based Expert Systems and Genetic Machine Learning**. Heidelberg : Physica-Verlag, 1997.

[MIC99] Michalewicz, Zbigniew. **Genetic algorithms + data structures = evolution programs.** Berlin : Springer-Verlag, 1999.

[HEI95] HEITKOETTER, Joerg; . BEASLEY, David. **The Hitch-Hiker's Guide to Evolutionary Computation.** Disponível no newsgroup: comp.ai.genetic. (23 de Agosto de 2000)

[PER96] PÉREZ SERRADA, Anselmo. **Una introducción a la Computación Evolutiva.** Disponível via WWW em http://www.geocities.com/igoryepes/spanish.zip. (16 de Setembro de 2000).

[PEY93] PEYRAL, Mathieu; DUCOULOMBIER, Antoine; RAVISE, Caroline; SCHOENAUER, Marc; SEBAG, Michèle. Mimetic Evolution. In: International Conference on Genetic Algorithms, 17-21 de Julho de 1993. **Proceedings…** San Mateo : Morgan Kaufmann, 1993.

[SOA97] SOARES, Pedro; MAMEDE, Nuno J.. Timetabling Using Demand Profiles. In: 8th Portuguese Conference in Artificial Intelligence, EPIA-97, Coimbra, Portugal. **Proceedings...** Springer - Verlag Berlin Heidelberg.

[WHI00] WHITLEY, Darrel. **A Genetic Algorithm Tutorial.** Disponível via WWW em http://www.geocities.com/igoryepes/ga_tutorial.zip. (16 de Setembro de 2000).

[WAL96a] WALL, Mathew. **GALib: A C++ Library of Genetic Algorithm Components**. Manual disponível por WWW em http://lancet.mit.edu/ga/ (7 de Agosto de 2000).

[WAL96b] WALL, Mathew. **GALib 2.4**. Disponível por WWW em http://lancet.mit.edu/ga/ (7 de Agosto de 2000).

Anexo 1: Glossário

É importante ressaltar que este glossário é voltado para a área de algoritmos genéticos e por isso faz uso de um considerável grau de liberdade na definição de termos com relação a seus correlatos na biologia.

Adaptação – no campo da biologia, o nível de adaptação mede o quanto um indivíduo esta apto para sobreviver no meio em que reside. Em sua analogia com a natureza, os AGs fazem uso do grau da adaptação para representar quão bem um determinado indivíduo (solução) responde ao problema proposto.

Alelo – os alelos de um gene correspondem ao conjunto de valores que ele pode assumir. Normalmente, o conjunto de alelos é o mesmo para todos genes de um genoma.

Alfabeto – um alfabeto consiste em um conjunto de símbolos que são manipulados por uma linguagem. Para a quase totalidade dos problemas tratados pelos algoritmos genéticos é possível definir uma linguagem formal que gere a partir de um alfabeto e uma série de regra pré-definidos todo o conjunto de soluções possíveis para o problema. Um alfabeto, para os AGs é, então, o conjunto união de todos os alelos possíveis para os genes pertencentes ao genoma.

Convergência – o grau de convergência é característica das populações dos AGs, e diz respeito à diferença entre a média de adaptação de sua geração atual e suas anteriores. A ascensão deste índice indica que o processo de evolução está efetivamente promovendo a melhora da média de adaptação da população, e sua estabilização em torno de um mesmo valor por muitas gerações normalmente indica que a população se estacionou em um determinado valor médio de adaptação, caso em que a continuação do processo de evolução se torna improdutiva.

Convergência Prematura – a convergência prematura é um problema bastante recorrente na técnica de algoritmos genéticos: ela ocorre quando a diversidade de uma população cai de tal forma que o processo de reprodução gera a cada geração filhos muito semelhantes aos pais, o que causa a convergência da população em

com média de adaptação em muitos casos pouco adequada e retarda (ou até mesmo estanca por completo) a evolução.

Cromossoma – um cromossoma consiste de uma cadeia de genes. Visto que tradicionalmente são utilizados genomas com uma cadeia simples de genes, muitas vezes este termo é utilizado, com certa liberdade, como sinônimo para genoma.

Cruzamento – durante o processo de cruzamento os indivíduos previamente selecionados são cruzados com seus pares para gerar filhos.

Diversidade (ou biodiversidade) – característica de uma população de indivíduos, a diversidade diz respeito ao grau de semelhança entre eles.

Esquema – os esquemas servem para definir e representar conjuntos de genomas. Para isto, faz uso do símbolo * para ilustrar situações em que o valor de determinado gene não importa. Por exemplo o esquema {*,*,*,1} representa o conjunto de todos cromossomas que possuem 1 como valor em seu gene de índice 4.

Evolução – o processo de evolução é o responsável pela busca efetuada pelos AGs. A cada passo os indivíduos da população a ele submetida passam pelos processos de seleção, reprodução e mutação, criando assim uma nova geração a partir de sua anterior.

Função objetivo – a função objetivo recebe como entrada um indivíduo e retorna o grau de adaptação que este apresenta.

Fenótipo – resultado do processo de decodificação do genótipo de um indivíduo, em um AG o fenótipo é a solução propriamente dita que este representa.

Gene – um gene serve como um *container*, um espaço para a alocação de um valor. As características dos indivíduos são definidas a partir dos valores contidos no conjunto de um ou mais genes que as codificam.

Genótipo – para um algoritmo genético o genótipo de um indivíduo é a informação codificada em um genoma (a estrutura que é manipulada durante o processo de evolução) e o fenótipo resultante do processo de decodificação é a solução que ele representa.

Genoma – um genoma é o conjunto de genes de um determinado indivíduo. Tradicionalmente genomas são formador por uma cadeia única de genes (um cromossoma), mas existem representações alternativas como a de árvores (bastante utilizada em problemas de programação genética) de listas e de matrizes multidimensionais.

Geração – a cada passo do processo de evolução uma nova geração é criada a partir da população anterior, e esta última é atualizada. Para que o processo de evolução possa ser considerado eficiente é necessário que em cada nova geração tenda a ser melhor (mais adaptada) que suas anteriores.

Indivíduo – um indivíduo pertencente a um algoritmo genético representa uma possível solução para o problema a ser tratado.

Lócus (ou índice) – o lócus de um gene determina a posição que ele ocupa no genoma. É necessário então uma escolha de determinação do lócus que não permita ambigüidades.

Mutação – na natureza, falhas no processo de reprodução podem facilmente ocorrer, fazendo com que os filhos apresentem características que não seriam atingíveis através de uma combinação de características dos pais. A este fenômeno dá-se o nome de mutação, e nos algoritmos genéticos esta transformação ocorre sobre indivíduos provindos do processo de reprodução.

População – uma população consiste em um conjunto de indivíduos. Ela apresenta características não observáveis nos indivíduos, tais como grau de diversidade e de convergência.

Reprodução – o processo de reprodução consiste da alocação em pares (ou em algum método alternativo de *mating*) dos indivíduos selecionados e do cruzamento entre estes. Ele é o responsável pela perpetuação de características ao longo do processo de evolução.

Scaling – é sabido que a presença de super indivíduos e a baixa diversidade numa população são fatores que facilmente levam ao problema de convergência prematura. Vários métodos são utilizados para diminuir a probabilidade de que isto ocorra, entre eles os de *scaling*. Estes últimos operam aplicando uma

transformação ao grau de adaptação de cada indivíduo da população, atenuando as discrepâncias nestes valores, e valorizando indivíduos diferentes da média.

Seleção – nesta etapa onde indivíduos são escolhidos para a reprodução de acordo com seu grau de adaptação. Para isto cada indivíduo é avaliado e os mais aptos da população possuem maior probabilidade de serem selecionados.

Super-indivíduo – um determinado genoma é considerado um super-indivíduo quando ele apresenta um grau de adaptação significativamente superior a média. Por possuir então uma probabilidade muito maior de ser selecionado para reprodução que seus concorrentes, suas características tendem a proliferar pelas gerações subseqüentes, freqüentemente causando queda na diversidade.

Anexo 2: Tabelas e Gráficos de Comparação

Os dados das tabelas e gráficos a seguir foram obtidos a partir de execuções do aplicativo *Timetabler*. Na verdade, por terem sido efetuados em momentos diferentes da fase de implementação, o que mais importa nesses dados não é o valor em si (pois em alguns casos condições melhores para a evolução poderiam ter sido fornecidas), mas a relação ou comparação deste com os demais da mesma tabela.

Para obter estes valores calculamos a média resultante de no mínimo 10 execuções com um número de gerações = 1000 e tamanho da população = 50. Obviamente, o único parâmetro alterado para cada entrada da tabela é aquele a que ela corresponde.

É importante ressaltar ainda que tal medida tem fins meramente ilustrativos, pois a experiência demonstrou que o considerável grau de imprevisibilidade do comportamento dos AGs pode gerar amostragens pouco significativas, que acarretam em execuções aberrantes e tem como efeito um impacto considerável no cálculo das médias.

Uma explicação da estrutura das tabelas:

a) adaptação – é o valor da média da adaptação do critério escolhido, ou seja, o valor médio da adaptação dos melhores indivíduos (média dos melhores – adaptação) encontrados a cada execução ou da média global de cada população final (média das médias – adaptação);

b) somatório – a partir do valor de adaptação faz-se um cálculo inverso ao final da função objetivo para se descobrir a média do somatório de penalidades atribuídas aos melhores indivíduos (média dos melhores – somatório) ou da média de toda população final (média das médias – somatório);

c) variação – indica quão melhor o indivíduo é do que a média da população. Apesar de importante não pode ser considerada representativa no que diz respeito à diversidade da população.

7.4. *Scaling*

Tabela 2: Comparação de desempenho dos métodos de scaling

método de escala	média das médias		média dos melhores		variação
	adaptação	somatório	adaptação	somatório	
Linear	0,048063183	19,80594628	0,194201708	4,149285304	404%
Sigma	0,043671146	21,89841441	0,119802368	7,347080439	274%
Power Law	0,061238419	15,32961826	0,090956461	9,994271152	149%
Sharing	0,031963977	30,28521814	0,051622593	18,37136308	162%
Nenhum	0,03394388	28,46039203	0,053641415	17,64231223	158%

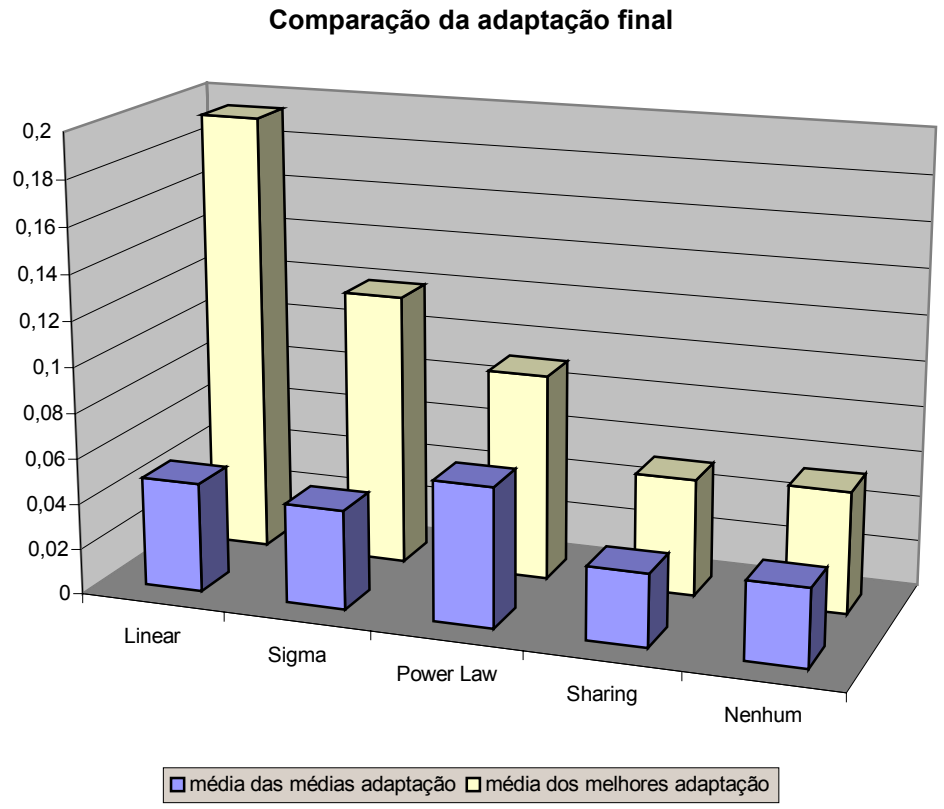

Figura 15: Gráfico de comparação dos métodos de escala.

7.5. Seleção

Tabela 3: Comparação do desempenho dos métodos de seleção.

método de seleção	média das médias		média dos melhores		variação
	adaptação	penalidades	adaptação	penalidades	
Ranking	0,00082429	1212,15794	0,00863124	114,858163	1047%
Torneio	0,04967945	19,1290476	0,15260968	5,55266418	307%
Roleta	0,00776611	127,764562	0,0257482	37,8376675	332%
DS	0,00106557	937,467197	0,00938218	105,584993	880%
SRS	0,01824066	53,8225764	0,02713538	35,8522637	149%
Uniforme	0,00119142	838,33299	0,01119591	88,3183078	940%

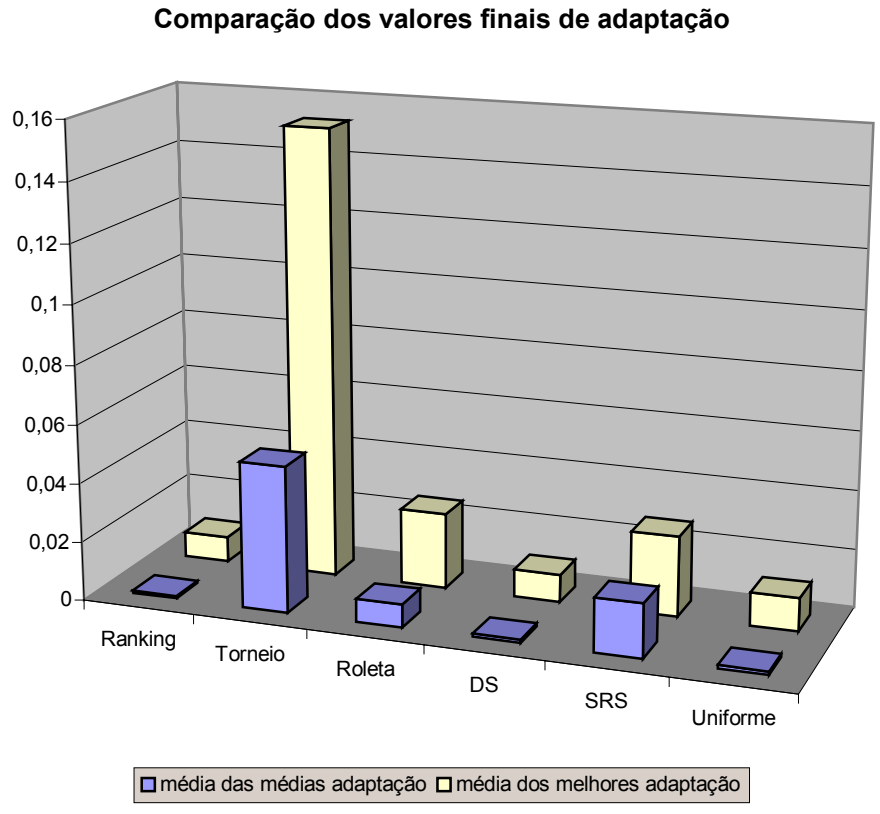

Figura 16: Gráfico de comparação dos métodos de seleção.

7.6. Operadores de cruzamento

Tabela 4: Comparação do desempenho dos operadores de cruzamento.

operador de cruzamento	média das médias		média dos melhores		variação
	adaptação	penalidades	adaptação	penalidades	
1PX	0,01889756	51,9168964	0,02423089	40,2696356	128%
2PX	0,02626158	37,0784401	0,02989421	32,4512983	114%
UX	0,01237979	79,7768382	0,0133788	73,745094	108%
PMX	0,00287945	346,288969	0,00341815	291,555932	119%
MBX (1)	0,03399378	28,4171432	0,04506252	21,1913919	133%
MBX (4)	0,05324522	17,7810277	0,169986	4,88283749	319%
MBX (6)	0,05772811	16,3225829	0,18053447	4,5391084	313%
MBX (8)	0,05601451	16,8525159	0,17933257	4,57623203	320%
MBX (10)	0,03758744	25,6046339	0,07530481	12,2793637	200%

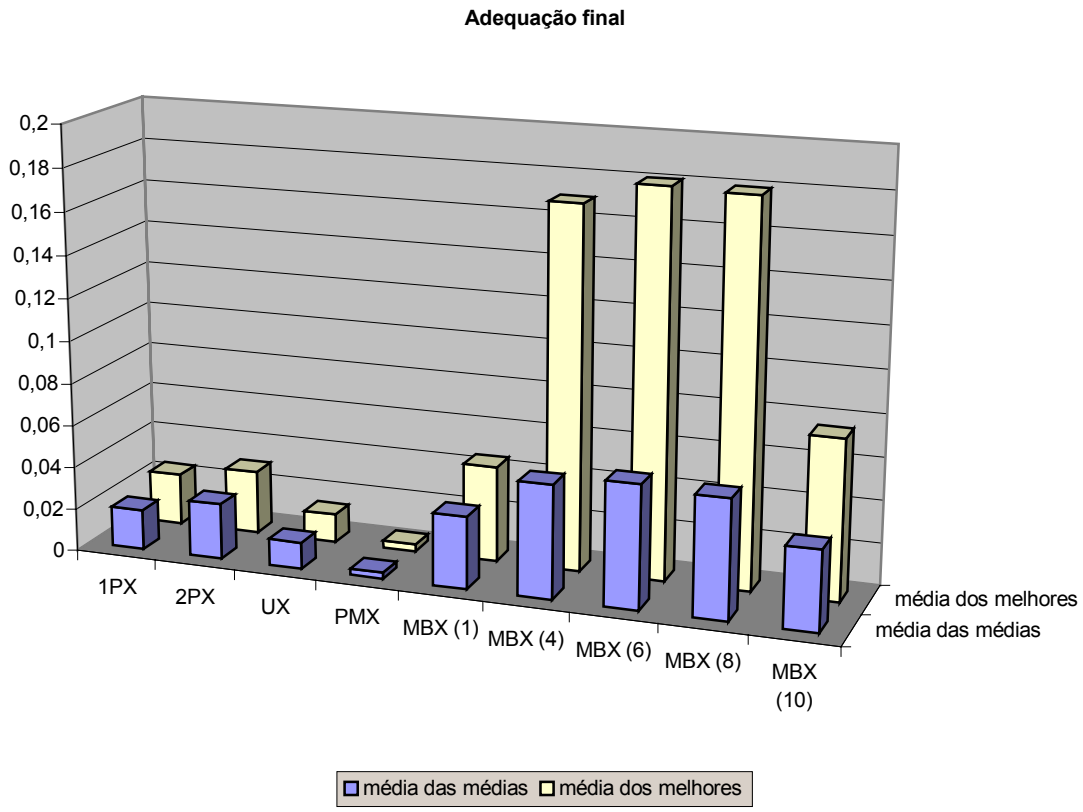

Figura 17: Gráfico de comparação entre os operadores de cruzamento