ESTADOS E BUSCAS III

Prof. Dr. Igor da Penha Natal

Busca local

Algoritmos de busca local e problemas de otimização

- Alguns espaços de busca são muito grandes para uma busca sistemática.
- Em muitos problemas de otimização o caminho até a solução é irrelevante, o estado objetivo é a solução.
  - Exemplo:
    - N-rainhas – o que importa é a configuração final e não a ordem em que as rainhas foram acrescentadas.
  - Exemplos de aplicações reais:
    - Projeto de CIs
    - Layout de instalações industriais
    - Escalonamento de jornadas de trabalho
    - Otimização de redes de telecomunicações
    - Roteamento de veículos

Algoritmos de busca local e problemas de otimização

- Os algoritmos de melhoramento interativo ou busca local operam sobre um conjunto de estados correntes (podendo ser somente um) ao invés de vários caminhos.
- Em geral a busca se move apenas para os vizinhos desse conjunto de estados.
- Vantagens:
  - Ocupam pouquíssima memória (normalmente um valor constante).
  - Podem encontrar soluções razoáveis em espaços de estados grandes ou infinitos, para os quais os algoritmos sistemáticos são inadequados.

Algoritmos de busca local

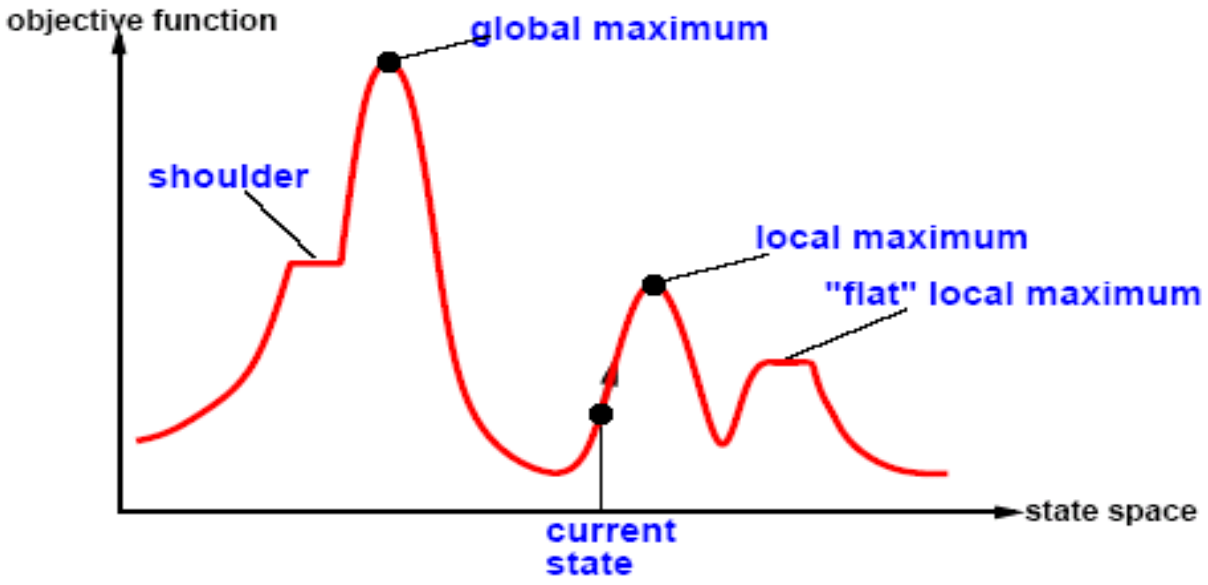
- Os algoritmos de busca local são úteis para resolver problemas de otimização puros, nos quais o objetivo é encontrar o melhor estado de acordo com uma função objetivo.

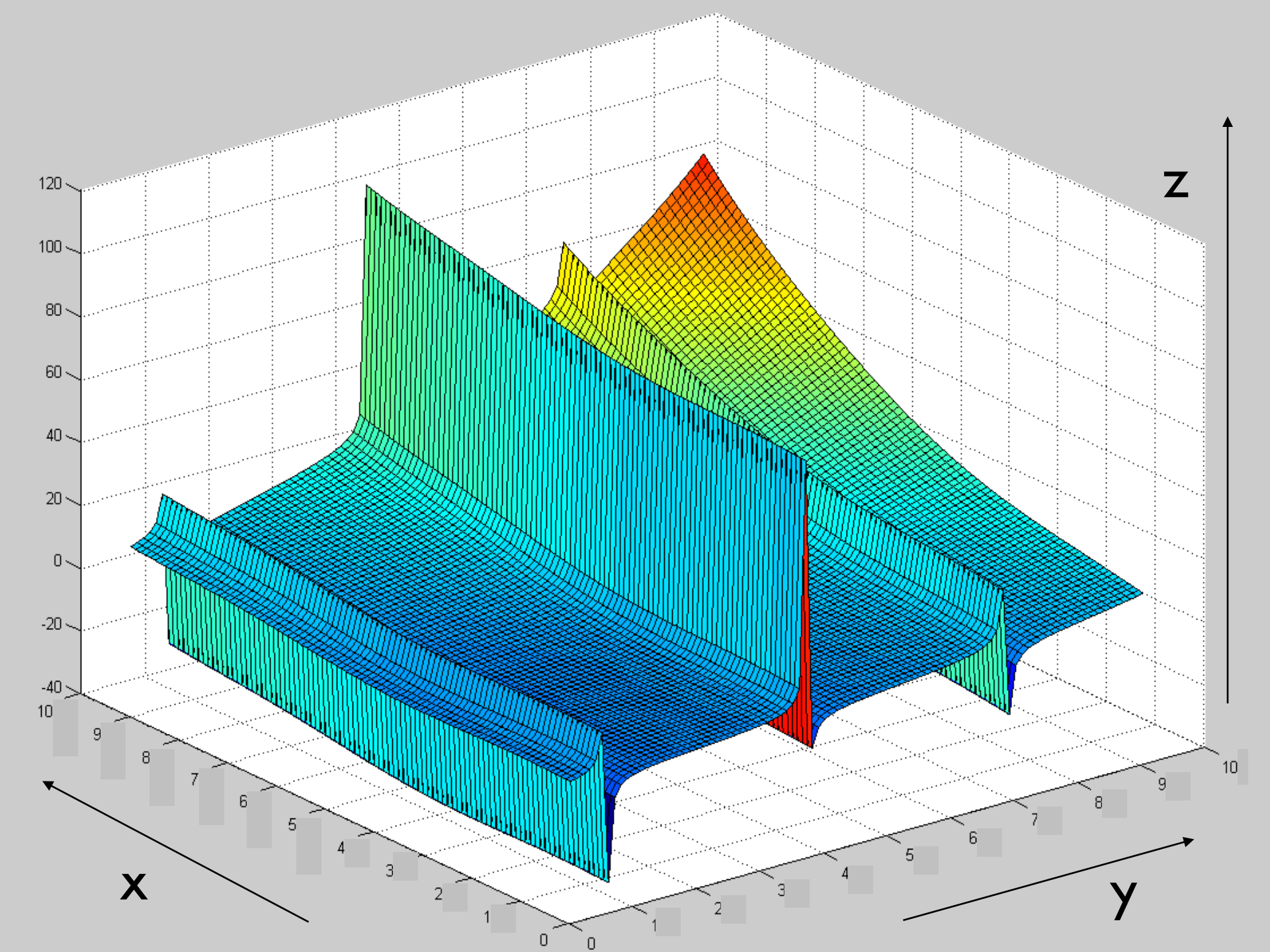
- O espaço de estados é, normalmente, considerado como tendo uma topologia onde existe:
  - Uma posição – definida pelo estado atual.
  - Uma elevação – definida pelo valor da função de custo da heurística ou da função objetivo no estado atual.

Algoritmos de busca local

- Se a elevação = custo → objetivo = mínimo global
- Se a elevação = função objetivo → objetivo = máximo global

- O algoritmo é **completo** se sempre encontra um objetivo.
- O algoritmo é **ótimo** se sempre encontra um mínimo/máximo **global**.




z
120
100
80
60
40
20
0
-20
-40
x
10
9
8
7
6
5
4
3
2
1
0
y
0
1
2
3
4
5
6
7
8
9
10

Algoritmo de busca de subida/descida de encosta (*Hill-Climbing*)

- Mantem um estado único que se e move de forma contínua no sentido do valor crescente/decrescente da função heurística.
- Termina quando alcança um pico/vale em que nenhum vizinho tem valor mais alto/baixo.
- Não mantém árvore de busca, somente o estado e o valor da função objetivo.
- Não examina antecipadamente valores de estados além de seus vizinhos imediatos (busca gulosa local).
- É análogo a subir/descer o Everest em meio a um nevoeiro e sofrendo de amnésia.

Busca de subida/descida de encosta (*Hill-Climbing*)

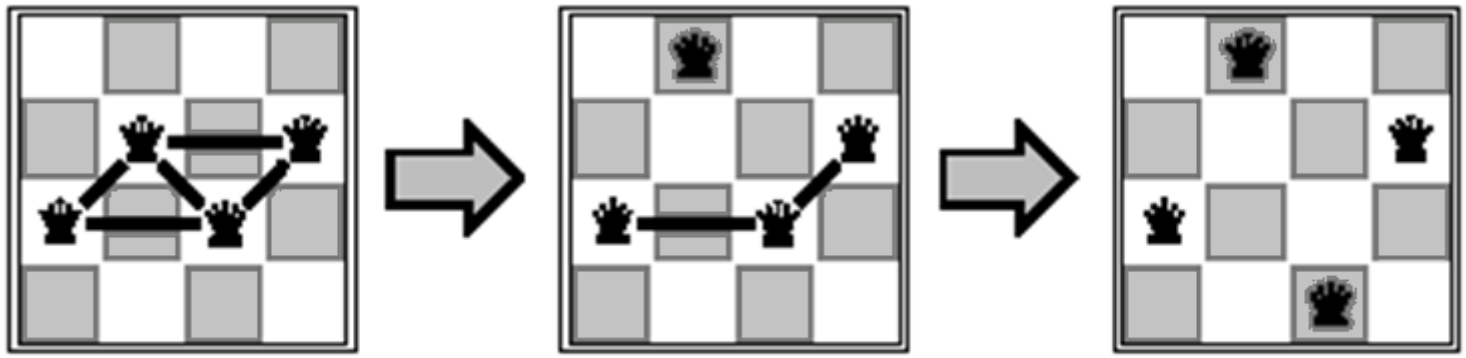

```
def Hill_Climbing(problema):
    entrada: problema # um problema
    saída: # um estado que é o máximo local
    local: atual # um nó
           vizinho # um nó

    atual <- criar_no(estado_inicial[problema])
    while:
          vizinho <- um sucessor atual com valor mais alto
          if valor[vizinho] <= valor[atual]:
                   return estado[atual]
          atual <- vizinho
```

Exemplo: busca de subida/descida de encosta para o problema da N-rainhas

- Os algoritmos de busca local utilizam uma formulação de estados completos.
  - E.g.: cada estado tem N rainhas, uma por coluna.
- A função sucessora gera todos os estados possíveis.
  - E.g.: Os estados são gerados pela movimentação de uma única rainha para outro lugar na mesma coluna.
- A função heurística é o números de pares de rainhas que estão se atacando umas às outras.
  - O mínimo global dessa função é zero, o que só ocorre em soluções perfeitas.

Exemplo: busca de subida/descida de encosta para o problema da N-rainhas

- Mova uma rainha para reduzir o número de conflitos.



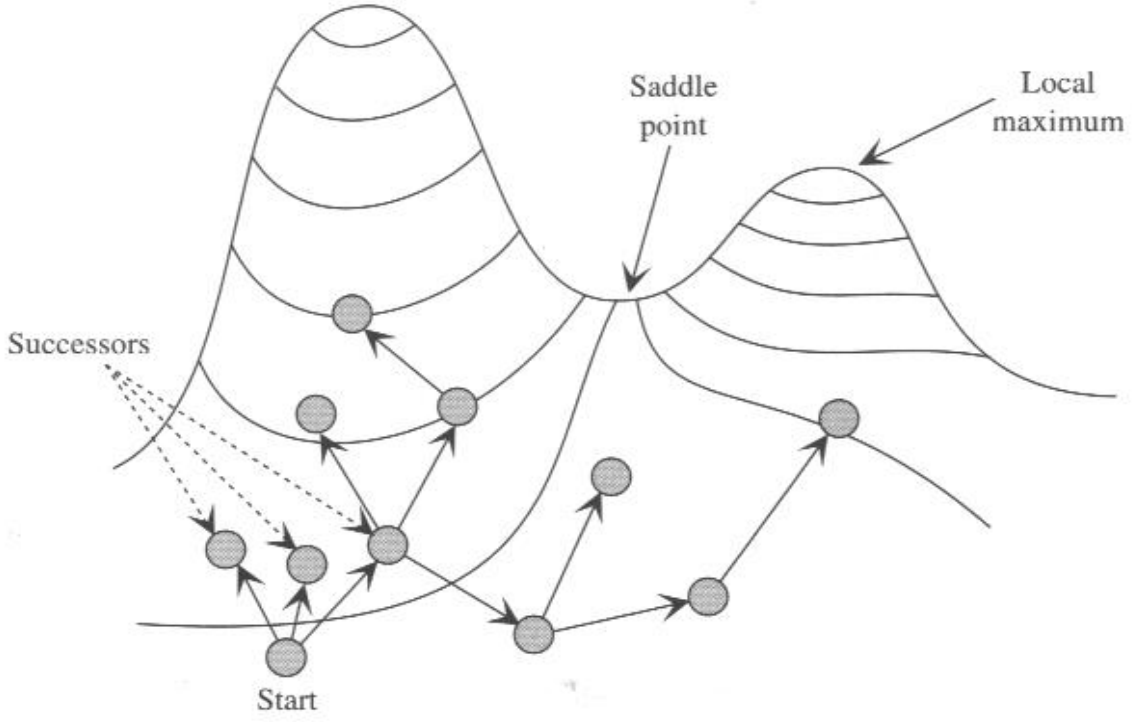
- Resolve o n-rainhas quase que instantaneamente para N muito grande, e.g. N = 1 milhão.

Exemplo: busca de subida/descida de encosta para o problema da N-rainhas

- A partir de um estado aleatório do problema:
  - Ficará paralisada 86% do tempo, resolvendo apenas 14% das instâncias do problema.
  - Usa em média 4 passos quando tem sucesso e 3 quando fica paralisada.
    - Nada mal para um espaço de estados com aproximadamente 17 milhões de estados.
- Se permitirmos movimentos laterais consecutivos (que devem ser limitados, a digamos 100):
  - Aumentamos a porcentagem de instâncias resolvidas de 14% para 94%.
  - Usa na média 21 passos quando tem sucesso e 64 quando fica paralisada.

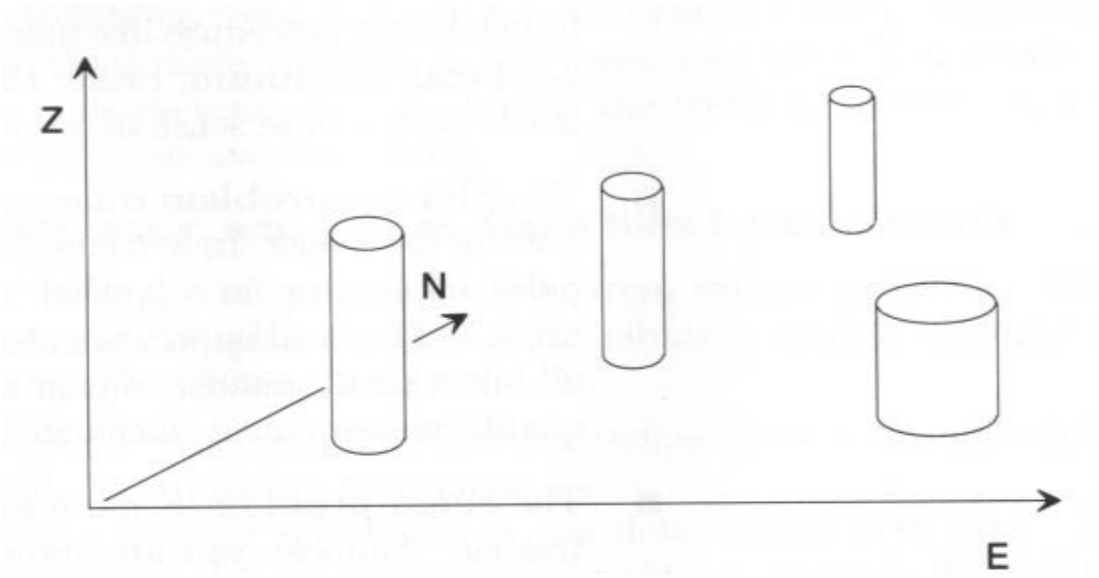
Busca de subida/descida de encosta - problemas

- Dependendo do estado inicial, pode ficar preso em um máximo local.



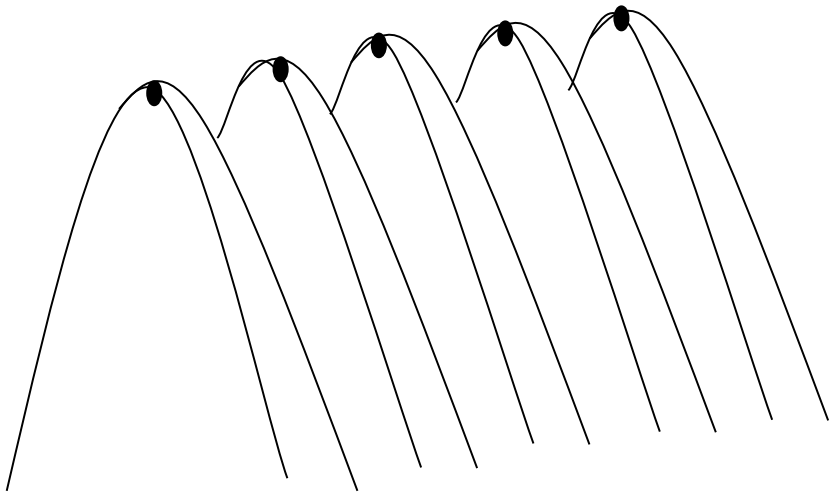
Busca de subida/descida de encosta - problemas

- Podem existir platôs fazendo com que em certas áreas a função tenha valores muito próximos e que, portanto, o algoritmo fique preso nesta região.



Busca de subida/descida de encosta - problemas

- Podem existir **cumes** (*ridges*) que fazem com que a função de qualidade oscile entre vários máximos locais.
- A técnica de usar n-passos de *look-ahead* pode ajudar.

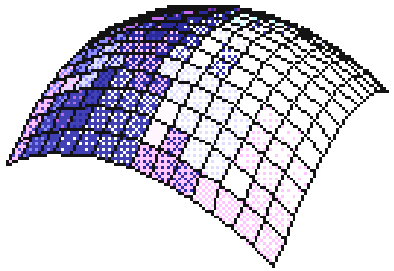


Busca de subida/descida de encosta – problemas

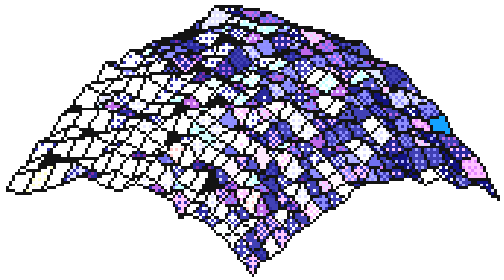
Como podemos melhorar o *hill-climbing*?

Reinícios aleatórios!

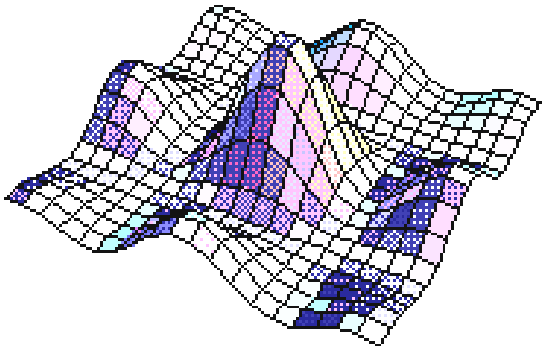
Intuição: chame o algoritmo de *hill-climbing* tantas vezes quanto for possível, escolha a melhor resposta.



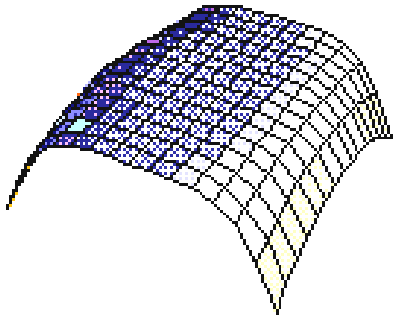
A



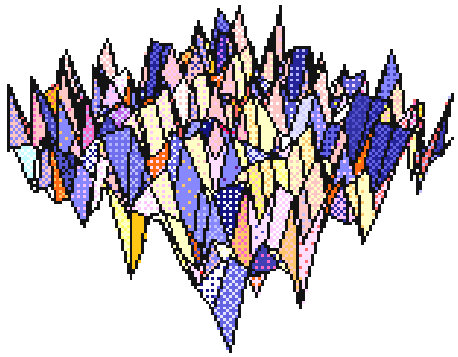
B



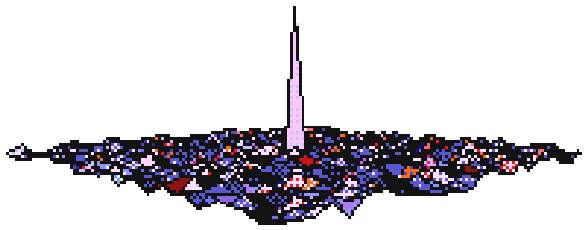
C



D



E



F

Variantes da subida/descida de encosta

- Subida/descida de encosta **estocástica.**
  - Escolhe ao acaso um estado vizinho com valor melhor do que o estado corrente.
  - Converge mais lentamente, mas encontra soluções melhores.
- Subida/descida de encosta pela **primeira escolha.**
  - Gera sucessores ao acaso, até gerar um sucessor melhor do que o estado corrente.
  - É uma boa estratégia quando um estado tem muitos sucessores.
- Subida/descida de encosta com **reinício aleatório.**
  - Faz-se uma série de buscas a partir de estados iniciais gerados aleatoriamente.

Algoritmo de busca de subida/descida de encosta - desempenho

- O sucesso deste tipo de busca depende muito da topologia do espaço de estados.
  - Muitos problemas reais tem uma topologia mais parecida com uma família de ouriços em um piso plano, com ouriços em miniatura vivendo na ponto de cada espinho de um ouriço, *ad infinitum*.
- Os problemas NP-difíceis têm um número exponencial de máximos locais, nos quais a subida de encosta fica paralisada.

Algoritmos aleatórios

- Considere dois métodos para encontrar um valor máximo/mínimo:
  - Subida/descida gulosa, a partir de alguma posição, mantendo-se em movimento para cima/baixo & relate o valor máximo/mínimo encontrado.
  - Escolher valores aleatoriamente & relate o valor máximo/mínimo encontrado.
- O que você espera funcionar melhor para encontrar um mínimo global?
- Uma mistura pode funcionar melhor?

Subida/descida de encosta gulosa aleatória

- Como passos ascendentes/descendentes podemos permitir:
  - Passos aleatórios: mover-se para um vizinho aleatório.
  - Reinicialização aleatória: reatribuir valores aleatórios para todas as variáveis.
- O que é mais caro computacionalmente?

Busca estocástica local

- Busca estocástica local é uma mistura de:
  - **Subida/descida gulosa**: mover-se para vizinho maior/menor.
  - **Passeio aleatório**: tomar alguns passos aleatórios.
  - **Reinicialização aleatória**: reatribuir valores para todas as variáveis.

Algoritmo de busca por *Simulated Annealing* (Têmpera simulada)

- Têmpera: processo usado para temperar ou endurecer metais e vidro aquecendo-os a alta temperatura e depois resfriando gradualmente.
- Ideia:
  - Fugir do máximo local permitindo alguns movimentos "ruins" para fora do máximo, mas gradualmente decrescendo seu número e frequência.
- A temperatura diminui em função do tempo, diminuindo assim a probabilidade de se escolher um estado pior.
- É amplamente utilizado para *layout* de *VLSI*, planejamento de linhas aéreas, etc.

Algoritmo de busca por *Simulated Annealing*

- Selecione uma variável ao acaso e um novo valor ao acaso.
- Se for uma melhoria, adote-o.
- Se não for uma melhoria, adotá-lo probabilisticamente dependendo de um parâmetro de temperatura, **T**.
  - Com uma atribuição atual **n** e a atribuição proposta $\mathbf{n_0}$ passamos para $\mathbf{n_0}$ com probabilidade $e^{(h(n')-h(n))/T}$
- A temperatura pode ser reduzida. Probabilidade de aceitar uma alteração:

Temperature	1-worse	2-worse	3-worse
10	0.91	0.81	0.74
1	0.37	0.14	0.05
0.25	0.02	0.0003	0.000005
0.1	0.00005	0	0

Propriedades do *Simulated Annealing*

- Na "temperatura" fixa ***T***, a probabilidade de ocupação de um estado pior que o atual é:

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

- Com ***T*** decrescendo suficientemente lento $\Rightarrow$ sempre alcança o melhor estado.

- Para valores maiores de ***T***, soluções ruins são permitidas.

- Com ***T*** próximo de zero, a probabilidade de se escolher soluções ruins diminui.

- ***E(x)*** determina qual é a variação entre a solução corrente e a próxima solução.

Simulated Annealing


```
def simulated_annealing(problema, escala)
entradas: problema # um problema
          escala # um mapeamento do tempo pela temperatura.
saída: # um estado solução
local: atual  # um nó
       próximo  # um nó
       T  # uma temperatura controlando a probabilidade de dar passos ruins

atual ← cria_nó(estado_inicial[problema])
for tempo = 1 to ∞:
   T ← escala[tempo]
   if T = 0: return atual
   próximo ← um sucessor de atual aleatoriamente selecionado
   ΔE ← (h(n') – h(n))
   if ΔE > 0: atual ← próximo
   else: atual ← próximo somente com uma probabilidade e^(ΔE/T)
end
```

Listas *Tabu*

- Para evitar ficar em ciclos pode-se manter uma lista Tabu das últimas ***k*** atribuições.
- Para evitar os ciclos, não permita que uma atribuição que já está na lista Tabu.
- Se $k = 1$, não permitimos uma atribuição do mesmo valor para a variável escolhida.
- Podemos implementá-lo mais eficientemente do que uma lista de atribuições completas.
- Este método pode ser caro se ***k*** é grande.

Busca paralela

- Uma atribuição total é chamada de um indivíduo.
  - Ideia:
    - Manter uma população de *k* indivíduos em vez de um.
  - Em cada fase, atualize cada indivíduo na população.
  - Sempre que um indivíduo é uma solução, ele pode ser reportado.
  - É como *k* reinícios, mas usa *k* vezes o número mínimo de passos.

Busca em feixe local

- Mantém o controle de ***k*** estados ao invés de somente um.
  - Começa com ***k*** estados gerados aleatoriamente.
  - Em cada passo gera todos os sucessores dos ***k*** estados.
  - Se algum sucessor for o objetivo, termina.
  - Se não escolhe os ***k*** melhores sucessores e repete a ação.
- É diferente da busca com reinício aleatório porque os ***k*** estados compartilham informações entre eles.
- Quando $k = 1$, ele é descendente/ascendente gulosa.
- Quando $k = \infty$, é busca de largura.
- O valor de ***k*** nos permite limitar o espaço e o paralelismo.

Busca em feixe local

- Problema:
  - Os ***k*** estados podem rapidamente ficar concentrados em uma pequena região do espaço de estados.
- Solução:
  - Busca em feixe estocástica - escolhe ***k*** sucessores melhores que seus pais ao acaso.

Busca em feixe estocástica

- Como o feixe de busca, mas ela escolhe probabilisticamente os ***k*** indivíduos na próxima geração.
- A probabilidade de um vizinho ser escolhido é proporcional ao seu valor heurístico.
- Isso mantém a diversidade entre os indivíduos.
- O valor heurístico reflete a adequação do indivíduo.
- Como na reprodução assexuada: cada indivíduo sofre mutações e aqueles mais adaptados sobrevivem.

Algoritmo genético

- Algoritmo genético - ***k*** estados melhores do que os seus pais são gerados.
  - Um estado é gerado pela combinação de **dois** ou **mais** estados pais.
  - Para cada geração:
    - Escolha aleatoriamente pares de indivíduos nos quais os indivíduos mais aptos são mais propensos a serem escolhidos.
    - Para cada par, realizar um *crossover*: formar duas crias cada, tomando partes diferentes de seus pais.
    - Mutar alguns valores.
- Analogia com a seleção natural por reprodução sexuada.

Indivíduo e população

- Normalmente começam com um conjunto de ***k*** estados gerados aleatoriamente chamado de **população.**
  - Quando possível, o conhecimento do problema pode ser utilizado para definir a população inicial.

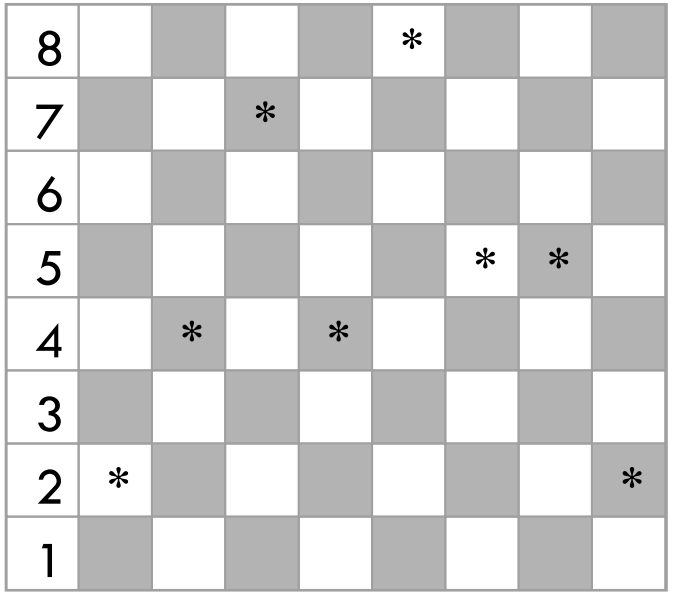
- Um estado é chamado de **indivíduo**, ou **cromossomo.**
  - É uma estrutura de dados que representa uma possível solução para o problema de forma não ambígua.
  - É normalmente representado por uma cadeia de valores:
    - Vetores de reais, (2.345, 4.3454, 5.1, 3.4)
    - Cadeias de bits, (111011011)
    - Vetores de inteiros, (1,4,2,5,2,8)
    - ou outra estrutura de dados.

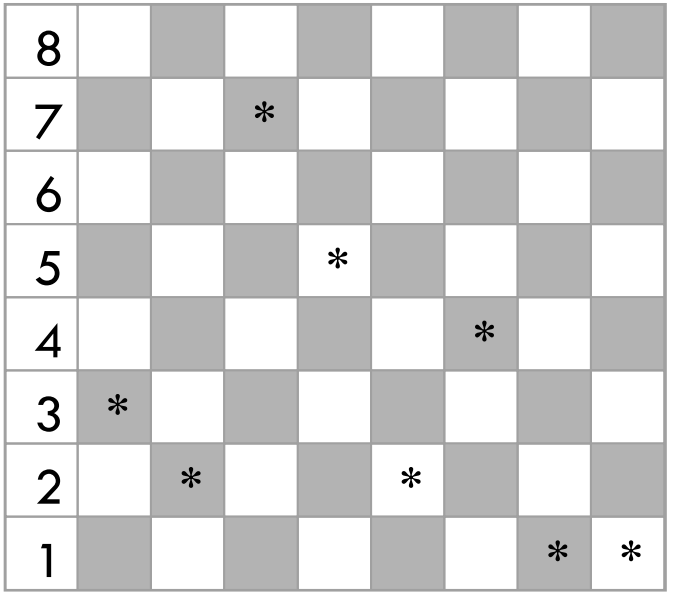
Exemplo: um cromossomo para o problema das 8-rainhas

- Deve especificar a posição das 8 rainhas, cada uma em uma coluna de 8 quadrados.
- Pode ser representado por 8 dígitos, variando de 1 a 8.
  - 2 4 7 4 8 5 5 2 ou 3 2 7 5 2 4 1 1
- Ou por uma cadeia de 24 bits = cada 3 bits = 1 posição.
  - 001|011|110|011|111|100|100|001
  - 010|001|110|100|001|011|000|000

Exemplo: indivíduo para o problema da 8-rainhas

- 2 4 7 4 8 5 5 2 = 001|011|110|011|111|100|100|001
- 3 2 7 5 2 4 1 1 = 010|001|110|100|001|011|000|000





Função de aptidão (*fitness*)

- Cada estado (ou indivíduo) é avaliado pela função de avaliação – chamada de **função de *fitness*.**
- Aptidão pode ser:
  - Igual a função objetivo.
  - Baseado no **ranking** do indivíduo da população.
- Quanto melhor o estado – maior é o valor da função *fitness*.
  - Ex.: das 8 rainhas: nº de pares de rainhas não atacantes (solução = 28)

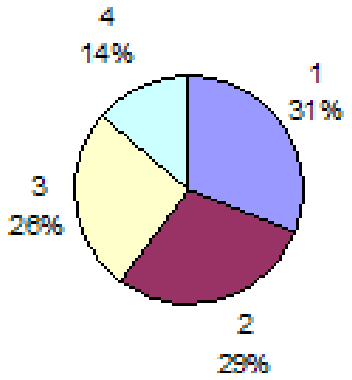
Nº	Cromossomo	*f*(n)	ranking (total = 100)
1	2 4 7 4 8 5 5 2	24	31
2	3 2 7 5 2 4 1 1	23	29
3	2 4 4 1 5 1 2 4	20	26
4	3 2 5 4 3 2 1 3	11	14

Seleção dos indivíduos para reprodução

- Normalmente os melhores indivíduos (maior aptidão) são selecionados para gerar filhos.
- Objetivos:
  - Propagar material genético dos indivíduos mais adaptados.
  - Dirigir a busca para as melhores regiões do espaço de estados.
- Tipos mais comuns de seleção.
  - **Proporcional a aptidão** (roleta)
    - Indivíduos com maior aptidão tem maior probabilidade de serem selecionados.
  - **Torneio**
    - Seleciona *n* (tipicamente 2) indivíduos aleatoriamente da população e o melhor é selecionado.
  - **Ranking** (os n mais adaptados)
    - Seleciona os *n* indivíduos mais adaptados.

Seleção para o problema das 8-rainhas

- Baseando-se no ranking dos indivíduos na população, temos as seguintes probabilidades de escolha:
  - 1 → 2 4 7 4 8 5 5 2 = 24 → 31%
  - 2 → 3 2 7 5 2 4 1 1= 23 → 29%
  - 3 → 2 4 4 1 5 1 2 4 = 20 → 26%
  - 4 → 3 2 5 4 3 2 1 3 = 11 → 14%



- Vamos supor que usando o **método da roleta** foram selecionados os indivíduos:
  - 1 → 2 4 7 4 8 5 5 2 = 24 → 31%
  - 2 → 3 2 7 5 2 4 1 1= 23 → 29%

Reprodução dos indivíduos selecionados - *Crossover*

- Cria novos indivíduos **misturando** características de dois ou mais indivíduos pais (***crossover***) – variação.
- Objetivos:
  - Combinar e/ou perpetuar material genético dos indivíduos mais adaptados.
  - Fazer com que o algoritmo genético **explore estados longe dos estados pais**, no começo da execução.
    - À medida em que os melhores indivíduos ficam na população, a probabilidade de gerar um filho longe dos pais, diminui.
  - É um dos principais mecanismos de busca do AG.

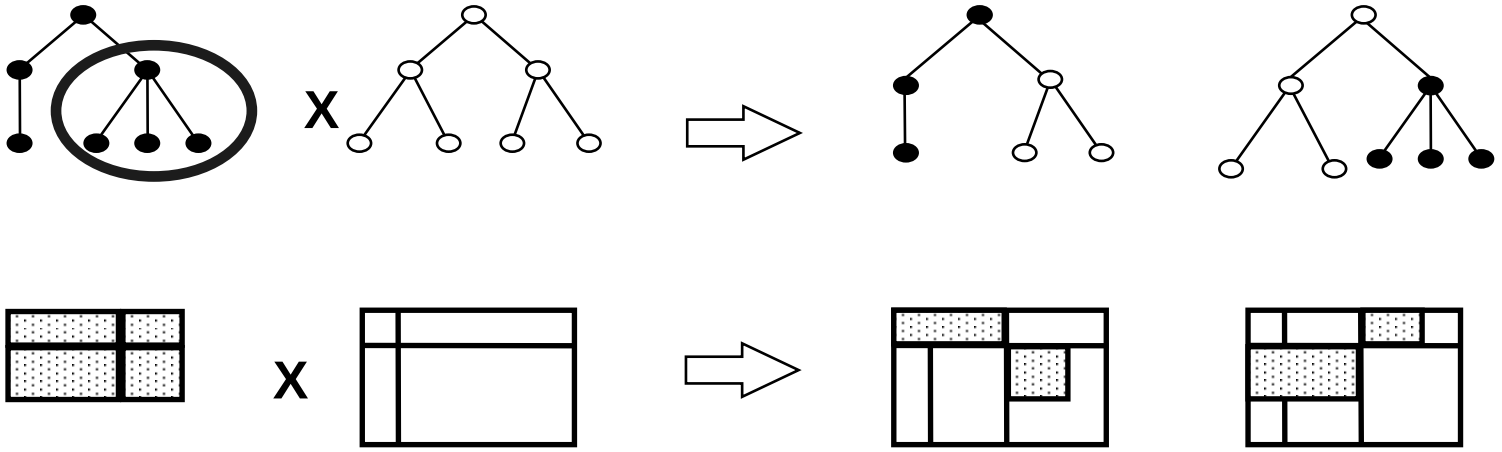
Reprodução dos indivíduos selecionados

Escolha do ponto de *crossover*

- O *crossover* normalmente é feito dividindo os indivíduos selecionados em um ponto.
  - Mas pode ser feito em mais de um ponto.
  - A escolha dos pontos ponde ser fixa ou aleatória para cada iteração do algoritmo.
- Escolha do ponto de *crossover* para os filhos selecionados no problema das 8-rainhas.
  - 1 => 2 4 7 | 4 8 5 5 2
  - 2 => 3 2 7 | 5 2 4 1 1
- Formação dos filhos pelo processo de *crossover* em um ponto:
  - Filho 1 => 2 4 7 5 2 4 1 1
  - Filho 2 => 3 2 7 4 8 5 5 2

Crossover e a representação do cromossomo

- Quanto **mais estruturada** for a representação do cromossomo, **mais custosa** se torna a operação de *crossover*, mutação e análise do indivíduo.
  - Para alguns problemas se faz necessária também uma operação que analisa se o indivíduo gerado é realmente uma possível solução ou não.

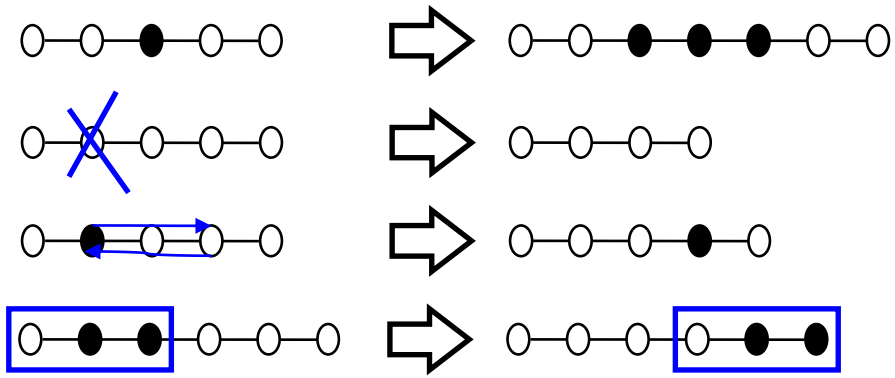


Mutação nos filhos gerados

- Os indivíduos gerados podem sofrer **mutação** com uma pequena probabilidade.
- Objetivo:
  - A ideia é que quando os pais são muito parecidos, a mutação possa trazer alguma característica que não está presente nos indivíduos da população para ajudar a escapar do ótimo local.
- A **taxa de mutação** normalmente é **pequena** (menos de 1% dos indivíduos) e pode diminuir com o tempo para garantir a convergência.

Tipos de mutação

- Gerativa
- Destrutiva
- *Swap*
- *Swap* de sequência



- Alguns problemas simplesmente não suportam alguns tipos de mutação como, por exemplo, a gerativa e a destrutiva.
- Exemplo de mutação modificativa para o problema das 8-rainhas:
  - Filho 2 antes da mutação ➔ 3 2 7 4 8 5 5 2
  - Filho 2 após a mutação ➔ 3 2 7 4 8 **3** 5 2

Adição dos filhos à nova população

- Objetivo:
  - Garantir uma convergência adequada.
- Tipos:
  - Simples: a nova geração **substitui** a antiga.
  - Elitista ou *steady-state*: a nova geração se **mistura** com a antiga.
- Critérios de substituição no caso elitista:
  - os piores.
  - os mais semelhantes.
    - para evitar convergência prematura.
  - os melhores.
  - os pais.
  - aleatoriamente, ...

Adição dos filhos à nova população

- Pontos a considerar:
  - A substituição simples da geração antiga pela nova pode destruir o melhor indivíduo.
    - Por que perder a melhor solução encontrada?
  - Elitismo transfere cópias dos melhores indivíduos para a geração seguinte.
    - Mas normalmente, **converge muito rápido** podendo ficar preso em um máximo/mínimo local.
    - Mesmo assim, **tem resultados melhores** do que a substituição simples.

Critérios de parada

- Número de gerações.
- Encontrou a solução (quando esta é conhecida).
- Perda de diversidade (estagnação).
  - Muitos indivíduos com características e função de aptidão semelhantes.
- Convergência
  - Nas últimas ***k*** gerações não houve melhora na aptidão do melhore indivíduo.

Algoritmo genético - geral

```
def busca_por_Algoritmo_Genético(população, FN-FITNESS):
entradas: população, um conjunto de indivíduos
          FN_FITNESS, uma função que mede a adaptação de um indivíduo
          N, quantidade de filhos gerados em cada geração

while True:
   nova_população <- {}
   for i=1 to N:
      x <- SELEÇÃO(população, FN-FITNESS)
      y <- SELEÇÃO(população, FN-FITNESS)
       filho <- REPRODUZ(x,y)
      if (pequena probabilidade aleatória):
         filho <- MUTAÇÃO(filho)
       adicionar filho à nova_população
    exitif algum critério de parada
return o melhor indivíduo da população, de acordo com FN-FITNESS
```

Algoritmo genético - vantagens

- Troca informações entre processos de busca paralelos.
- A principal vantagem vem da operação de *crossover*:
  - Combina grandes blocos de genes que evoluem de forma independente para executar funções úteis.
    - Ex.: a colocação da três primeiras rainhas nas posições 2, 4 e 6 (em que elas não se atacam as outras) constitui um bloco útil.
  - Estes blocos podem ser combinados com outros, para formar uma solução.

Algoritmo genético

- A combinação de blocos úteis funciona usando a ideia de esquema.
- Um esquema é uma subcadeia na qual algumas posições podem ser deixadas sem especificação
  - Ex: 246*****
    - Cadeias do tipo 24625176 são chamadas instâncias do problema.

Questões centrais

- Como representar os indivíduos?
- Quem é a população inicial?
- Como definir a função objetivo?
- Quais são os critérios de seleção?
- Como aplicar/definir o operador de reprodução?
- Como aplicar/definir o operador de mutação?
- Como garantir a convergência e ao mesmo tempo obter a solução ótima?

Exemplo 1

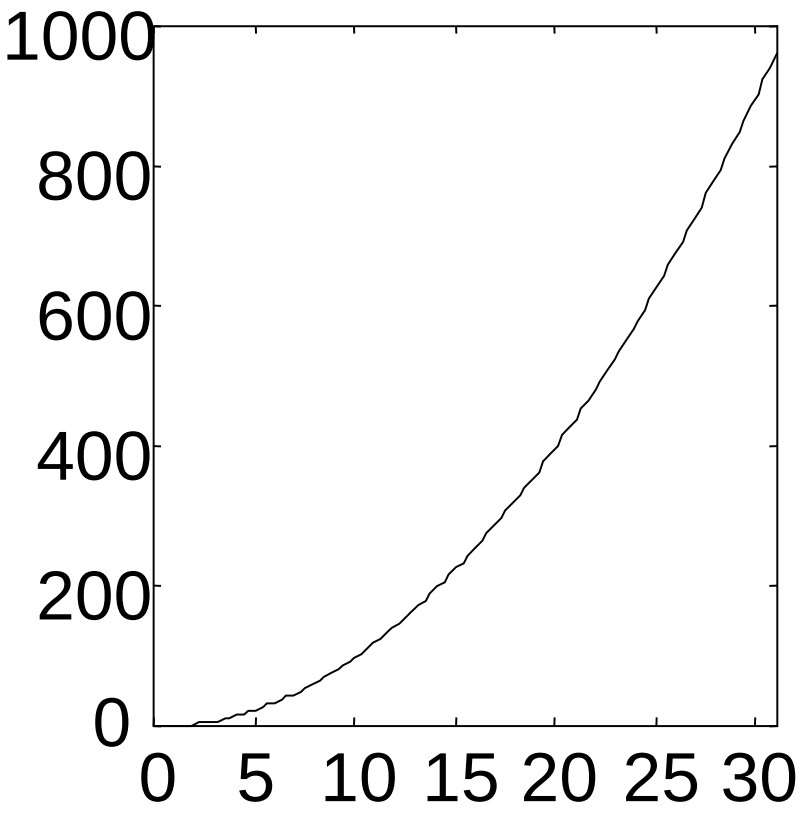
- Problema: Use um AG para encontrar o ponto máximo da função:

$$f(x)=x^2$$

- com **x** sujeito as seguintes restrições:

$$0 \leq x \leq 31$$

x é inteiro.



Cromossomo

- Cromossomos binários com *5* bits:
  - 0 = 00000
  - 31 = 11111
- Função de aptidão: pode ser a própria função objetivo.
  - Exemplo: aptidão(00011) = $f(3) = 9$

População Inicial

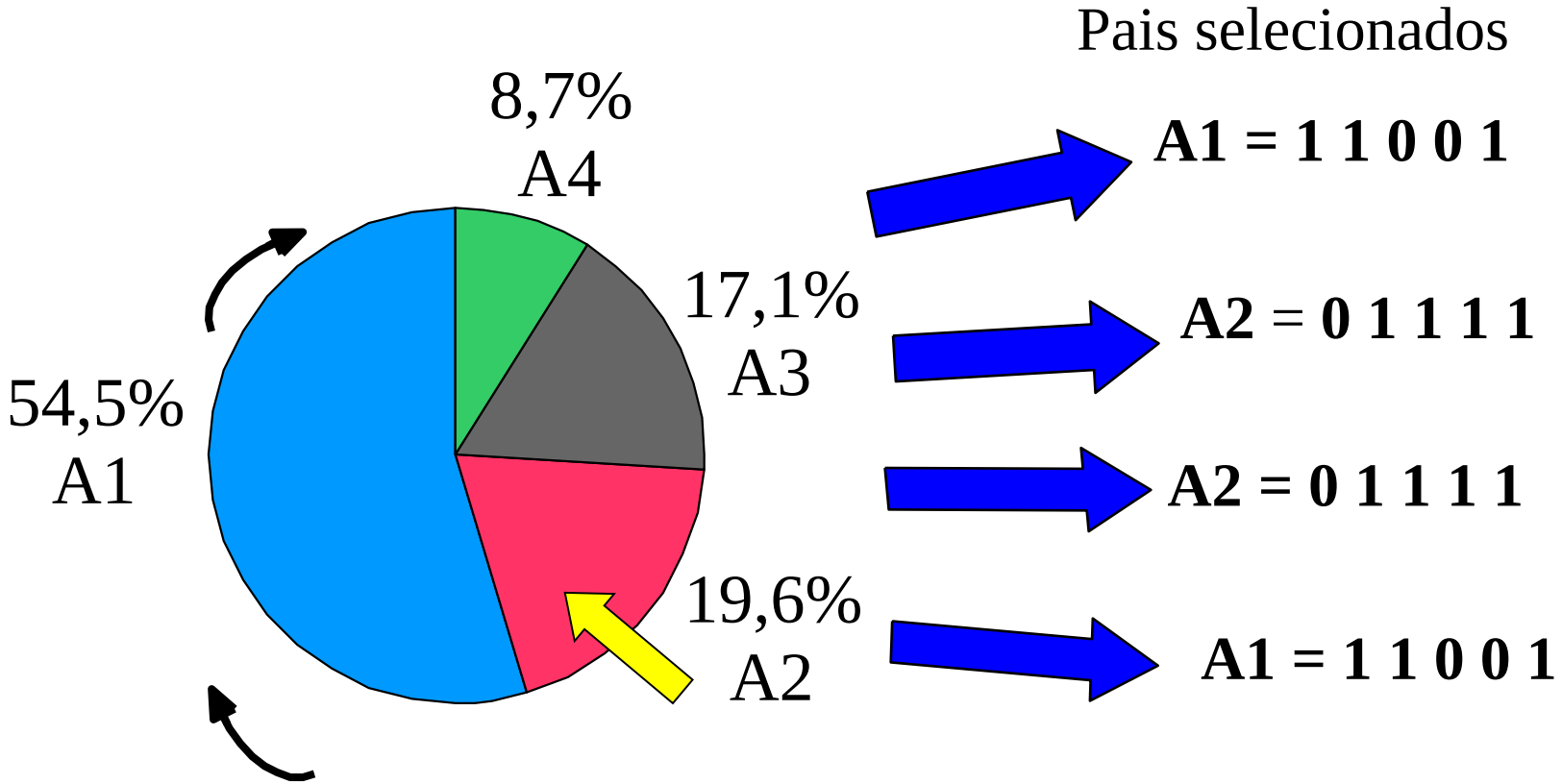
População gerada de forma aleatória: $p_i = \frac{f(x_i)}{\sum_{k=1}^{N} f(x_k)}$

Pop. inicial

cromossomos	x	$f(x)$	
A_1 = 1 1 0 0 1	25	625	54,5%
A_2 = 0 1 1 1 1	15	225	19,6%
A_3 = 0 1 1 1 0	14	196	17,1%
A_4 = 0 1 0 1 0	10	100	8,7%

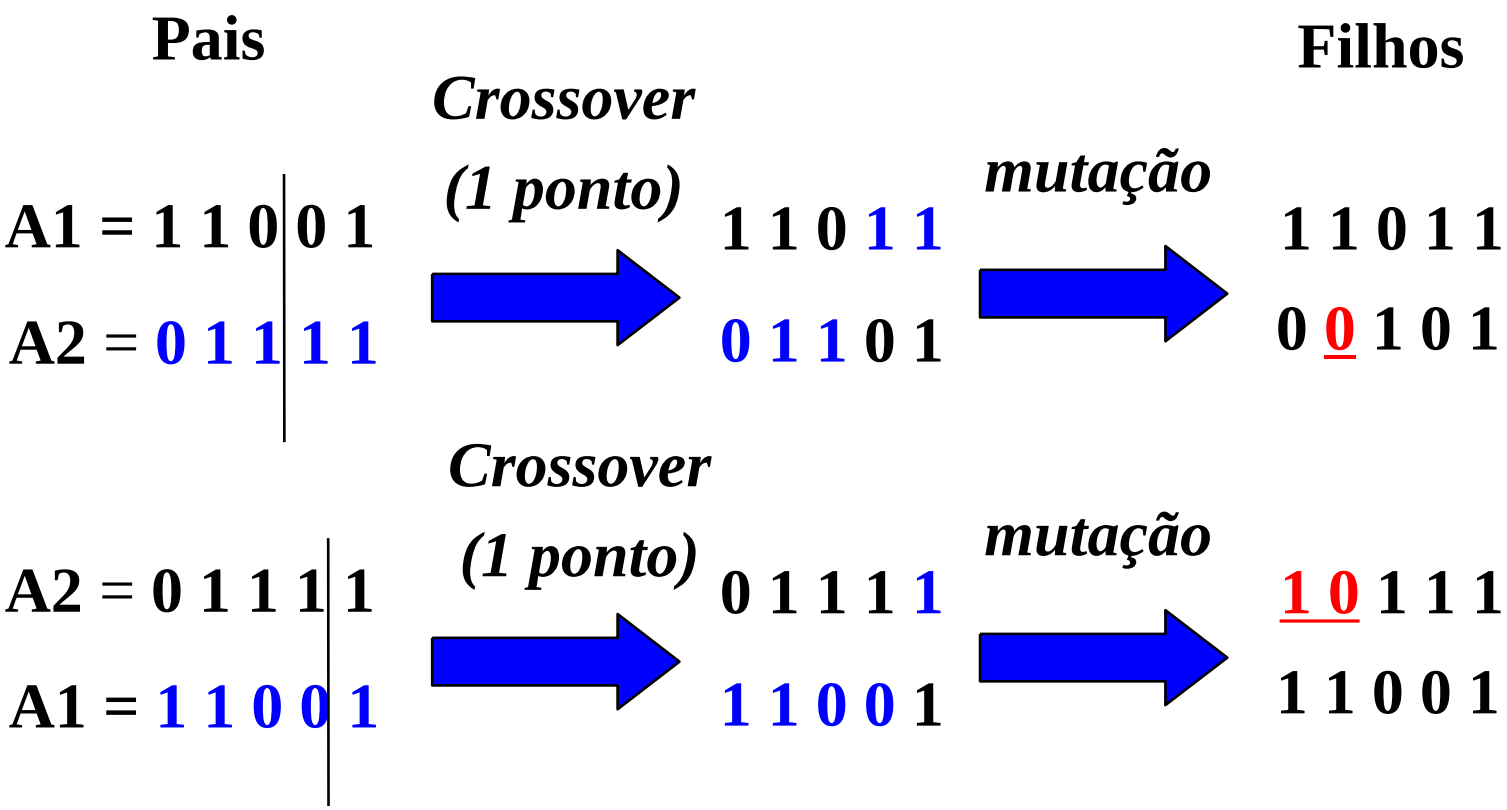
Probabilidade de seleção é proporcional a aptidão (roleta).

Seleção proporcional a aptidão (Roleta)



Problema: converge muito rápido por causa da variação pequena.

Crossover e mutação



A primeira geração

- Adição dos filhos à nova população : substituição simples.

	cromossomos	x	$f(x)$	prob. de seleção
1	1 1 0 1 1	27	729	29,1%
2	1 1 0 0 1	25	625	24,9%
3	1 1 0 0 1	25	625	24,9%
4	1 0 1 1 1	23	529	21,1%

As demais gerações

Segunda Geração

		x	$f(x)$
1	1 1 0 1 1	27	729
2	1 1 0 0 0	24	576
3	1 0 1 1 1	23	529
4	1 0 1 0 1	21	441

Terceira Geração

		x	$f(x)$
1	1 1 0 1 1	27	729
2	1 0 1 1 1	23	529
3	0 1 1 1 1	15	225
4	0 0 1 1 1	7	49

As demais gerações

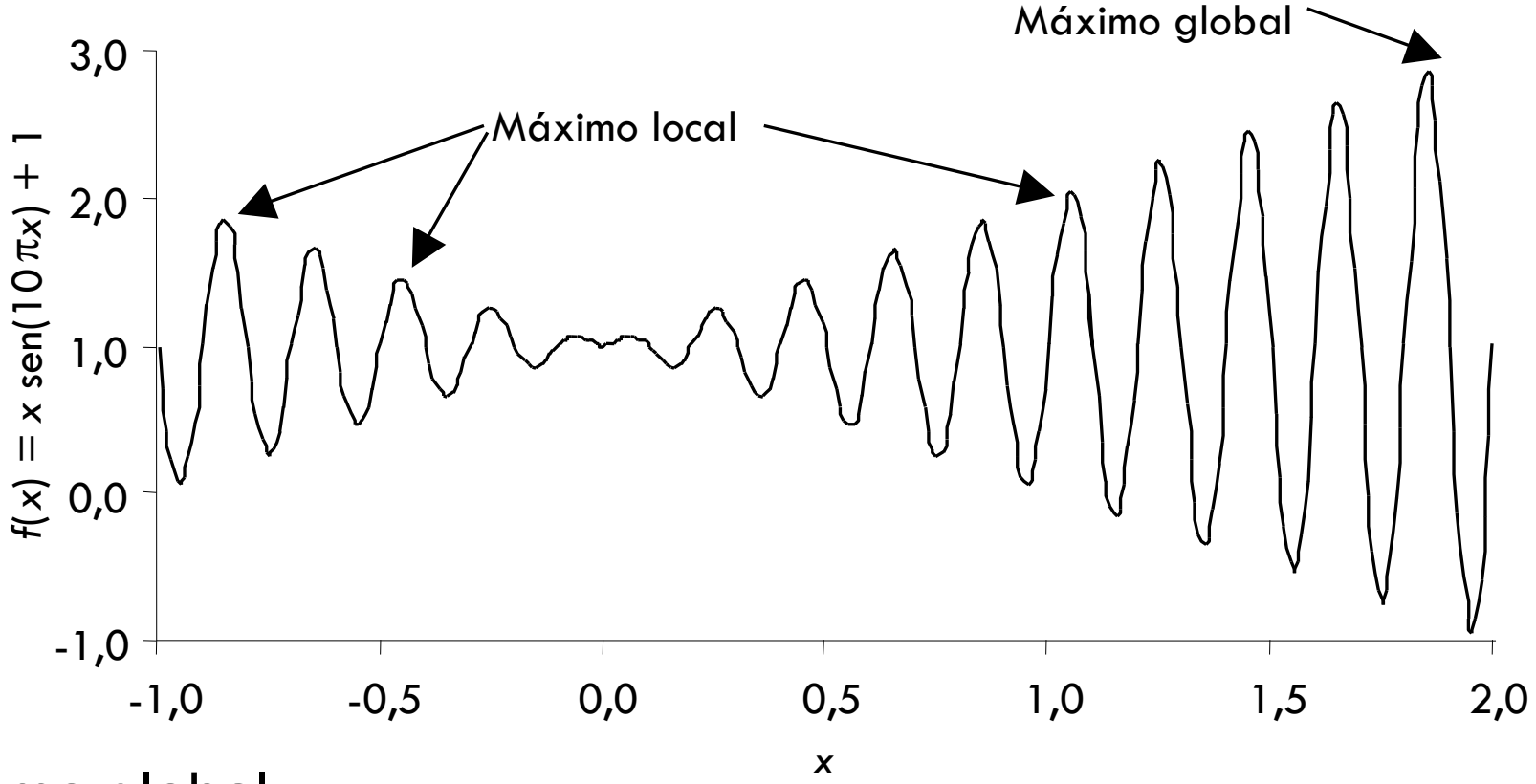
Quarta Geração

		x	$f(x)$
1	1 1 1 1 1	31	961
2	1 1 0 1 1	27	729
3	1 0 1 1 1	23	529
4	1 0 1 1 1	23	529

Quinta Geração

		x	$f(x)$
1	1 1 1 1 1	31	961
2	1 1 1 1 1	31	961
3	1 1 1 1 1	31	961
4	1 0 1 1 1	23	529

Problema 2



Máximo global:

x = 1,85055

f(x) = 2,85027

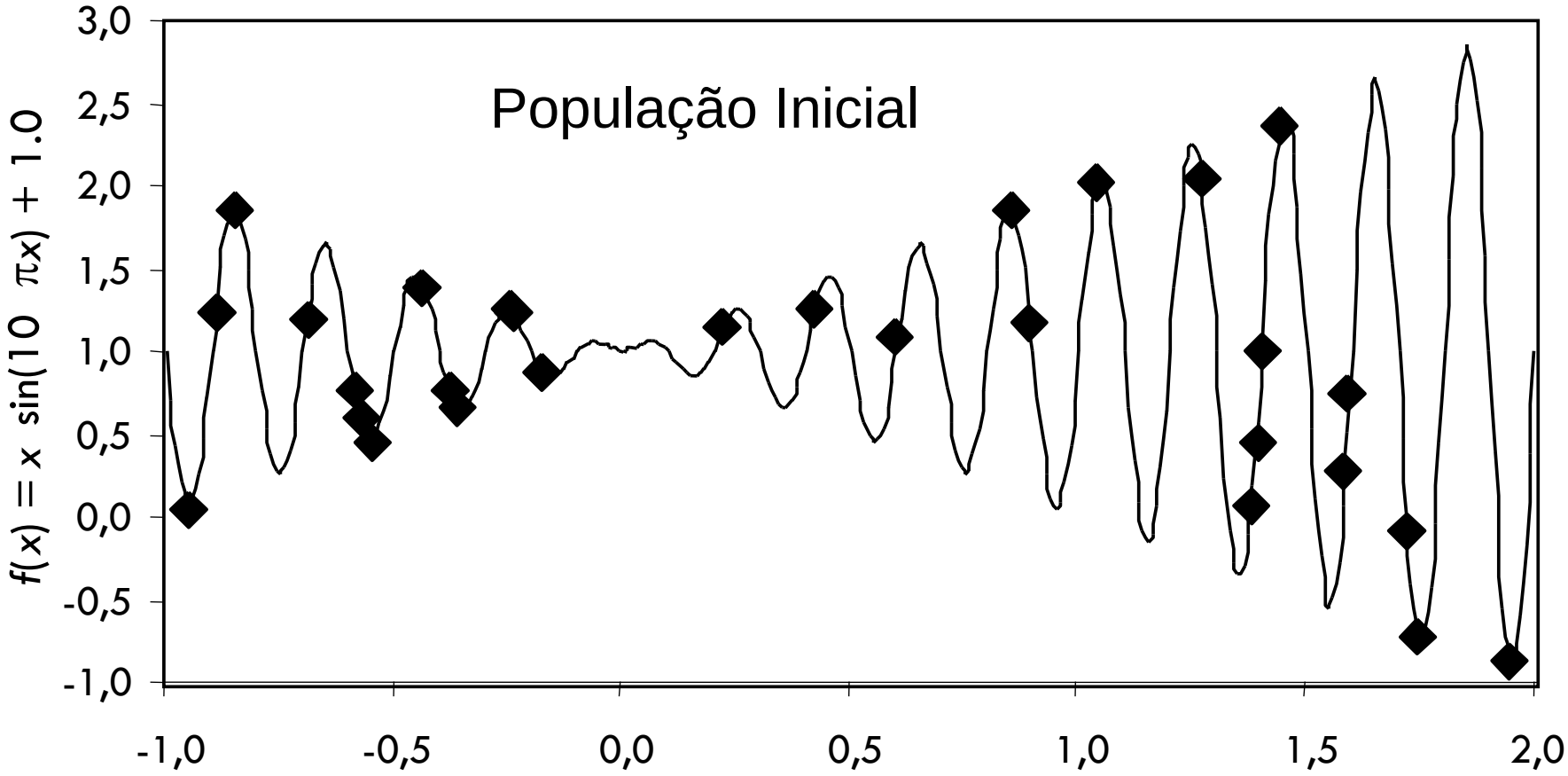
Problema 2

- Função multimodal com vários pontos de máximo.
- É um problema de otimização global (encontrar o máximo global).
- Não pode ser resolvido pela grande maioria dos métodos de otimização convencional.
- Há muitos métodos de otimização local, mas para otimização global são poucos.

O cromossomo do problema 2

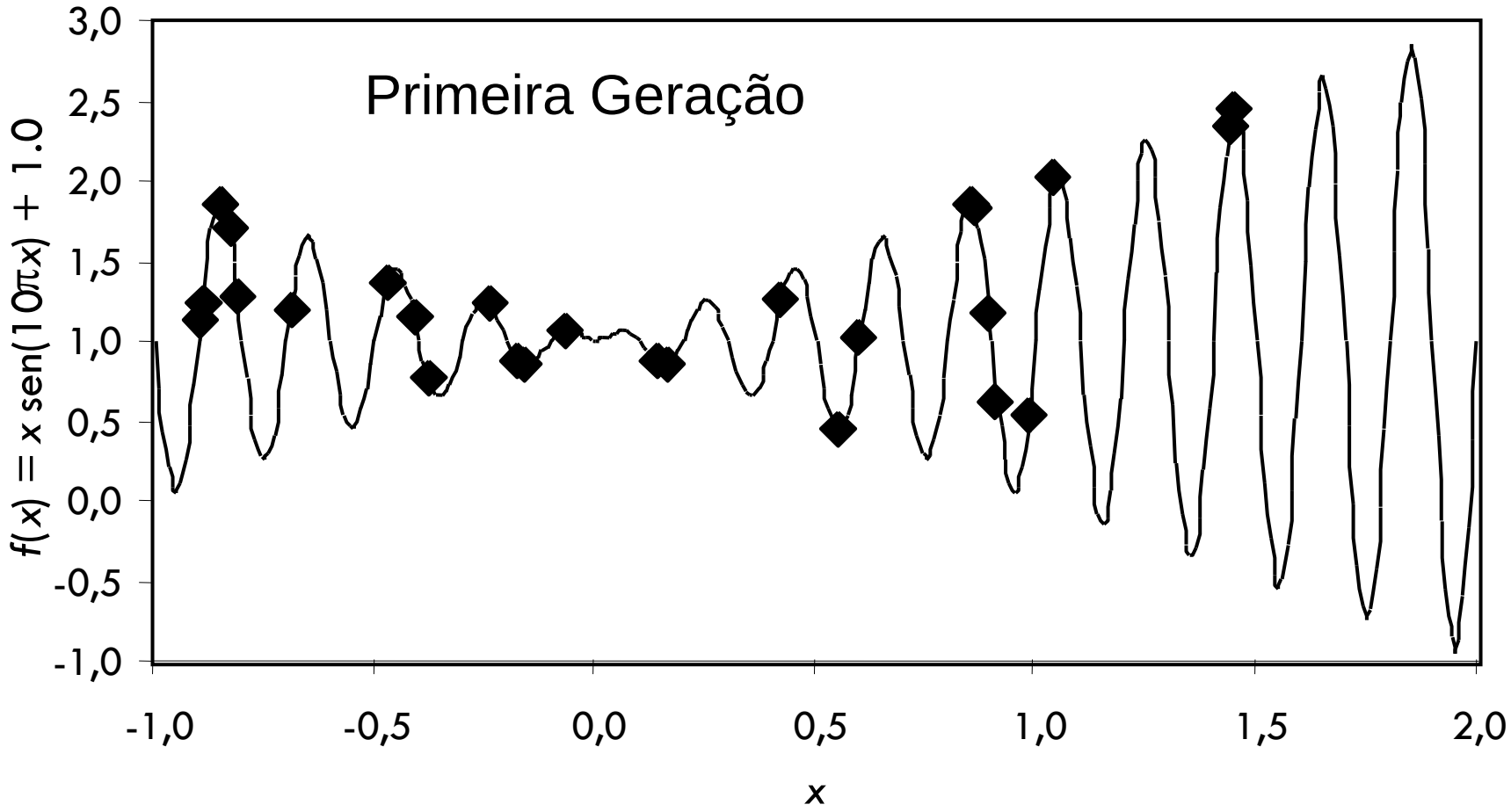
- Representar o parâmetro único deste problema (a variável *x*) na forma de um cromossomo:
  - Quantos bits deverá ter o cromossomo?
  - Quanto mais bits melhor precisão numérica.
  - Longos cromossomos são difíceis de manipular.
- Cromossomo com 22 bits: **1000101110110101000111**

As gerações



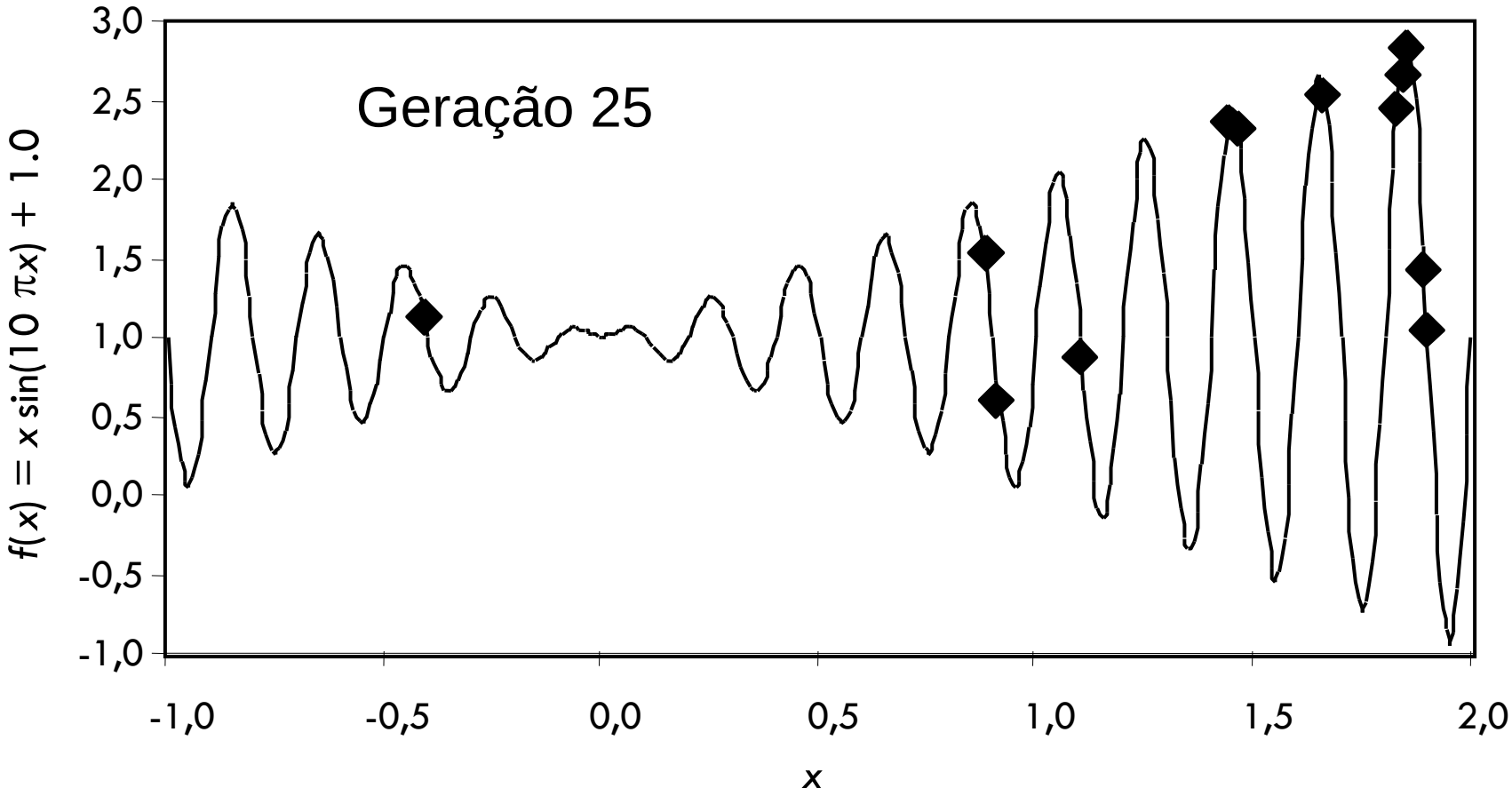
População gerada aleatoriamente

As gerações



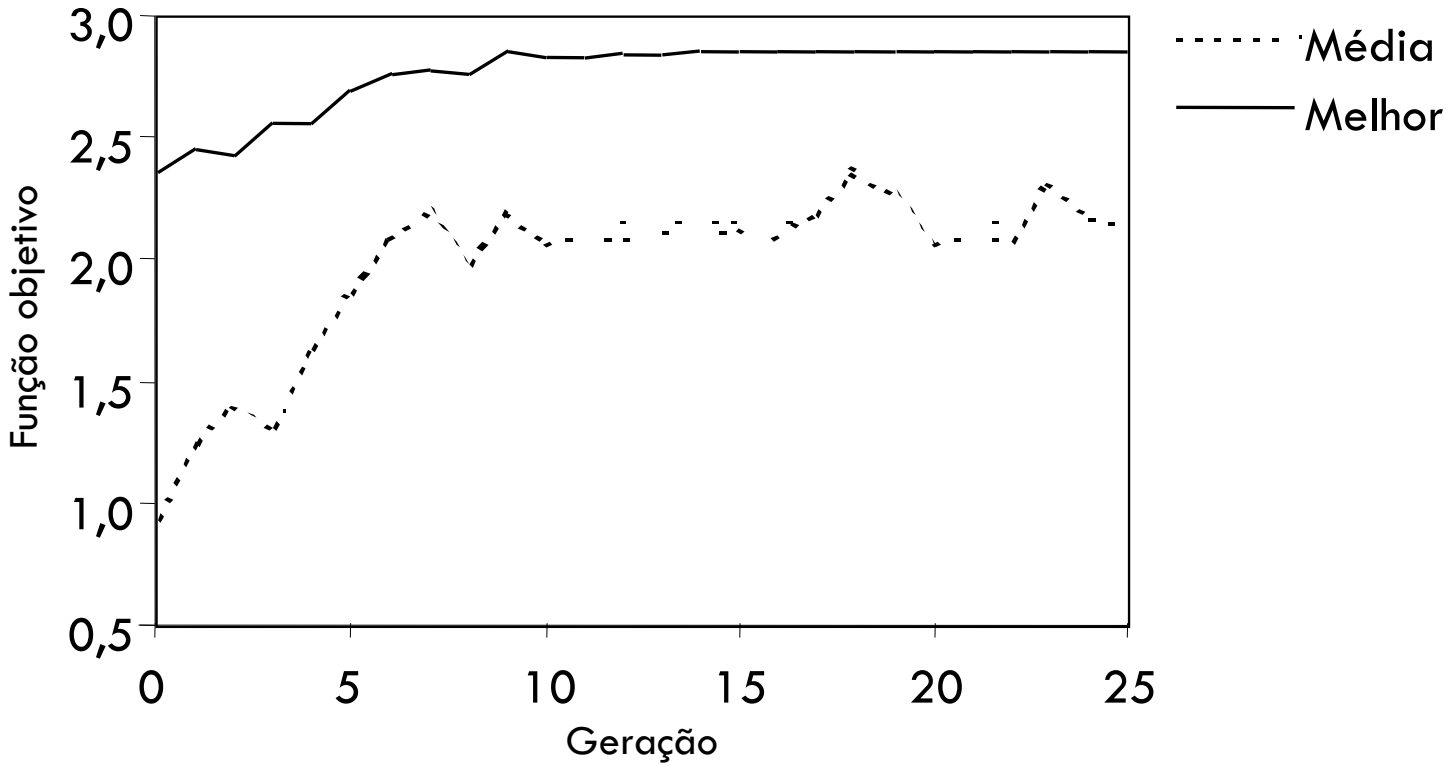
Pouca melhoria

As gerações



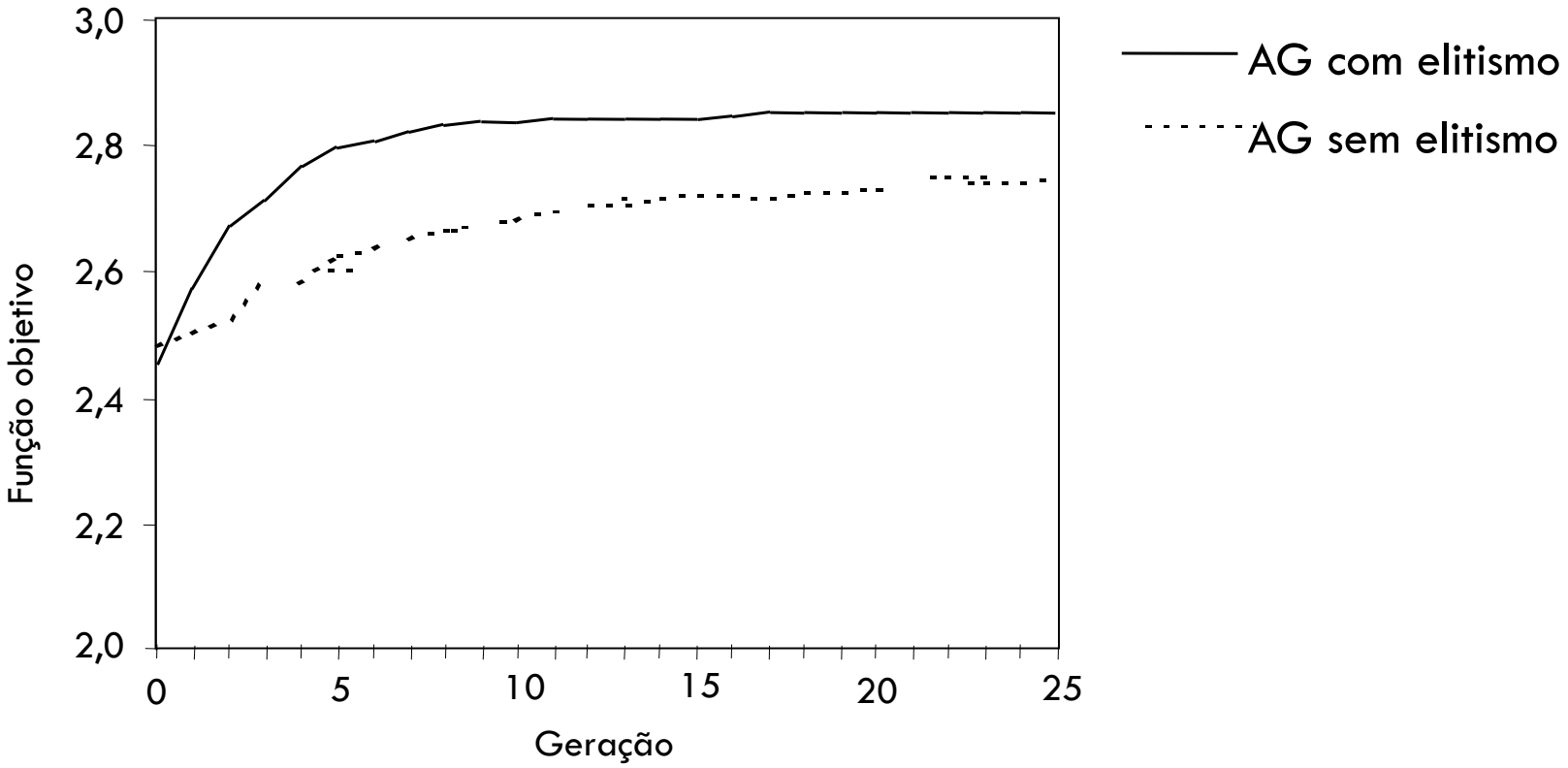
A maioria dos indivíduos encontraram o máximo global

As gerações



Na geração 15 o AG já encontrou o ponto máximo

Elitismo para o problema 2



Créditos

- Os dois exemplos utilizados no final desta aula, assim como algumas definições sobre os operadores estão baseados no material de aula da profa. Teresa Ludermir (UFPE) disponível em: *www.cin.ufpe.br/~if684/aulas/algoritmosgeneticostbl.ppt*