Algoritmos Genéticos - Capítulo 10

Representação Baseada em Ordem

Prof. Ricardo Linden

Conceitos básicos

- Existe uma classe de problemas que não consiste de otimização numérica, mas sim de otimização combinatorial;
- Estes problemas podem ser resolvidos perfeitamente através de GAs;
- Estes problemas são em geral NP-completos, o que significa que seu espaço de busca é, para efeitos práticos, considerado como infinito;
- Duas instâncias típicas deste caso são o problema de colorir um grafo e o problema do caixeiro viajante.

Problemas Típicos

- Colorir um grafo:
  - Termos um grafo com *n* nós, cada um com um peso distinto;
  - São dadas *k* cores para colorir este grafo (*k*<*n*).
  - Objetivo: conseguir o maior escore possível somando os pesos dos nós coloridos;
  - Nós adjacentes não podem receber a mesma cor.
- Caixeiro viajante:
  - Um caixeiro que tem de visitar *n* cidades;
  - Não se pode passar duas vezes por nenhuma delas;
  - Objetivo: percorrer a menor distância possível.

Representação

- No dois casos que melhor representam a necessidade desta representação, estamos interessados na ordem em que o problema é resolvido;
- Queremos uma representação que contenha todos os nós (ou cidades) colocados em uma ordem;
- Conseqüência:
  - representação em lista;
  - Cada cromossomo contém todos os elementos presentes no problema (todas as cidades ou todos os nós do grafo).

Representação

- Exemplos:
  - (1 4 6 5 2 3 7), (1 2 3 4 5 6 7) e (7 5 3 1 2 4 6) são exemplos de cromossomos válidos para um problema envolvendo sete nós.
  - (1 6 2 5 7 4) não é um cromossomo válido, visto que o elemento 3 não está presente na nossa lista;
  - (1 4 6 5 2 **3 3** 7) não é um cromossomo válido, pois o elemento 3 aparece duas vezes na nossa lista.

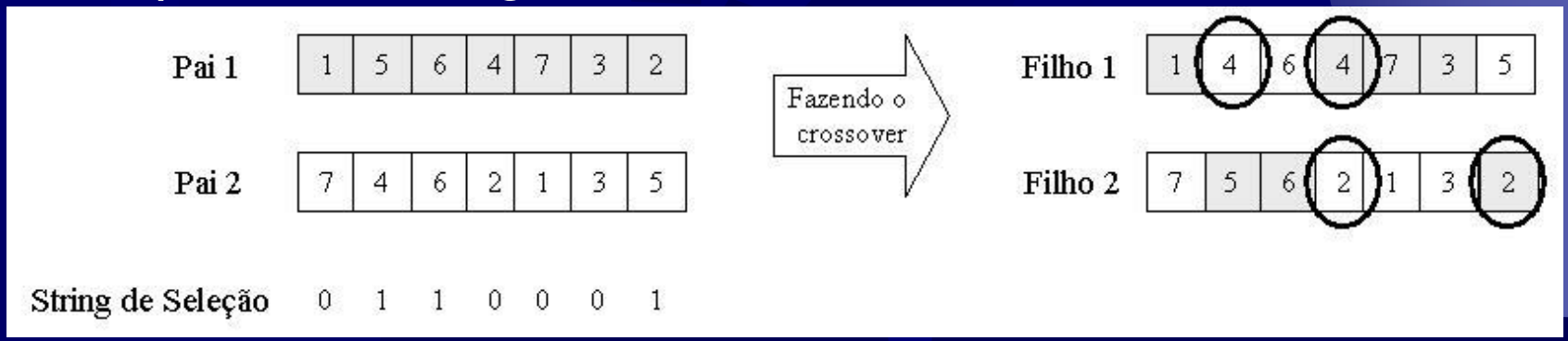
Avaliação

- A função de avaliação, como já discutido anteriormente, deve representar a qualidade de cada um dos cromossomos;
- Caixeiro viajante: some a distância entre a cidade contida no gene *i* à cidade contida no gene *i+1*, $\forall i=0,1,...,n-1$;
- Colorir um grafo:
  - tomaremos os nós um a um, na ordem fornecida pelo cromossomo;
  - designar-lhes-emos a primeira cor possível;
  - Se houver alguma cor possível, somaremos seu peso;

Agora só faltam os operadores...

Crossover baseado em ordem

- Versão especial do operador de crossover de dois pontos e de crossover uniforme;
- Necessário modificar operadores de modo que sempre geremos filhos válidos dentro deste novo formato de representação
  - não podemos simplesmente copiar posições do primeiro pai quando sortearmos um 1 e copiarmos posições do segundo pai quando sortearmos um 0;
  - poderíamos gerar um cromossomo com elementos



Crossover baseado em ordem

- A idéia de ordenação relativa leva a um novo conceito de esquema para a representação baseada em ordem.
- Um esquema, agora, é toda sub-lista de nosso cromossomo;
- Os *don't cares* correspondem a simplesmente ignorar a posição original de cada nó do cromossomo;
  - Exemplos de esquemas para o pai 1 da figura do slide anterior são (1 6 7), (1 4 3 2), (1 5), (6 4 7), etc.
  - Note que não colocamos mais os coringas (“*”).

Crossover baseado em ordem

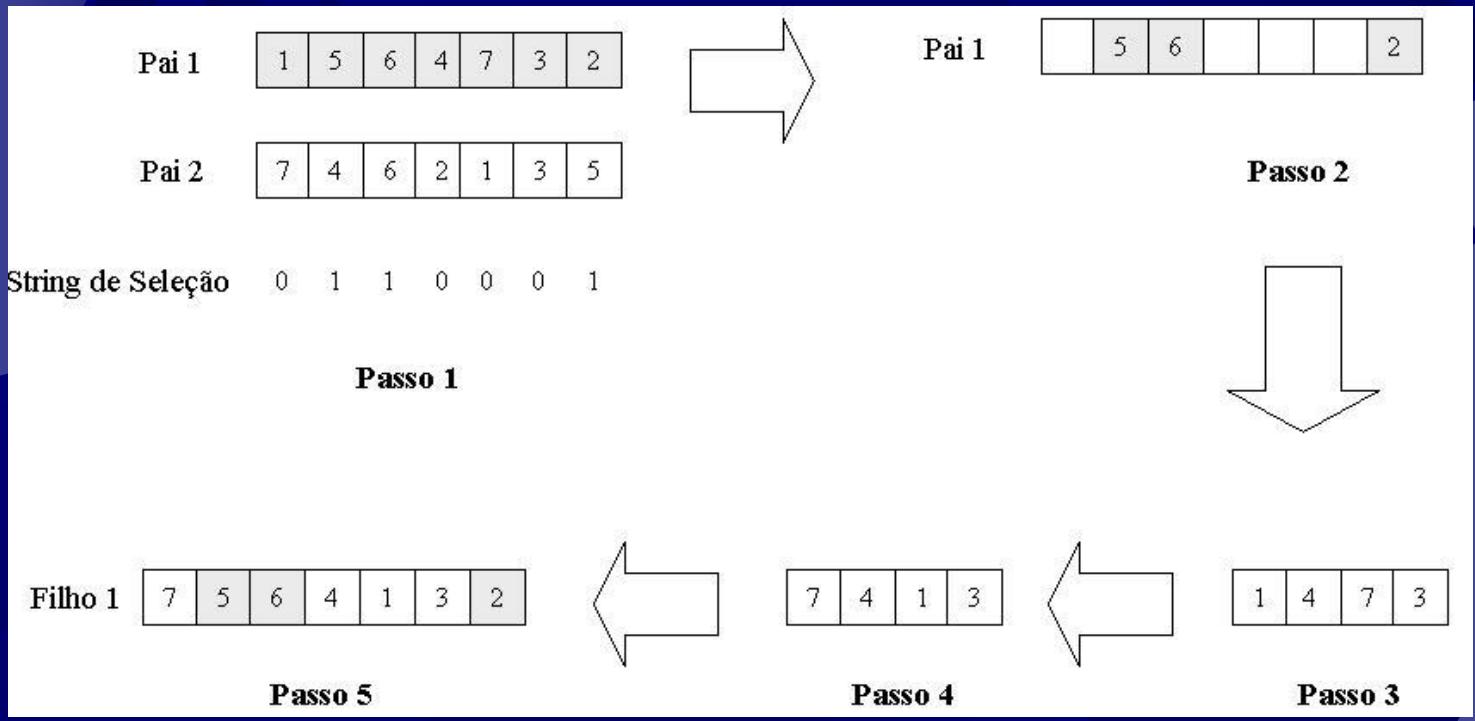
1. A partir desta nova visão dos cromossomos, genes e esquemas, podemos chegar ao seguinte algoritmo para atuação do crossover baseado em ordem que modifica o crossover de dois pontos:
   1. Selecione dois pontos de corte
   2. Copie para o filho 1 os elementos do pai 1
   3. Faça uma lista dos elementos do pai 1 fora dos pontos de corte.
   4. Permute esta lista de forma que os elementos apareçam na mesma ordem que no pai 2
   5. Coloque estes elementos nos espaços do pai 1 na ordem gerada no passo anterior
   - Analogia ⇨ Repita o processo para gerar o filho 2, substituindo o pai 1 pelo 2 e vice-versa

Crossover baseado em ordem

- Como o crossover de dois pontos binário, o número de esquemas que este operador pode manter é limitado;
- Há uma versão análoga ao funcionamento do crossover uniforme, que é o seguinte:
  1. Gere uma string de bits aleatória do mesmo tamanho que os elementos (assim como no crossover uniforme)
  2. Copie para o filho 1 os elementos do pai 1 referentes àquelas posições onde a string de bits possui um 1
  3. Faça uma lista dos elementos do pai 1 referentes a zeros da string de bits
  4. Permute esta lista de forma que os elementos apareçam na mesma ordem que no pai 2
  5. Coloque estes elementos nos espaços do pai 1 na ordem gerada no passo anterior
  - Analogia ⇨ Repita o processo para gerar o filho 2, substituindo o pai 1 pelo 2 e vice-versa

Crossover baseado em ordem

- Exemplo (omitindo o passo de analogia)



Edge Recombination

- Existem outras maneiras de fazer o crossover de dois cromossomos baseados em ordem;
- Possibilidade recombinação de arestas (*edge recombination*), ou ER;
- Conceito básico:
  - Informação importante em um cromossomo não é a ordenação dos nós;
  - Importantes são as arestas entre nós;
  - Filhos gerados pelo crossover devem ser baseados na arestas entre os nós, e não na ordenação relativa entre eles.

Edge Recombination

- Algoritmo básico:

1. Monte a lista de arestas existentes em cada um dos dois pais.
2. Escolha o nó inicial de um dos pais.
3. Escolha uma dentre as arestas válidas para o nó escolhido, seguindo as seguintes recomendações:
   a. Escolha o nó ou cidade com menor número de arestas válidas
   b. Se houver um empate, escolha uma dentre as vencedoras aleatoriamente.
   c. Se não houver arestas válidas para o nó escolhido, escolha qualquer uma aleatoriamente.
4. Repita o processo até que não haja mais nós a escolher.

Edge Recombination

- Experimentos que indicam que a taxa de falha, em que não temos nenhuma aresta para escolher é muito baixa (cerca de 1%);

- Operador ER preserva características estruturais do grafo subjacente à nossa representação, ao preservar as arestas entre os nós que o formam.

Crossover de mapeamento parcial

- Pode ser visto como um crossover de permutações;
- Idéia básica:
  - trocar uma seqüência intermediária entre os dois pais
  - garantindo no processo que ambos os filhos receberão o conjunto completo de nós existentes.
- Algoritmo:
  1. Escolhem-se dois pontos de corte aleatoriamente nos pais
  2. Faz-se o mapeamento de cada nó entre os pontos de corte do primeiro pai com o do segundo pai.
  3. Em cada pai, nós fazemos a inversão das posições entre os elementos do mapeamento.

Mutação baseada em ordem

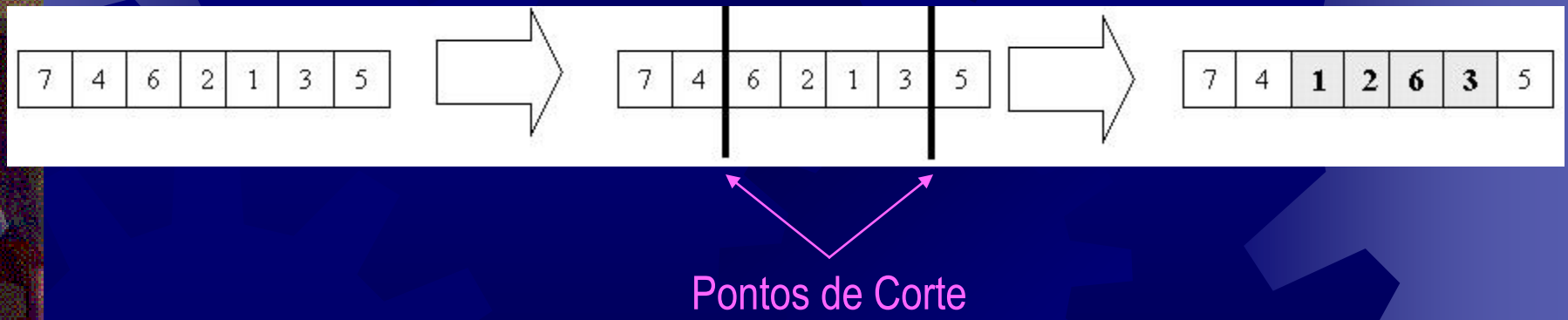
- Mutação realiza mudanças locais em cromossomos;
- Na representação baseada em ordem:
  - não há bits a inverter;
  - não podemos designar valores aleatoriamente pois poderíamos ter repetições de alguns nós enquanto outros ficariam de fora
  - temos que operar com diversos genes de um mesmo cromossomo simultaneamente.
- Existem três maneiras básicas de fazê-lo:
  - permutação de elementos;
  - mistura de sub-listas;
  - a inversão de sub-lista.

Mutação baseada em ordem

- permutação de elementos:
  - escolhem-se dois elementos ao acaso dentro do nosso cromossomo;
  - trocam-se as suas posições.

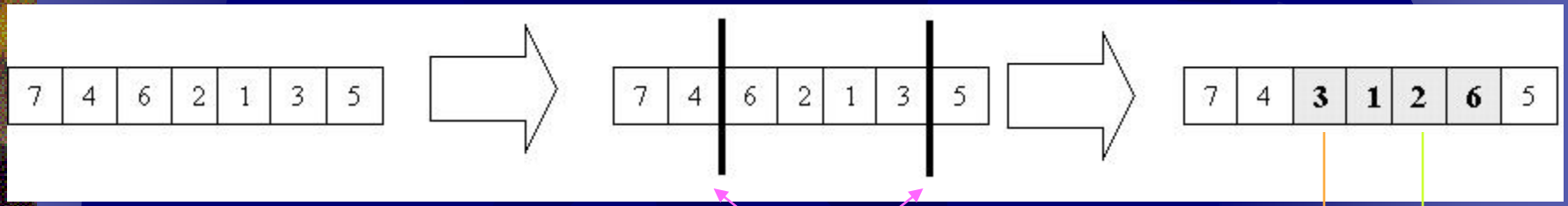
Mutação baseada em ordem

- Mistura de sublistas:
  - Escolhem-se dois pontos de corte dentro do nosso cromossomo;
  - Estes pontos delimitarão uma sub-lista.
  - Faz-se uma permutação aleatória dos elementos desta sub-lista.



Mutação baseada em ordem

- Inversão de sub-lista:
  - Similar à mistura de sub-lista;
  - Inverte-se a lista sorteada, ao invés de se realizar uma mistura aleatória dos seus elementos.



Outros operadores de mutação

- Idéia: substituir (ou combinar) os operadores puramente aleatórios por outros mutação que tenham embutidas funções de otimização local;
- A cada aplicação de um operador de mutação, será encontrado um máximo local da função de avaliação;
- Se usarmos elitismo, garantimos que o desempenho do GA será pelo menos igual a de um método de otimização local inicializado no ponto corrente.

Outros operadores de mutação

- Operador de otimização local é o chamado *2-opt*:
  - verifica todos os pares de nós ou cidades existentes dentro do cromossomo;
  - calcule qual seria a avaliação do cromossomo caso este par fosse invertido;
  - faça a inversão que garante o maior ganho de avaliação para o cromossomo corrente.
  - consiste em um algoritmo de *hill climbing*, otimizando a função de avaliação dentro desta vizinhança;
  - número de pares ordenados verificados:

$$C_2^n = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

Outros operadores de mutação

- Método pode ser expandido para avaliar combinações de mais cidades dentro da população, o que nos daria os métodos *3-opt*, *4-opt*, etc;
- Propiciam uma análise de uma vizinhança maior a cada iteração e por conseguinte, podem encontrar soluções mais promissoras;
- Quando aumentamos em um o número de elementos nas combinações, multiplicamos o número de indivíduos avaliados por um fator proporcional a *n*;
- Tamanho da vizinhança cresce rapidamente e faz com que o GA possa se tornar excessivamente lento.

Operador Inver-Over

- O operador Inver-Over é um operador que trabalha de forma isolada, substituindo tanto o operador de crossover quanto o de mutação;
- Ele aplica uma forte seleção sobre os indivíduos e usa um ou dois pais a cada iteração;
- Operador Inver-Over é aplicado a todos os pais, ao invés de uma seleção de pais feita por algum mecanismo aleatório;
- É equivalente, de certa forma, a um processo de busca local, otimizando as conexões localmente para melhorar o indivíduo corrente ao máximo.
- GAs que usam o operador Inver-Over costumam ser velozes e precisos, tendo atingido um resultado muito próximo da solução ótima em vários casos de teste;
  - não há um estudo teórico formal que prove que esta situação se repetirá em todos os problemas possíveis.

Operador Inver-Over

- Algoritmo básico:

```
Selecione um pai S_i da populaçao.
S' ← S_i
Selecione aleatoriamente um nó c de S'
Repita
    Selecione um número p_s aleatoriamente
    Se p_s<p então
        Sorteie um nó c' "à direita" de c
    Senão
        Sorteie outro indivíduo S_2 da população
        c' ← nó adjacente a c em S_2.
    Fim Se
```

(continua...)

Operador Inver-Over

- Algoritmo Básico (cont)

```
        Se c′ é adjacente a c então
                Saia do loop
        Fim Se
        Inverta a seleção do nó c até o nó c′ em S′
        c ← c′
        Se avaliação(S′) melhor que avaliação(S_i) então
                S_i ← S′
        Fim Se
Fim Repita
```