Representação e Operadores Evolutivos

1 Introdução

- Até o momento foram estudados os principais algoritmos evolutivos: algoritmos genéticos, estratégias evolutivas, programação evolutiva, e programação genética.
- Verificou-se que, embora todos eles sigam praticamente o mesmo algoritmo evolutivo padrão (básico), existem algumas diferenças entre eles. (ver tabelas na página 9 - Tópico 8, e página 28, Tópico 11.)
- Este tópico tem por objetivo revisar as principais estruturas de dados utilizadas como representação em algoritmos evolutivos, assim como os diversos tipos de operadores genéticos e mecanismos de seleção empregados em computação evolutiva.

2 Representações

- Cada método de busca manipula soluções candidatas que representam uma instância do problema a ser resolvido.

- Em boa parte dos problemas de engenharia uma solução pode ser dada por um vetor real que especifica as dimensões dos parâmetros importantes do problema. Exemplos:
  - *Problemas de controle*: uma solução é dada por uma variação funcional de parâmetros de controle em relação ao espaço ou ao tempo.
  - *Teoria de jogos*: uma solução é uma estratégia de realização de uma determinada tarefa.
- Entretanto, a estrutura de uma solução (candidata) poderá depender diretamente do problema abordado. Cada método de busca também possui características que ajudam na especificação do tipo de representação a ser empregada.
  - Sendo assim, a eficiência e complexidade do método de busca irá depender da representação e conseqüentemente dos operadores de busca escolhidos.

- Fica claro, portanto, que a definição de uma representação é uma das etapas mais críticas na implementação de um algoritmo evolutivo. A definição inadequada da representação pode levar a superfícies de *fitness* extremamente "acidentadas".
- Em problemas de otimização com restrições, a codificação adotada pode fazer com que indivíduos modificados por crossover/mutação sejam infactíveis.
  - Nestes casos, cuidados especiais devem ser tomados na definição da codificação e/ou dos operadores (BÄCK *et al.*, 2000b).

2.1 Codificação Binária

- Na maioria das aplicações de algoritmos genéticos (GAs) as estruturas de dados utilizadas como representação das soluções candidatas são cadeias binárias, mesmo quando as variáveis do problema são inteiras ou reais.
  - Uma variável real $x \in (a,b)$ pode ser codificada utilizando-se cadeias binárias de comprimento l, o que resultará em uma precisão numérica de $(a-b)/(2^l-1)$.

- A motivação para o uso de codificação binária vem da teoria dos esquemas (*schemata theory*). HOLLAND (1975; 1992) argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao GA, e prova que um alfabeto binário maximiza o paralelismo implícito.
- Entretanto, em diversas aplicações práticas a utilização de codificação binária leva a um desempenho insatisfatório. Em problemas de otimização numérica com parâmetros reais, algoritmos genéticos com representação em ponto flutuante freqüentemente apresentam desempenho superior à codificação binária.
- MICHALEWICZ (1996) argumenta que a representação binária apresenta desempenho pobre quando aplicada a problemas numéricos com alta dimensionalidade e onde alta precisão é requerida. Suponha por exemplo, que temos um problema com 100 variáveis com domínio no intervalo [–500, 500] e que precisamos de 6 dígitos de precisão após a casa decimal. Neste caso precisaríamos de um cromossomo de comprimento 3000, e teríamos um espaço de

busca de dimensão aproximadamente 10^{1000}. Neste tipo de problema o algoritmo genético clássico apresenta desempenho pobre.

- Uma das características da representação binária é que dois pontos vizinhos no espaço de parâmetros não são necessariamente vizinhos no espaço de busca definido pela representação do problema.
- Uma forma de solucionar este problema é utilizar uma representação do tipo código de Gray, onde a distância Hamming entre duas cadeias consecutivas quaisquer é sempre 1.

Inteiro	Binário	Gray
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

- Mesmo assim, a mudança de um único bit em codificação binária ou Gray pode resultar em grandes “saltos” no valor decodificado.
- Procedimentos para transformar um código binário em Gray e vice-versa. Sejam $\boldsymbol{b} = \langle b_1, ..., b_l \rangle$ a cadeia binária e $\boldsymbol{g} = \langle g_1, ..., g_l \rangle$ a cadeia em código Gray.

```
procedimento Binário_para_Gray
  g1 ← b1
  para k = 2 até l faça,
    gk ← gk-1 XOR bk
  fim para
fim procedimento

procedimento Gray_para_Binário
  valor ← g1
  b1 ← valor
  para k = 2 até l faça,
      se gk == 1
          então valor ← NOT valor
      fim se
    bk ← valor
  fim para
fim procedimento
```

2.2 Codificação Real

- As estratégias evolutivas e a programação evolutiva, em suas formad atuais, são empregadas, na maioria dos casos, para resolver problemas de otimização contínua.

  $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Re} f(\boldsymbol{x})$

- Enquanto o algoritmo genético padrão trabalha com o vetor de atributos codificado em cadeias binárias, a EE e a PE operam diretamente no vetor de atributos.
  - Nos GAs, a escolha de uma codificação para os parâmetros a serem otimizados está fundamentada na teoria dos esquemas. Ou seja, acredita-se que é importante operar com sub-seções (blocos construtivos) ótimas de soluções candidatas.

2.3 Permutações

- Uma permutação de um conjunto finito é um arranjo linear de seus elementos (KNUTH, 1973).
  - Dados *n* objetos, existem *n*! permutações destes objetos.
- Existem diversos problemas que são naturalmente representados por permutações. Exemplos: TSP, seqüenciamento de tarefas, e caminho Hamiltoniano (conjunto de nós formando um ciclo que passa por cada nó uma única vez). Estes problemas apresentam características distintas dos problemas de otimização de parâmetros.
  - Exemplo: no caixeiro viajante simétrico, cada deslocamento positivo ou negativo (*shift* ou *reversal*) da permutação corresponde a mesma rota.
- Neste tipo de codificação os operadores genéticos a serem empregados devem ser distintos dos utilizados com representação binária ou real.

2.4 Máquinas de Estado Finito

- As primeiras técnicas de programação evolutiva (PE) utilizavam como representação máquinas de estado finito (MEF), definidas por uma 5-tupla: $M = \langle Q,\tau,\rho,s,o \rangle$.
- Esta representação era apropriada para evoluir estruturas capazes de realizar previsões e generalizações dado um conjunto de sinais de entrada utilizados para a evolução da MEF.

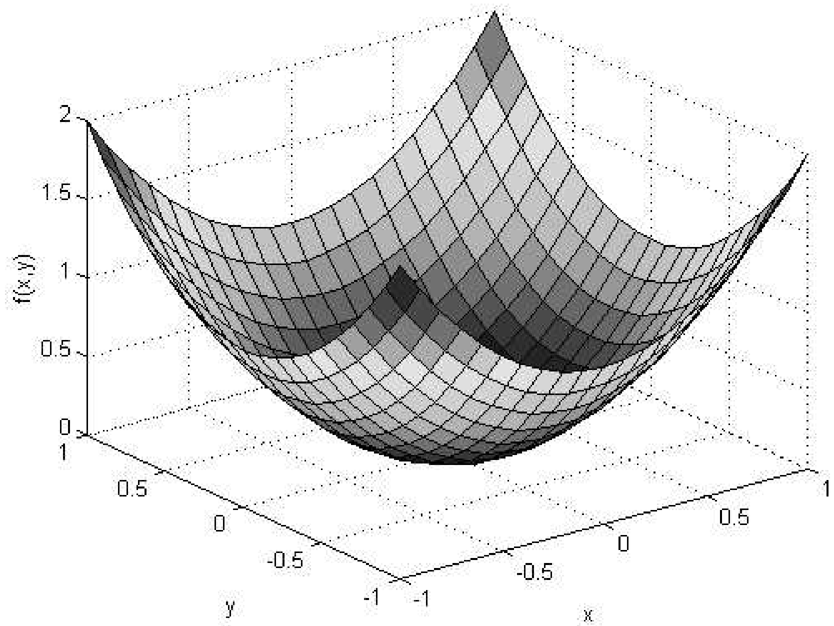
2.5 Árvores Sintáticas (Parse Trees)

- Quando uma estrutura executável, tal qual um programa ou uma função computacional, é a estrutura de dados de um algoritmo evolutivo, a representação possui um papel fundamental no sucesso da busca.
- Se uma linguagem de programação tradicional for utilizada para representar os programas a serem evoluídos, a manipulação através dos operadores evolutivos irá

resultar quase que invariavelmente em programas sintaticamente inválidos (BÄCK et al., 2000a).
  - Sendo assim, é útil projetar uma representação que garanta a correção sintática dos programas criados.
  - Uma representação com estas características é dada por árvores sintáticas.
- A representação por árvores sintáticas apresenta uma característica recursiva natural, permitindo a criação de estruturas com dimensão variável.
  - Entretanto, todas as aplicações práticas descritas em KOZA (1992) utilizam mecanismos para limitar o tamanho final do programa evoluído. Duas técnicas são usualmente empregadas: limitação de profundidade ou de número de nós (ANGELINE & KINNEAR JR., 1996).
- É importante salientar que praticamente todos os compiladores das linguagens de programação utilizam árvores sintáticas para interpretar expressões aritméticas ou algébricas.

2.6 Dicas para uma Codificação Apropriada

- No Tópico 1 foi discutido que para resolver um problema empregando-se um método de busca é necessário definir 3 aspectos: *representação, função objetivo,* e *função de avaliação*.
- Para a aplicação de algoritmos evolutivos como ferramenta de busca, é necessário, além dos três ítems acima, definir também *operadores genéticos* e *mecanismos de seleção*.
  - Entretanto, nenhum destes cinco itens é independente dos demais.
- A interdependência entre a representação e as outras características do algoritmo evolutivo escolhido sugerem que, na maioria dos casos, a escolha apropriada da representação irá depender da habilidade do projetista visualizar a dinâmica da busca na superfície de adaptação.
- Exemplo: considere a seguinte função $f(x,y) = x^2 + y^2$ a ser minimizada.



- Uma representação intuitiva é utilizar vetores reais para representar as variáveis x e y.

- Operadores genéticos apropriados seriam mutação Gaussiana e crossover aritmético.
- Um mecanismo de seleção poderia ser elitista, uma vez que a função é convexa.

- Conclui-se portanto, que a representação pode ser o mais natural possível, ou seja, derivada das próprias características do problema.

3 Mecanismos de Seleção

- Trata-se de um dos principais operadores dos algoritmos evolutivos cujo objetivo é selecionar os "melhores" indivíduos da população em detrimento dos "piores".
  - Portanto, a seleção é uma mistura de dois conceitos: *seleção* e *reprodução*.
- A identificação da qualidade de um indivíduo é baseada em seu valor de fitness.
- A idéia básica é privilegiar (aumentar a probabilidade de seleção de) indivíduos com valores relativos mais elevados de fitness.
- Os operadores de seleção podem ser *determiníticos* ou *probabilísticos*.

3.1 Teoria da Pressão Seletiva

- Os operadores de seleção são caracterizados por um parâmetro conhecido como *pressão seletiva*, que relaciona o tempo de *dominância* (*takeover*) do operador.
- O tempo de dominância é definido como sendo a velocidade para que a melhor solução da população inicial domine toda a população através da aplicação isolada do operador de seleção (BÄCK, 1994; GOLDBERG & DEB, 1991).
- Se o tempo de dominância de um operador é grande, então a pressão seletiva é fraca, e vice-versa.
- Portanto, a pressão seletiva oferece uma medida de quão "guloso" é o operador de seleção no que se refere à dominância de um indivíduo da população.
  - Se o operador de seleção apresenta uma forte pressão seletiva, então a população perde diversidade rapidamente.

- Sendo assim, para evitar uma rápida convergência para pontos sub-ótimos é necessário empregar populações com dimensões elevadas e/ou operadores genéticos capazes de introduzir e/ou manter a diversidade populacional.
  - Por outro lado, operadores genéticos com estas características tornam a convergência do algoritmo lenta nos casos em que a pressão seletiva é fraca. Isto permite uma maior exploração do espaço de busca.
- A discussão acima sugere que o sucesso na aplicação de um algoritmo evolutivo depende do mecanismo de seleção empregado, que, por sua vez, irá depender dos operadores genéticos e outros parâmetros escolhidos para o algoritmo.

3.2 A Função de Fitness

- O processo de avaliação de indivíduos em um algoritmo evolutivo começa com a definição de uma função objetivo: $f : A_x \rightarrow \Re$, onde A_x é o espaço de atributos das variáveis.

- A função de fitness $\phi : A_x \rightarrow \Re_+$ faz um mapeamento dos valores nominais da função objetivo em um intervalo não-negativo.
- A função de fitness é geralmente uma composição da função objetivo com uma função de escalonamento: $\phi(a_i(t)) = g(f(a_i(t)))$, onde $a_i(t) \in A_x$.
- O mapeamento acima faz-se necessário caso se deseje <u>minimizar</u> a função objetivo, uma vez que valores maiores de fitness corresponderão a valores menores da função objetivo. Exemplos de funções objetivo escalonadas:
  - $\phi(a_i(t)) = f_{\max} - f(a_i(t))$, caso o ótimo global seja conhecido
  - $\phi(a_i(t)) = f_{\max}(t) - f(a_i(t))$, onde $f_{\max}(t)$ é o máximo até a iteração t
  - $\phi(a_i(t)) = 1 / [1 + f(a_i(t)) - f_{\min}(t)]$, onde $f_{\min}(t)$ é o mínimo até a iteração t
  - $\phi(a_i(t)) = 1 / [1 + f_{\max}(t) - f(a_i(t))]$, onde $f_{\max}(t)$ é o máximo até a iteração t
- As duas últimas funções acima retornam o valor do fitness normalizado ente (0,1].

3.2.1 Fitness Ajustado

- O fitness ajustado é calculado a partir do fitness escalonado:

- $\phi'(a_i(t)) = 1 / [1 + \phi(a_i(t))]$, onde $\phi(a_i(t))$ é o fitness escalonado de $a_i(t)$.

- $\phi'(a_i(t)) \in [0,1]$.
- O fitness ajustado apresenta o benefício de salientar pequenas diferenças no valor escalonado do fitness.

3.2.2 Fitness Normalizado

- Se o método de seleção a ser empregado é do tipo proporcional ao fitness, então o fitness a ser empregado deve ser normalizado:
  - $\phi_n(a_i(t)) = \phi'(a_i(t)) / \sum_j \phi'(a_j(t))$, onde $\phi'(a_i(t))$ é o fitness ajustado de $a_i(t)$.
- O fitness normalizado apresenta as seguintes características:
  - $\phi_n(a_i(t)) \in [0,1]$;
  - é maior para melhores indivíduos; e
  - $\sum_i \phi_n(a_i(t)) = 1$.

IA707 – Profs. Leandro de Castro/Fernando Von Zuben
DCA/FEEC/Unicamp

3.2.3 Escalonamento do Fitness (Fitness Scaling)

- Devido à pressão seletiva, a população tende a ser dominada por aqueles indivíduos de maior fitness.
- Nestes casos, as funções de fitness descritas acima tendem a definir valores similares de fitness a todos os membros da população.
- Para solucionar este problema, métodos de escalonamento de fitness foram propostos com o objetivo de acentuar pequenas diferenças nos valores de fitness, mantendo assim os efeitos da pressão seletiva.
- GREFENSTETTE (1986) propôs uma função de fitness como sendo uma transformação linear variante no tempo da função objetivo:
  - $\phi(a_i(t)) = \alpha.f(a_i(t)) - \beta(t)$, onde $\alpha = 1$ para problemas de maximização e $\alpha = -1$ para problemas de minimização, e $\beta(t)$ representa o pior valor encontrado nas últimas gerações.

- GOLDBERG (1989) propôs o *sigma scaling* baseado na distribuição dos valores objetivo da população atual:
  - $\phi(a_i(t)) = f(a_i(t)) - (f_{av}(t) - c\sigma_f(t))$, caso $f(a_i(t)) > (f_{av}(t) - c\sigma_f(t))$, e
  - $\phi(a_i(t)) = 0$, caso $f(a_i(t)) \leq (f_{av}(t) - c\sigma_f(t))$.
  - Onde $f_{av}(t)$ é o valor médio da função objetivo da população atual, $\sigma_f(t)$ é o desvio padrão dos valores da função objetivo da população atual, e c é uma constante.

3.3 Seleção Proporcional ao Fitness

- O GA clássico utiliza a *seleção proporcional ao fitness*, geralmente implementado com o algoritmo da roleta (*roulette wheel*).
- O *roulette wheel* atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração proporcional a seu fitness normalizado, ou seja, o fitness medido em relação à somatória do fitness de todos os indivíduos da população.

- Assim, quanto maior o *fitness* de um indivíduo, maior a probabilidade dele passar para a próxima geração.
- Observe que a seleção de indivíduos por *roulette wheel* permite a perda do melhor indivíduo da população.

3.4 Seleção Por Torneio

- É um dos mais refinados processos de seleção, por permitir ajustar a pressão seletiva.
- A seleção é feita em função do número de vitórias de cada indivíduo em N competições contra q oponentes aleatórios da população, sendo que vence uma competição aquele que apresentar o maior fitness (comparado ao de seu(s) oponente(s)).
- Para propósitos práticos, $q \geq 10$ conduz a uma forte pressão seletiva, enquanto valores de q entre 3 e 5 levam a uma fraca pressão seletiva.
  - Para $q = 1$, nenhuma seleção está sendo feita.

- Para $q = 2$, tem-se o chamado torneio binário.
- Para $q \to N$ tem-se simplesmente a seleção por ordem de fitness, sem nenhuma aleatoriedade.

3.5 Seleção Baseada em Rank

- Este mecanismo utiliza apenas as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção.
- A seleção por rank simplifica o processo de mapeamento do objetio para a função de fitness: $\phi(a_i(t)) = \alpha f(a_i(t))$, onde $\alpha = 1$ para problemas de maximização e $\alpha = -1$ para problemas de minimização.
- O rank também elimina a necessidade de escalonamento do fitness, uma vez que a pressão seletiva é mantida mesmo que os valores de fitness dos indivíduos sejam muito próximos um do outro, o que normalmente ocorre após muitas gerações.
- Podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção.

- *Ranking linear*: a probabilidade de seleção de um indivíduo é proporcional a seu rank, onde o indivíduo menos apto (com menor fitness) possui rank 0 e o rank do melhor indivíduo é $N - 1$.
  - Sejam β_{rank} e α_{rank} a quantidade esperada de filhos do melhor e do pior indivíduo da população a cada geração. A probabilidade de seleção de um indivíduo é dada por:
  - $p(i) = [\alpha_{\text{rank}}+[\text{rank}(i)/(N-1)].(\beta_{\text{rank}}-\alpha_{\text{rank}})]/N$,
- *Ranking não-linear*: a probabilidade de seleção de um indivíduo é uma função não-linear de seu rank. Exemplos:
  - $p(i) = \alpha(1-\alpha)^{N-1-\text{rank}(i)}$, onde $\alpha \in (0,1)$.
  - $p(i) = (1-\exp(-\text{rank}(i)))/c$, onde c é um fator de normalização.
- Note que os mecanismos de seleção (μ,λ) e $(\mu+\lambda)$ das estratégias evolutivas são do tipo rank determinístico.

3.6 Seleção de Boltzmann

- Os mecanismos de seleção de Boltzmann controlam termodinâmicamente a pressão seletiva utilizando princípios de simulated annealing (SA).
- Os algoritmos evolutivos com seleção de Boltzmann podem ser vistos como extensões paralelas do algoritmo de SA (MAHFOUD & GOLDBERG, 1995).
- A idéia básica da seleção de Boltzmann é utilizar uma distribuição de Boltzmann-Gibbs como mecanismo de competição entre indivíduos:

  $P(x') = [1+\exp(f(x) - f(x'))/T]^{-1}$,

  Onde T é a temperatura do sistema, x e x' são os dois indivíduos que estão competindo para serem selecionados, e $f(\cdot)$ o valor da função de fitness.

3.7 Seleções Bi-Classista e Elitista

- Na seleção bi-classista, são escolhidos os b% melhores indivíduos e os w% piores indivíduos da população. O restante (100–(b+w))% é selecionado aleatoriamente com ou sem reposição.
- O elitismo consiste em um caso particular de seleção bi-classista na qual um ou mais dos melhores indivíduos da população é sempre mantido e nenhum dos piores indivíduos é selecionado. Ou seja, $b \neq 0$ e $w = 0$.

4 Operadores de Busca

- Sabemos que a evolução é resultado da *reprodução*, *variação genética* e *seleção natural* aplicados à uma população de indivíduos.
- Até o momento já foram discutidos os principais tipos de representação e mecanismos de seleção dos algoritmos evolutivos.
- Resta agora identificar os principais tipos de operadores genéticos a serem empregados.

- Existem *transformações unárias*, do tipo mutação, que criam um novo indivíduo (filho) partindo de um único pai, e também *transformações de ordem mais elevada*, do tipo recombinação, que geram novos indivíduos através da recombinação de características de dois ou mais indivíduos pais.
- Como discutido, existe uma interdependência entre a representação empregada (que geralmente depende do problema) e os operadores genéticos escolhidos.
- Sendo assim, os operadores genéticos serão apresentados considerando-se a representação adotada.

5 Operadores de Mutação

- Geração de um novo indivíduo (filho) partindo de um único pai.

5.1 Codificação Binária

- A mutação inicialmente proposta para representação com cadeias binárias é denominada de *mutação pontual*. Cada posição da cadeia possui uma probabilidade *pm* de sofrer mutação.

- Estudos empíricos e teóricos sugerem que:
  - Um valor inicial grande para a mutação deve ser adotado e decrescido geometricamente ao longo das gerações (FOGARTY, 1989);
  - Um limite inferior $pm = 1/l$ para a taxa ótima de mutação pode ser empregado (BREMERMANN et al., 1966, MÜHLENBEIN, 1992, BÄCK, 1993).

5.2 Codificação Real

- A mutação refere-se ao processo de gerar novos indivíduos partindo de um único pai. Sendo assim, dado um indivíduo $\mathbf{x} \in \Re$, seu correspondente valor mutado pode ser expresso por: $\mathbf{x}' = m(\mathbf{x})$, onde $m(\cdot)$ é a função de mutação. Exemplos:
  - $\mathbf{x}' = \mathbf{x} + \mathbf{M}$, onde $\mathbf{M}$ é uma variável aleatória.
  - Geralmente $\mathbf{M}$ possui média zero tal que $E(\mathbf{x}') = \mathbf{x}$, ou seja, a esperança matemática da diferença entre $\mathbf{x}$ e $\mathbf{x}'$ é zero.

- *Mutação Uniforme*: ***M*** pode assumir diversas formas, como, por exemplo, uma distribuição aleatória uniforme $U(a,b)^l$, onde a e b são os limites inferiores e superiores da variável. Geralmente $a = -b$.
- *Mutação Gaussiana*: Outra alternativa para ***M*** é utilizar uma distribuição normal ou Gaussiana $N(mean,\sigma)^l$ de média *mean* (em geral *mean* = 0) e desvio padrão σ.

• Outro operador de mutação, especialmente desenvolvido para problemas de otimização com restrição e codificação em ponto flutuante, é a chamada *mutação não-uniforme* (MICHALEWICZ, 1996; MICHALEWICZ & SCHOENAUER, 1996). A mutação não-uniforme é um operador dinâmico, destinado a melhorar a sintonia fina de um elemento simples. Podemos definí-lo da seguinte forma: seja $\mathbf{x}^t = [x_1 \ldots x_n]$ um cromossomo e suponha que o elemento x_k foi selecionado para mutação; o cromossomo resultante será $\mathbf{x}^{t+1} = [x_1 \ldots x'_k \ldots x_n]$, onde

$$x'_k = \begin{cases} x_k + \Delta(t, a - x_k), & \text{com 50\% de probabilidade} \\ x_k - \Delta(t, x_k - b), & \text{com 50\% de probabilidade} \end{cases}$$

onde a e b são os limites inferiores e superiores da variável x_k. A função $\Delta(t, y)$ retorna um valor no intervalo $[0, y]$ tal que a probabilidade de $\Delta(t, y)$ ser próximo de zero aumenta à medida que t aumenta.

- Esta propriedade faz com que este operador inicialmente explore o espaço globalmente (quando t é pequeno) e localmente em gerações avançadas (quando t é grande);

• MICHALEWICZ (1996) propõe a seguinte função:

$$\Delta(t, y) = y \cdot \left(1 - r^{(1-t/T)^p}\right)$$

onde r é um número aleatório no intervalo [0, 1], T é o número máximo de gerações e p é um parâmetro que determina o grau de dependência do número de iterações (valor proposto por MICHALEWICZ (1996): $p = 5$).

5.3 Permutações

- Qualquer mutação a ser aplicada à uma permutação deve gerar uma estrutura de dados que também é uma permutação.
- A mutação mais comum para permutações é conhecida como *2-opt* (LIN & KERNIGHAN, 1973). Este procedimento seleciona dois pontos da cadeia e reverte o segmento entre os pontos. Exemplo:
  - A | B C D E | F $\rightarrow$ A | E D C B | F
- O operador de mutação também pode ser extendido para o caso *k-opt*, ou seja, *k* pontos são selecionados e as sub-seqüências são revertidas. Exemplo, $k = 4$:
  - A | B C D | E F | G H | I J $\rightarrow$ A | D C B | F E | H G | I J
- Um caso particular da mutação 2-opt é aquele em que duas posições (alelos) são selecionadas e seus genes trocados. Esta mutação é denominada de *mutação baseada em ordem* (SYSWERDA, 1991).

IA707 – Profs. Leandro de Castro/Fernando Von Zuben
DCA/FEEC/Unicamp

- Outro operador de mutação, denominado de *mistura* (*scramble*) seleciona uma sublista e reordena seus elementos aleatoriamente (SYSWERDA, 1991).
- Note que a mutação *k*-opt é sensível a adjacência dos elementos da permutação. Isso pode causar alguns efeitos interessantes:
  - O TSP pode eliminar um cruzamento na rota.
  - Em um problema de seqüenciamento de tarefas a mutação pode corresponder a uma mudança grande no acesso aos recursos disponíveis.

5.4 Máquinas de Estado Finito

- Existem diversas formas de mutar uma máquina de estado finito (MEF).
  - Mudar um símbolo de saída;
  - Mudar uma transição de estado;
  - Adicionar um estado;
  - Deletar um estado;
  - Mudar o estado inicial.

5.5 Árvores Sintáticas

- Assim como no caso dos algoritmos genéticos, em PG a mutação é geralmente relegada a segundo plano. Como conseqüência, o tamanho dos indivíduos da população pode crescer significativamente.
- ANGELINE & KINNEAR JR. (1996) propuseram 4 tipos básicos de mutação em PG:
    - Crescimento, encolhimento, troca, e ciclo.

6 Operadores de Recombinação

- Os operadores de recombinação trocam partes das estruturas de dados de dois ou mais pais com o objetivo de produzir um ou mais filhos.

6.1 Codificação Binária

- Crossover de 1 ou *n* pontos entre indivíduos aleatórios (Roulette Wheel), ou então entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo.

- Crossover *uniforme* (ACKLEY, 1987; SYSWERDA, 1989) ou baseado em *máscara* entre indivíduos aleatórios (Roulette Wheel), ou então entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo.
- SCHAFFER & MORISHIMA (1987) propuseram o *crossover puntuado* (*punctuated crossover*) onde uma cadeia binária representando *marcas puntuadas* m_i, $i = 1,...,l$ indicam os pontos de crossover para o crossover de múltiplos pontos. Esta cadeia binária é adicionada ao final da cadeia representante da estrutura de dados e estará sujeita ao processo evolutivo: $\boldsymbol{x} = (x_1, x_2,..., x_l, m_1,..., m_l)$

6.2 Codificação Real

- A maioria dos operadores de crossover utilizados com codificação real são derivados das estratégias evolutivas.
- Entretanto, crossover de um ou mais pontos também podem ser empregados para representação real.

- Diversos operadores de recombinação podem ser definidos. Seja x_i o atributo i do vetor x a sofrer mutação, e a,b os índices dos pais a ou b, respectivamente:
  - *Recombinação discreta* (*local*): $x_i = x_{a,i}$ ou $x_{b,i}$ (crossover *uniforme*)
  - *Recombinação intermediária* (*local*): $x_i = ½(x_{a,i} + x_{b,i})$
  - *Recombinação discreta global*: $x_i = x_{a,i}$ ou $x_{bi,i}$
  - *Recombinação intermediária global*: $x_i = ½(x_{a,i} + x_{bi,i})$

  onde bi é um pai qualquer escolhido da população atual.

- Outro operador para representação real é o *crossover aritmético* definido como uma combinação linear de dois vetores (cromossomos): sejam $\mathbf{x}_1$ e $\mathbf{x}_2$ dois indivíduos selecionados para crossover, então os dois filhos resultantes serão $\mathbf{x}'_1 = a\mathbf{x}_1 + (1-a)\mathbf{x}_2$ e $\mathbf{x}'_2 = (1-a)\mathbf{x}_1 + a\mathbf{x}_2$, onde a é um número aleatório pertencente ao intervalo [0,1]. Este operador é particularmente apropriado para problemas de otimização numérica com restrições, onde a região factível é convexa. Isto porque, se $\mathbf{x}_1$ e $\mathbf{x}_2$ pertencem à região factível, combinações

convexas de $\mathbf{x}_1$ e $\mathbf{x}_2$ serão também factíveis. Assim, garante-se que o crossover não gera indivíduos infactíveis para o problema em questão.

- Outros exemplos de crossover especialmente desenvolvidos para utilização em problemas de otimização numérica restritos e representação real são:
  - *Crossover heurístico* (WRIGHT, 1994): $\boldsymbol{x}' = u(\boldsymbol{x}_1 - \boldsymbol{x}_1) + \boldsymbol{x}_2$, onde $u \in [0,1]$ e $\boldsymbol{x}_1$ e $\boldsymbol{x}_2$ são dois pais tais que $f(\boldsymbol{x}_2) \geq f(\boldsymbol{x}_1)$.
  - *Crossover geométrico*, *crossover esférico*, e *crossover simplex* descritos em MICHALEWICZ & SCHOENAUER (1996). Veja também BÄCK *et al.* (2000a).

6.3 Permutações

- A representação por permutações possui a característica de que os operadores de crossover mais simples falham na geração de um indivíduo que também seja uma permutação.

- Sendo assim, operadores específicos para permutações devem ser propostos. Exs.:

  - *Crossover OX*:

    P1: A B | C D E F | G H I
    P2: f h | d a b c | i g e
    F1: a b | C D E F | i g h
    F2: H I | d a b c | E F G

  - *Partially Mapped Crossover PMX* (GOLDBERG & LINGLE, 1985):

    P1: A B | C D E | F G
    P2: c f | e b a | d g
    F1: a f | C D E | b g
    F2: C D | e b a | F G

  - *Crossover de ordem 2* (SYSWERDA, 1991):

    P1: A B C D **E** F **G**
    P2: c **f** g **b** **a** d e
    F1: f b C D E a G
    F2: c f E b a d G

  - *Crossover de posição* (SYSWERDA, 1991):

    P1: A B **C** **D** **E** F **G**
    P2: c f **g** **b** **a** d **e**
    F1: f b C D E a G
    F2: C D g b a F e

  - *Crossover de máxima preservação MPX* (MÜHLENBEIN, 1991):

    P1: A | B C D | E F G
    P2: c | f g b | a d e
    F1: e B C D f g a
    F2: E f g b A C D

6.4 Máquinas de Estado Finito

- Embora na versão inicialmente proposta para a PE a recombinação entre duas MEFs não fosse empregada, alguns autores propuseram mecanismos para recombinar partes de máquinas de estado finito. Exemplos:

- o Troca de estados entre MEFs, ou seja, para um dado estado, troca-se o símbolo de saída e a transição de próximo estado para cada entrada.

6.5 Árvores Sintáticas

- Árvores sintáticas, da forma como empregadas na PG, requerem operadores de recombinação que garantam a geração de filhos válidos. Ou seja, a árvore deve possuir apenas terminais em suas folhas, e funções em seus nós de grau maior do que 1. Além disso, cada nó função da árvore deve possuir o número correto de sub-árvores, uma para cada argumento da função.
- CRAMER (1985) definiu o operador que hoje é padrão para a recombinação de árvores sintáticas, o crossover de sub-árvores.

7 Abordagens baseadas em população

- **Abordagem Michigan** – a população como um todo é a solução para o problema.
- **Abordagem Pittsburgh** – cada elemento da população corresponde a uma solução do problema.

8 Definição da População Inicial

- O método mais comum utilizado na criação da população é a inicialização aleatória dos indivíduos. Se algum conhecimento inicial a respeito do problema estiver disponível, pode ser utilizado na inicialização da população.
- Por exemplo, se é sabido que a solução final (assumindo codificação binária) vai apresentar mais 0's do que 1's, então esta informação pode ser utilizada, mesmo que não se saiba exatamente a proporção.
- Em problemas com restrição, deve-se tomar cuidado para não gerar indivíduos inválidos já na etapa de inicialização.

9 Decisões Críticas

- Tipo de codificação?
- Representar toda ou parte da solução?
- Só crossover, só mutação, ou ambos?
- Populações de tamanho fixo ou variável ?

- Cromossomos de tamanho fixo ou variável?
- Cromossomos haplóides ou diplóides?
- Qual critério de parada?
- Usar controle de diversidade?
- Usar busca local?
- Usar busca multi-modal?
- Usar co-evolução?
- Abordagem Michigan ou Pittsburgh?
- Algoritmos genéticos, programação genética, estratégias evolutivas, programação evolutiva?

10 Referências bibliográficas

ACKLEY, D. H. (1987), *A Connectionist Machine for Genetic Hillclimbing*, Kluwer.

ANGELINE, P.J., KINNEAR JR., K.E. (eds.) "Advances in Genetic Programming". volume II, MIT Press, 1996.

BÄCK, T. (1994), "Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms", *Proc. Of the 1st IEEE Conf. on Evolutionary Computation*, Orlando, FL, pp. 57-62.

BÄCK, T. (1993), "Optimal Mutation Rates in Genetic Search", *Proc. of the 5th Int. Conf. on Gen. Algorithms*, S. Forrest (ed.), pp. 2-8.

BÄCK, T., FOGEL, D.B. & MICHALEWICZ, Z. (eds.) "Evolutionary Computation 1: Basic Algorithms and Operators", Institute of Physics Publishing, 2000a.

BÄCK, T., FOGEL, D.B. & MICHALEWICZ, Z. (eds.) "Evolutionary Computation 2: Advanced Algorithms and Operators", Institute of Physics Publishing, 2000b.

BERTONI, A. & DORIGO, M. "Implicit Parallelism in Genetic Algorithms", *Artificial Intelligence*, 61(2): 307-314, 1993.

BREMERMANN ET AL. (1966), "Global Properties of Evolution Processes", *Natual Automata and Useful Simulations*, H. H. Pattec et al. (eds.), pp. 3-41.

CRAMER, N.L. "A representation for the adaptive generation of simple sequential programs". *in* J.J. Grefenstette (ed.) *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 1985.

FOGARTY, T. C. (1989), "Varying the Probability of Mutation in the Genetic Algorithm", *Proc. of the 3rd Int. Conf. on Gen. Algorithms*, pp. 104-109.

FOGEL, D. B. "An Introduction to Simulated Evolutionary Computation", *IEEE Transactions on Neural Networks*, 5(1): 3-14, 1994.

GOLDBERG, D. E. "Messy Genetic Algorithms: Motivation, Analysis, and First Results", *Complex Systems*, 3: 493-530, 1989.

GOLDBERG, D. E. & DEB, K. (1991), "A Comparison of Selection Schemes Used in Genetic Algorithms", *Foundations of Genetic Algorithms*, G. J. E. Rawlins (ed.), Morgan Kaufmann, pp. 69-93.

GOLDBERG, D. E. & LINGLE, R. JR. (1985), "Alleles, Loci, and the Travelling Salesman Problem", *Proc. of the 1st ICGA*, S. Forrest (ed.), pp. 536-542.

GREFENSTETTE, J. (1986), "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. on Syst., Man, and Cybernetics*, **16**, pp. 122-128.

HOLLAND, J.H. "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975.

HOLLAND, J.H. "Adaptation in Natural and Artificial Systems", 2nd edition, The MIT Press, 1992.

KNUTH, R. M. (1973), *The Art of Computer Programming Volume 3: Sorting and Searching*, Addison-Wesley.

KOZA, J.R. "Genetic Programming: On the Programming of Computers by means of Natural Selection", MIT Press, 1992.

LIN, S. & KERNIGHAN, B. (1973), "An Efficient Heuristic Procedure for the Travelling Salesman Problem", *Oper. Res.*, **21**, pp. 498-516.

MAHFOUD, S. & GOLDBERG, D. (1995), "Parallel Recombinative Simulated Annealing: A Genetic Algorithm", *Parallel Comput.*, **21**, pp. 1-28.

MICHALEWICZ, Z. "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer, 1996.

MICHALEWICZ, Z. & SCHOENAUER, M. "Evolutionary Algorithms for Constrained Parameter Optimization Problems", *Evolutionary Computation*, 4(1): 1-32, 1996.

MÜHLENBEIN, H. (1992), "How Genetic Algorithms Really Work: I Mutation and Hillclimbing", *PPSN*, **2** pp. 15-25.

MÜHLENBEIN, H. (1991), "Evolution in Time and Space – the Parallel Genetic Algorithm", *Foundations of Genetic Algorithms*, G. J. E. Rawlins (ed.), Morgan-Kaufmann.

SCHAFFER, J. D., & MORISHIMA, A. (1987), "An Adaptive Crossover Distribution Mechanism for Genetic Algorithms", *Proc. of the 2nd ICGA*, J. J. Grefensttete (ed.), pp. 36-40.

SYSWERDA, G. (1991), "Schedule Optimization Using Genetic Algorithms", *Handbook of Genetic Algorithms*, L. Davis (ed.), pp. 332-349.

SYSWERDA, G. "Uniform Crossover in Genetic Algorithms", *in* Schaffer, J.D. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pp. 2-9, 1989.

WRIGHT, A. H. 1994, "Genetic Algorithms for Real Parameter Optimization", *Foundations of Genetic Algorithms*, G. Rawlins (ed.), Morgan kaufmann, pp. 205-218.