Introdução a Inteligência Artificial

Aula 6

Prof. Lucas Cambuim

Inventando Funções Heurísticas

Capítulo 3 – Russell & Norvig

Heurística e a sua Aplicação

A heurística é utilizada:

- Como técnica de busca para a obtenção de metas em problemas não algorítmicos que geram “explosões” combinatórias.
- As funções heurísticas são específicas para cada problema. Estas funções podem ser pouco certas ou podem não encontrar a melhor resposta, no entanto a sua utilização permite a libertação do uso das análises combinatórias.

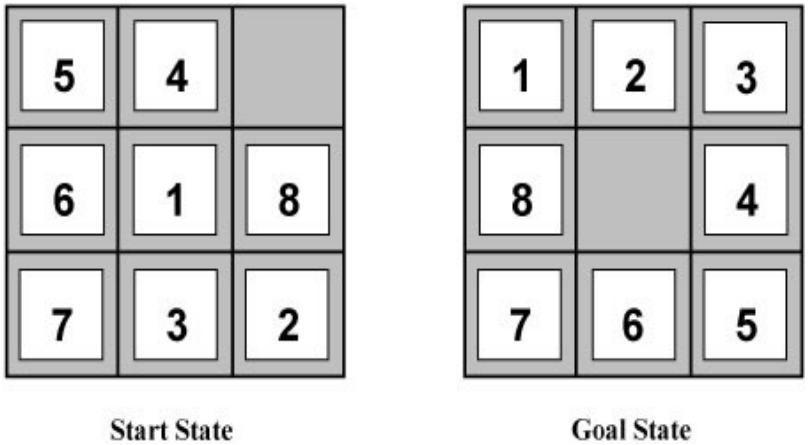
Inventando Funções Heurísticas

- Como escolher uma boa função heurística *h*?
- *h* depende de cada problema particular.
- *h* deve ser *admissível*
  - não superestima o custo real da solução
- Existem estratégias genéricas para definir *h*:
  1) Relaxar restrições do problema
  2) Usar informação estatística
  3) Identificar os atributos mais relevantes do problema (exige aprendizagem)

Relaxando o problema

- Problema Relaxado:
  - versão simplificada do problema original, onde os operadores são menos restritivos
- Exemplo: jogo dos 8 números
  - operador original
    - um numero pode mover-se de A para B se A e adjacente a B e B esta vazio
    - busca exaustiva $\approx$ 320 estados possíveis
  - Operadores relaxados:
    - um numero pode mover-se de A para B se A e adjacente a B (*h2*)
    - um numero pode mover-se de A para B se B esta vazio
    - um numero pode mover-se de A para B (*h1*)

Heurísticas para jogo 8 números



- **Heurísticas possíveis**
  - ***h1*** = no. de elementos fora do lugar (*h1*=7)
  - ***h2*** = soma das distancias de cada numero a posicão final (*h2*=2+3+3+2+4+2+0+2=18)

Usando informação estatística

- Funções heurísticas podem ser “melhoradas” com informação estatística:
  - executar a busca com um conjunto de treinamento (e.g., 100 configurações diferentes do jogo), e computar os resultados.
  - se, em 90% dos casos, quando h (n) = 14, a distancia real da solução e 18,
    - então, quando o algoritmo encontrar 14 para o resultado da função, vai substituir esse valor por 18.

Usando informação estatística

- Informação estatística expande menos nós, porem elimina *admissibilidade:*
- em 10% dos casos do problema acima, a função de avaliação poderá superestimar o custo da solução, não sendo de grande auxilio para o algoritmo encontrar a solução mais barata.

Escolhendo Funções Heurísticas

- E sempre melhor usar uma função heurística com valores mais altos, contanto que ela seja admissível.
  - no exemplo anterior, h2 é melhor que h1
  - hi domina hk $\rightarrow$ hi(n) $\geq$ hk(n) $\forall$n no espaço de estados
  - h2 domina h1 no exemplo anterior
    - Como ambas heurísticas são admissíveis, menos nos serão expandidos pela heurística dominante.

Escolhendo Funções Heurísticas

- Caso existam muitas funções heurísticas para o mesmo problema, e nenhuma delas domine as outras, usa-se uma heurística composta:
  - $h(n) = \max(h1(n), h2(n), \ldots, hm(n))$
  - Assim definida, h e admissível e domina cada função hi individualmente

Qualidade da função heurística

- Medida através do fator de expansão efetivo (b*)
  - b* e o fator de expansão de uma arvore uniforme com N nós e nível de profundidade d
  - N = 1 + b* + (b*)2 + ... + (b*)d , onde
    - N = total de nos expandidos para uma instancia de problema
    - d = profundidade da solução
- Mede-se empiricamente a qualidade de h a partir do conjunto de valores experimentais de N e d.
  - uma boa função heurística terá o b* muito próximo de 1

Experimento com 100 problemas

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

- Uma boa função heurística terá o *b** muito próximo de 1.

Busca Local

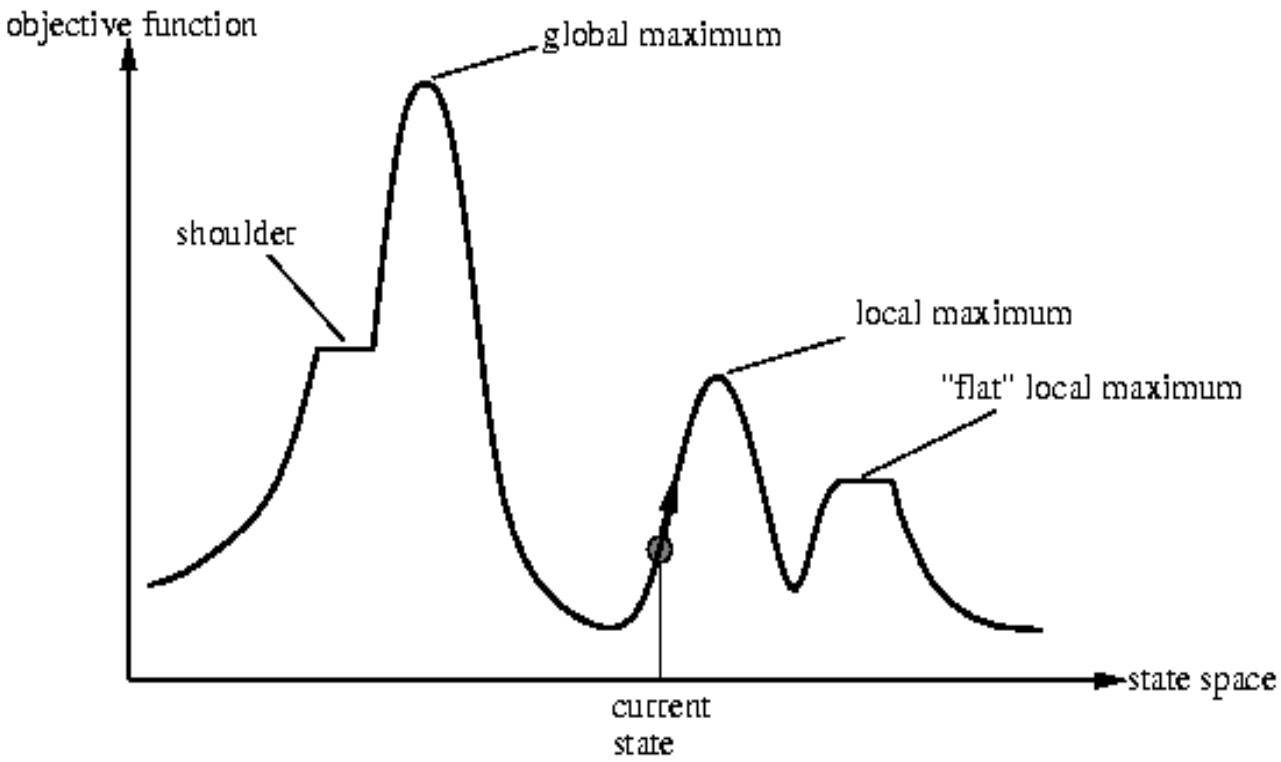
Capítulo 4 – Russell & Norvig

Algoritmos de busca local

- Não se interessam pelo caminho de solução, mas com a configuração final (ex. 8 queens);
- Operam com um único estado corrente
- Usam pouca memória
- Podem encontrar soluções **razoáveis** onde algoritmos sistemáticos se perdem

Topologia de espaço de estados

- elevação é o custo: encontrar o **mínimo global;**
- elevação é função objetivo: **máximo global.**



Busca de subida de encosta

- O algoritmo consiste em um laço repetitivo que percorre o espaço de estados no sentido do valor crescente (decrescente);
- Termina quando encontra um pico (vale) em que nenhum vizinho tem valor mais alto;

Busca de subida de encosta

- Não mantém uma árvore, o nó atual só registra o estado atual e o valor da função objetivo;
- Não examina antecipadamente valores de estados além dos vizinhos imediatos do estado corrente!
  - “ é como tentar alcançar o cume do Everest em meio a um nevoeiro durante uma crise de amnésia”
- Valor crescente ate um pico (nenhum vizinho mais alto)

```
função SUBIDA-DE-ENCOSTA(problema) retorna um estado que é um máximo local
  corrente ← CRIAR-NÓ(ESTADO-INICIAL[problema])
  repita
    vizinho ← um sucessor de corrente com valor mais alto
    se VALOR[vizinho] ≤ VALOR[corrente] então retornar ESTADO[corrente]
    corrente ← vizinho
```

Busca de Subida de Encosta

- Elevação é a função objetivo: queremos encontrar o máximo global.
- Elevação é o custo: queremos encontrar o mínimo global.
- O algoritmo consiste em uma repetição que percorre o espaço de estados no sentido do valor crescente (ou decrescente).
- Termina quando encontra um pico (ou vale) em que nenhum vizinho tem valor mais alto.

Busca de Subida de Encosta

- Não mantem uma arvore, o no atual só registra o estado atual e o valor da função objetivo.
- Não examina antecipadamente valores de estados além dos valores dos vizinhos imediatos do estado atual.

Busca de Subida de Encosta

- Problema: dependendo do estado inicial pode ficar presa em máximos (ou mínimos) locais.

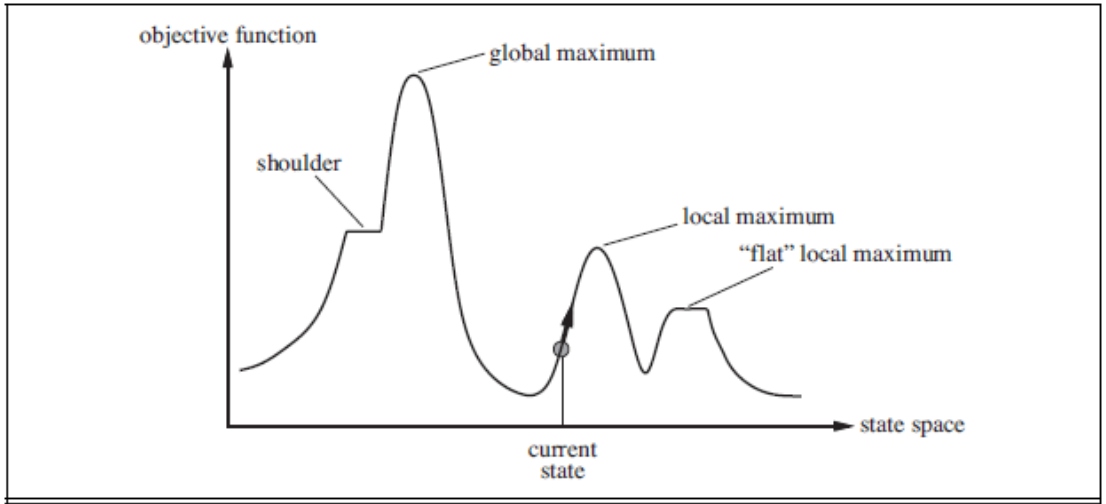
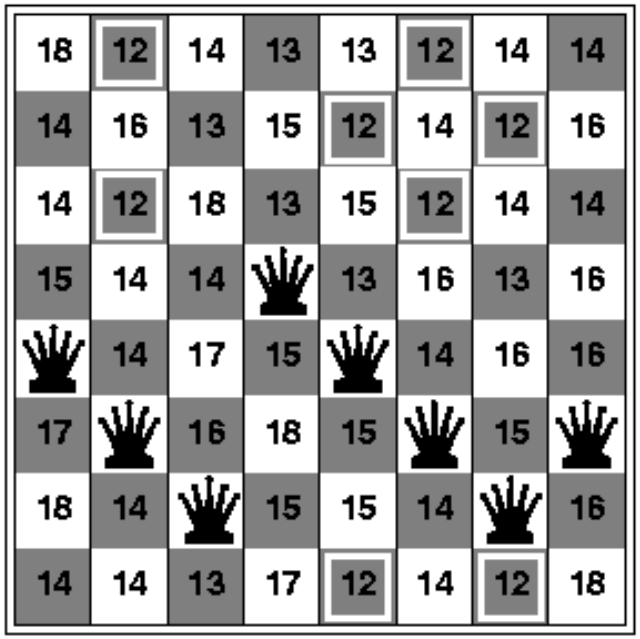


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Busca de subida de encosta: Problema das 8-rainhas

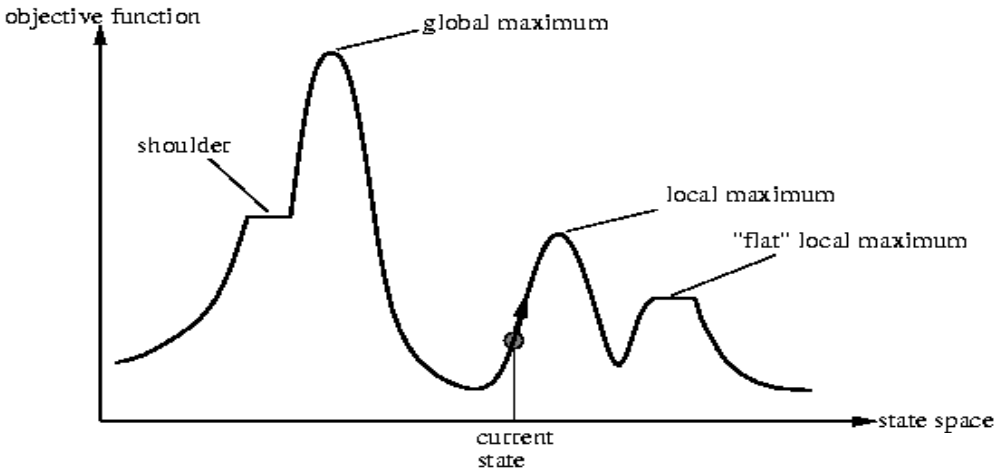
- h = número de pares de rainhas que estão “se atacando”, direta ou indiretamente
- h=17
- Em cada quadrado, valor de h para cada sucessor possível obtido pela **movimentação de uma** rainha dentro de sua coluna

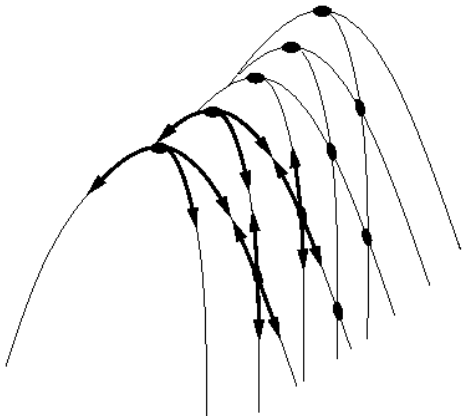


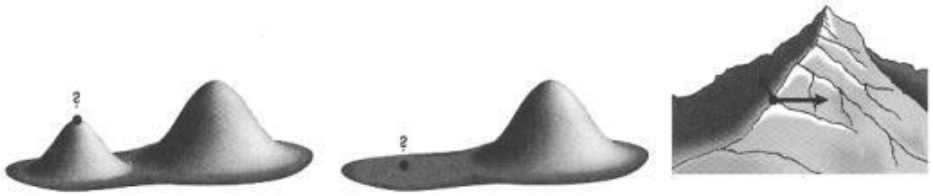
Subida de encosta: problemas

1. Máximos locais
2. Planíceis (platôs)
3. Encostas e picos: somente poucos vizinhos podem melhorar a solução

Nestes casos, o algoritmo chega a um ponto de onde não faz mais progresso



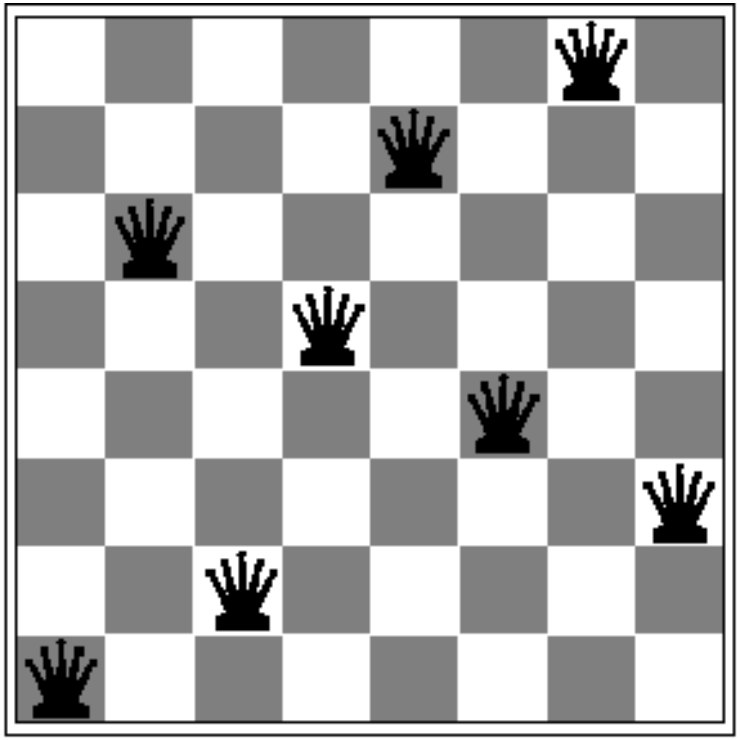




1 2 3

Subida de encosta: problemas

- Ex. máximo local em 8 queens



h=1, mas qualquer movimento de qualquer rainha só piora a situação...

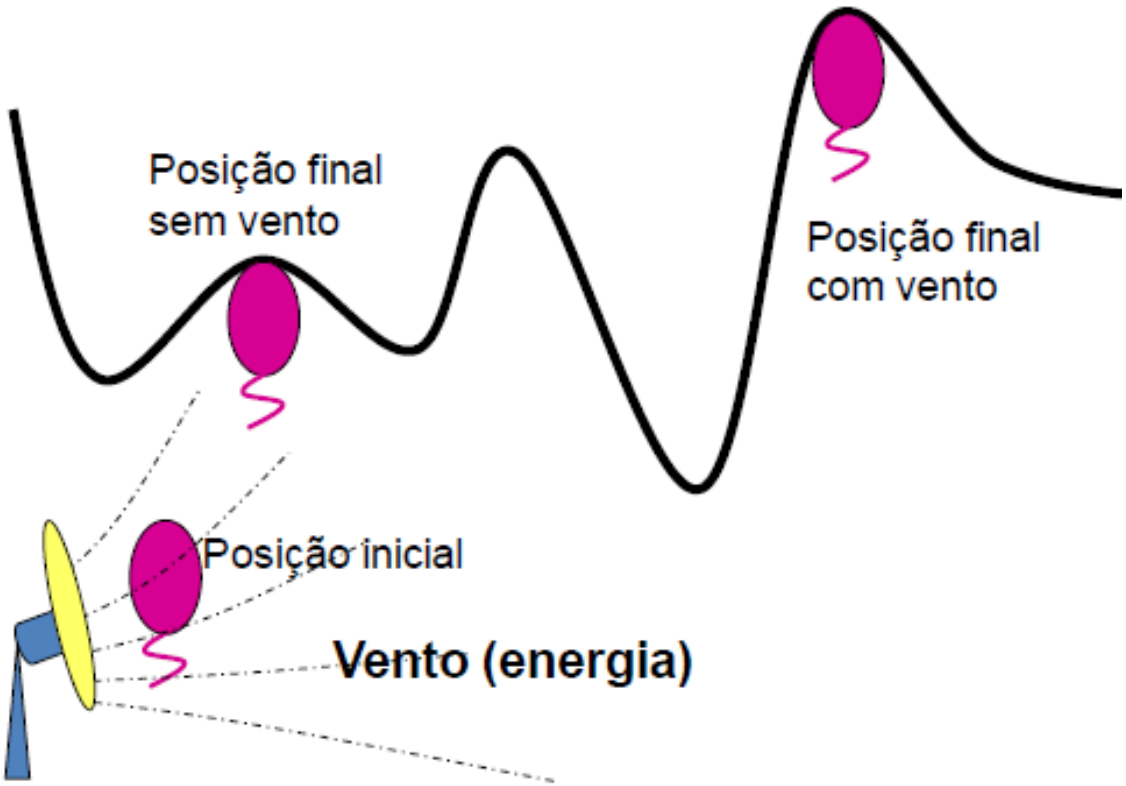
Subida de encosta: soluções

- Movimento lateral para evitar platôs
  - Porém pode ocorrer repetição infinita se for um máximo local plano que não seja planície;
  - Temos que impor um limite sobre o número de movimentos laterais;
- Subida de encosta com reinício aleatório
  - Conduz vária buscas subida de encosta a partir de diversos estados iniciais gerados ao acaso, para quando encontra um objetivo.
  - É completa, pois irá acabar gerando um estado objetivo como estado inicial. Porém ineficiente...

Têmpera simulada (simulated annealing)

- Combina a subida de encosta com um percurso aleatório resultando em eficiência e completeza.
- Subida de encosta dando uma “chacoalhada” nos estados sucessores;
  - i.e. estados sucessores com avaliação pior que o estado corrente podem ser escolhidos com uma certa probabilidade;
  - esta probabilidade diminui com o tempo

Têmpera simulada



Têmpera Simulada

- Têmpera simulada ou Arrefecimento simulado é uma *meta-heurística* para otimização que consiste numa técnica de busca local probabilística, e se fundamenta numa metáfora de um processo térmico de metalurgia, dito *recozimento:*

  –Passo 1: a temperatura do sólido é aumentada para um valor máximo no qual ele se funde;

  –Passo 2: o resfriamento deve ser realizado lentamente até que o material se solidifique.

- Isto provoca que os átomos desse material ganhem energia para se movimentarem livremente e, ao arrefecer de forma controlada, dar-lhes uma melhor hipótese de se organizarem numa configuração com menor energia interna, para ter, como resultado prático, uma redução dos defeitos do material.

Têmpera simulada

- O algoritmo utiliza um *mapeamento de resfriamento* de instantes de tempo (*t*) em temperaturas (*T*).
- Nas iterações iniciais, não escolhe necessariamente o "melhor" passo, e sim um movimento aleatório:
  - Se a situação melhorar, esse movimento será sempre escolhido posteriormente;
  - Caso contrário, associa a esse movimento uma probabilidade de escolha menor do que 1.
- Essa probabilidade depende de dois parâmetros, e decresce exponencialmente com o agravamento causado pelo movimento, $e^{\left\{-\frac{\Delta E}{T}\right\}}$, onde:
  - *ΔE = Valor[próximo-nó] - Valor[nó-atual]*
  - *T = Temperatura*

Têmpera simulada

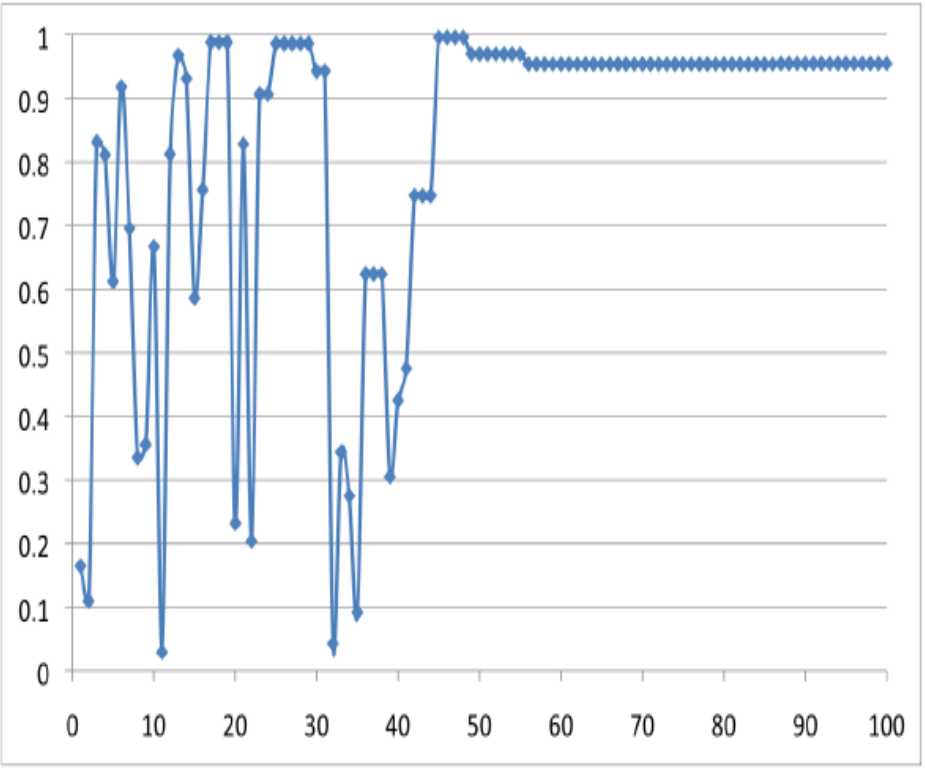
- Escapa de máximos locais permitindo alguns passos "**ruins**" mas gradualmente decresce a sua frequência.

```
função TÊMPERA-SIMULADA(problema, escalonamento) retorna um estado solução
   entradas: problema, um problema
escalonamento, um mapeamento de tempo para "temperatura"
   atual ← CRIAR-NÓ(problema.ESTADO-INICIAL)
   para t = 1 até ∞ faça
      T ← escalonamento[t]
      se T = 0 então retornar corrente
      próximo ← um sucessor de atual selecionado aleatoriamente
      ΔE ← próximo.VALOR – atual.VALOR
      se ΔE > 0 então atual ← próximo
      senão atual ← próximo somente com probabilidade e^(ΔE/T)
```

- Com o tempo (diminuição da temperatura), este algoritmo passa a
- funcionar como Trepa Colinas .
- O algoritmo é óptimo e completo se o mapeamento de resfriamento
- tiver muitas entradas com variações suaves.
- Isto é, se o mapeamento diminui *T* suficientemente devagar no tempo,
- o algoritmo irá encontrar um máximo global óptimo.

Têmpera simulada

- Encontrar o ponto máximo da função: **f(*x*)** = ***x*²**
- 100 iterações, cada estado é um número x ∈ [0,1], estado inicial é x = 0, f(x) = x^2



Propriedades da busca de tempera simulada

- Com o tempo (diminuição da temperatura), este algoritmo passa a funcionar como o hill climbing
- Pode-se provar que se T decresce devagar o suficiente, a busca pode achar uma solução ótima global com probabilidade tendendo a 1.
- Muito usada em projetos de circuitos integrados, layout de instalações industriais, otimização de redes de telecomunicações, etc.

Busca em Feixe Local

- Manter *k* estados em vez de um.
- Começa com **k** estados gerados aleatoriamente.
- Em cada passo, são gerados todos os sucessores de todos os **k** estados.
- Se um dos sucessores for o objetivo, o algoritmo para; caso contrário, escolhe os **k** melhores sucessores a partir da lista completa.

```
function BEAM-SEARCH(problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

Busca em Feixe Local

- Note que isso NÃO corresponde à execução de k reinícios aleatórios em paralelo da busca local subida da encosta (*random start*)!
  - ➢ Na busca por reinício aleatório, cada processo de busca roda independentemente uns dos outros.
  - ➢ Na busca em feixe local, informação útil é passado entre todas as busca em paralelo.
- Note que sempre somente **k** estados são considerados como estados atuais na busca.
- Em sua versão mais simples a busca em feixe local pode sofrer da falta de diversidade entre os k estados.
  - Eles podem rapidamente tornar-se concentrado em uma região pequena do espaço de estado.
  - Como se fosse um hill climbing mais custoso.
- Uma variante alivia este problema: Feixe local estocástico
  - Escolhe os k sucessores randomicamente, com a probabilidade de escolher um dado sucessor sendo uma função crescente de seu valor.
  - Assemelha-se ao processo de seleção natural
    - Sucessores (descendências) de um estado gera a próxima geração de acordo com o valor de qualidade da sua solução (fitness)

Algoritmos genéticos

- Um estado sucessor é gerado por meio da combinação de dois estados pais.
- Começa com *k* estados gerados aleatoriamente (população).
- Um estado (possível solução) é representado por uma string de um alfabeto finito (normalmente strings de 0s e 1s).
- Função de avaliação (função de fitness). Valores mais altos pra estados melhores.
- Produz a próxima geração de estados por **seleção**, **cruzamento** (crossover) e **mutação**.

Algoritmos genéticos

Inspiração para a classe de algoritmos Genéticos:

Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes

Algoritmos genéticos

A modelagem de um problema AG envolve primeiramente:

- Codificação da solução
- Função de avaliação
- Função de aptidão (*fitness*)

Função de avaliação

- Provém uma medida de desempenho com respeito a um conjunto particular de parâmetros.
- Deve ser relativamente rápida, uma vez que, em cada iteração, cada membro da população é avaliado e recebe um valor de aptidão.
- A avaliação de um membro (cromossomo) representando um conjunto particular de parâmetros é independente da avaliação de qualquer outro membro

Função de aptidão

–Transforma a medida da função de avaliação em alocação de oportunidades **reprodutivas**.

–É sempre definida de acordo com outros membros da atual população.

–No algoritmo genético canônico, aptidão é definida como $fi_x = f(x)/f'$ onde $f(x)$ é a avaliação associada ao cromossomo **x** e f' é a soma da avaliação (ou média) de todos os membros da população.

–A aptidão pode também ser associada à classificação de um cromossomo na população ou outras medidas.

Codificação

- O cromossomo deve, de algum modo, conter informação sobre a solução a qual representa.
- O modo mais usual de codificação é a sequência binária. O cromossomo, então deve ser algo como:

Cromossomo 1	1101100100110110
Cromossomo 2	1101111000011110

- Cada bit na sequência (gene) pode representar uma característica da solução ou a série como um todo pode representar um número

Pseudo-algoritmo de um AG (canônico)

- **[Início]** Gerar uma população aleatória com *n* cromossomos

  Repetir até obter a solução (ou terminar)

  Geração

  1. **[Avaliação]** Determinar ***f(x)*** *e* fi_x de cada cromossomo *x* na população.
  2. **[Seleção]** Selecionar elementos para criar uma população intermediária

     //Quanto melhor o fi_x, maior chance de ser selecionado); uma porcentagem dos mais adaptados é mantida, enquanto os outros são descartados (mortos).
  3. **[Recombinação (Crossover)]** Com uma probabilidade de recombinação, realizar uma recombinação sobre os pais para formar uma nova prole

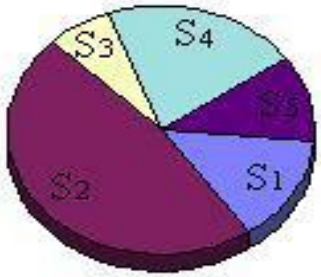
     //Se não ocorrer recombinação, a prole é uma cópia exata dos pais.
  4. **[Mutação]** Com uma probabilidade de mutação, realizar mutação sobre a nova prole em cada *gene* (posição no cromossomo).
  5. **[Atualização da população]** Usar a nova população gerada para repetir os passos 1-5 do algoritmo

  Retornar a melhor solução na população atual.

Seleção através da Roleta

- Os pais são selecionados de acordo com sua função de aptidão. Quanto melhor os cromossomos são, maior a chance de serem selecionados. Imagine uma **roleta** onde são colocados todos os cromossomos da população, o espaço ocupado depende do valor da sua função de aptidão, como mostra a figura:

Indivíduo S_i		Aptidão $f(S_i)$	Aptidão Relativa
S_1	10110	2.23	0.14
S_2	11000	7.27	0.47
S_3	11110	1.05	0.07
S_4	01001	3.35	0.21
S_5	00110	1.69	0.11



Elitismo

- Elitismo é o nome do método, aliado à seleção, que primeiro copia o melhor cromossomo (ou alguns melhores cromossomos) para a nova população. O processo restante é feito do modo clássico.
- O elitismo pode aumentar rapidamente o desempenho do AG porque evita que se perca a melhor solução encontrada até o momento.

Recombinação (Crossover)

- Seleciona genes a partir dos cromossomos pais e cria uma nova prole.
- O modo mais simples: 1. escolher aleatoriamente algum ponto (*locus* entre genes) no cromossomo, 2. tudo que estiver antes desse ponto será copiado do primeiro pai, 3. tudo que estiver depois será copiado do segundo pai.

Cromossomo 1	11011 | 00100110110
Cromossomo 2	11011 | 11000011110
Prole 1	11011 | 11000011110
Prole 2	11011 | 00100110110

- Existem outros meios de realizar recombinação, por exemplo, podemos escolher mais pontos de recombinação.
- Recombinações desenvolvidas para problemas específicos podem melhorar o desempenho do AG.

Mutação

- Mutação permite evitar que a população fique presa em um mínimo (máximo) local.
- A mutação altera aleatoriamente a nova prole. Na codificação binária, podemos mudar alguns bits de 1 para 0 e de 0 para 1:

Prole Original 1	**11101111000011110**
Prole Original 2	**11101100100110110**
Prole com Mutação 1	**11100111000011110**
Prole com Mutação 2	**11101101100110110**

- A mutação depende tanto da codificação como da recombinação

Parâmetros - I

- **Probabilidade de recombinação:** indica o quão frequente a recombinação é executada.
  - –Se **não** ocorre recombinação, a prole é a cópia dos pais.
  - –Se ocorre recombinação, a prole é formada por partes dos pais.
  - –Se a probabilidade de recombinação é **100%**, então toda a prole será formada com recombinação.
  - –Se for **0%**, a nova geração será formada por cópias exatas da população anterior
- A recombinação é realizada com a esperança de que os novos cromossomos tenham partes boas dos cromossomos anteriores e talvez sejam melhores que seus pais. Entretanto é aconselhável deixar parte da população sobreviver na geração seguinte.

Parâmetros - II

- **Probabilidade de mutação:** indica o quão freqüente partes dos cromossomos sofrerão mutações.

  –Se não ocorrer mutação (probabilidade de mutação = 0%), a prole não sofrerá mudanças após a recombinação.

  –Se ocorrer mutação, parte dos cromossomos será alterada.

  –Se a probabilidade de mutação for **100%,** todos os cromossomos serão alterados.

- A mutação é realizada para prevenir que o AG caia em um extremo local (mínimo ou máximo, depende das funções usadas), porém ela não deve ocorrer com muita frequência, pois acarretaria em uma **busca aleatória** no espaço de soluções.

Parâmetros - III

- **Tamanho da população:** indica quantos cromossomos existem em uma população (em uma geração).

  –Se existirem muito poucos cromossomos, o AG terá poucas possibilidades de realizar a recombinação e apenas uma pequena porção do espaço de estados será explorada.

  –Se existirem muitos cromossomos, o AG ficará lento.

  –Pesquisas mostram que após algum limite (que depende principalmente da codificação e do problema) não é vantajoso aumentar o tamanho da população pois não tornará o algoritmo mais rápido.

Parâmetros - IV

- **Intervalo de Geração**. Controla a porcentagem da população que será substituída durante a próxima geração.

  –Com um valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão.

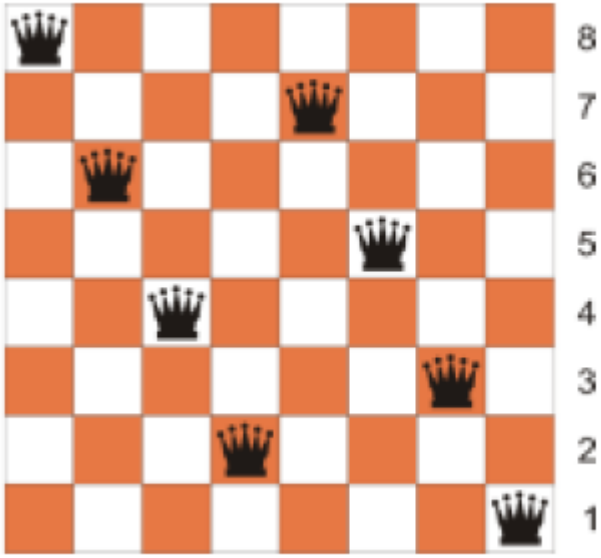
  –Com um valor baixo, o algoritmo pode tornar-se muito lento.

Exemplo: 8 Rainhas

Codificação

–Cromossomos compostos por 8 números (genes), a posição do gene indica a coluna, o valor do número indica a linha

86427531 ➔

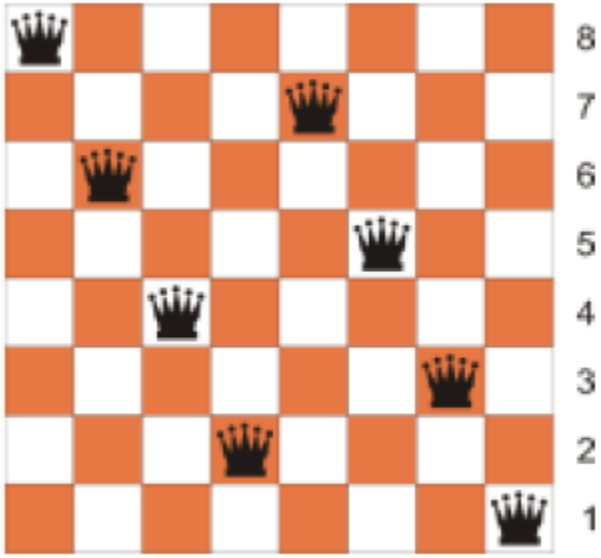


Exemplo: 8 Rainhas

- Função de avaliação:
  - f(x) =número de pares que **não** se atacam
  - Na solução: 1-2,1-3, ...,1-8, 2-3,...,2-8, 3-4,..., 3-8,..., 6-7, 6-8, 7-8 = 28 pares
- Função de aptidão:
  - $fi_x = f(x)/\sum f(x)$ [%]

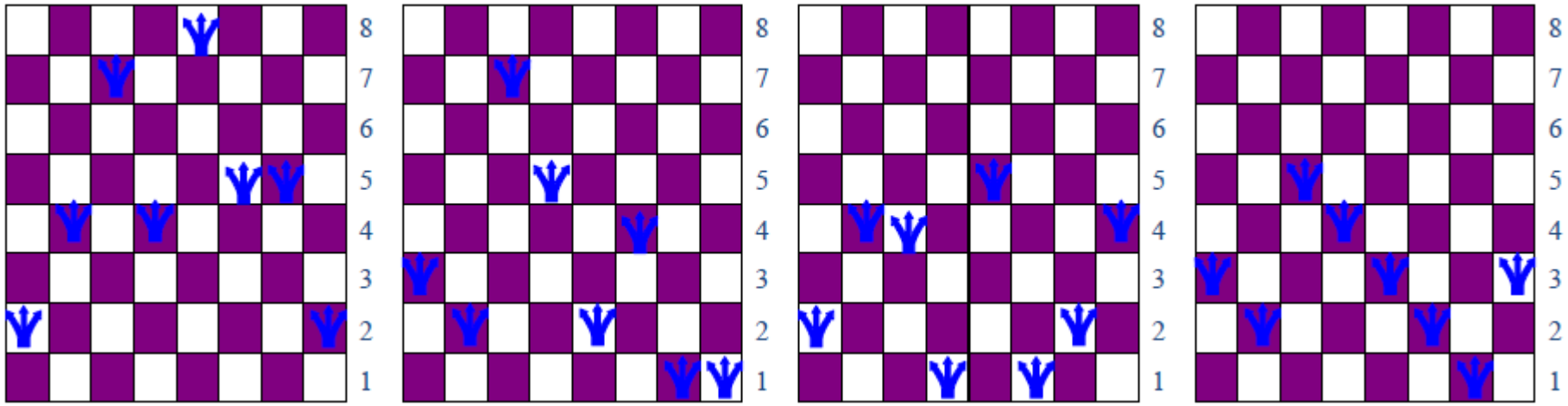
x: 1-8 em ataque

✓ f(x) = 27



Exemplo: 8 Rainhas

- Considere a seguinte população inicial:



1. Determine a codificação de cada cromossomo.

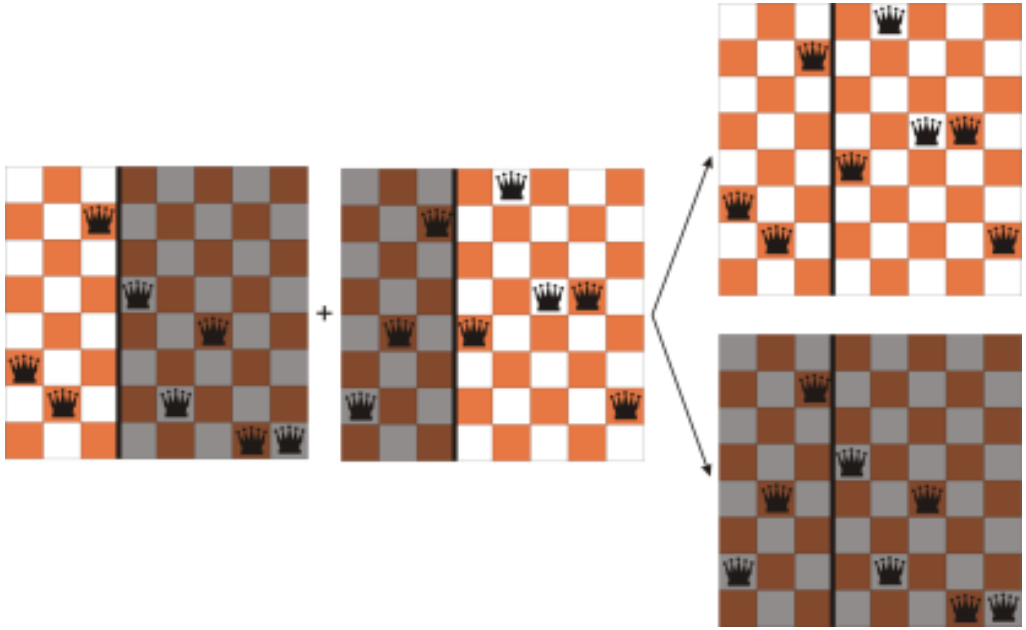
2. Determine a população intermediária considerando que a seleção (roleta) tenha sorteado, nesta ordem: o 2º mais apto, o 1º mais apto, novamente o 2º mais apto e o 3º mais apto (matou o menos apto).

3. Recombine os 2 primeiros após o 3º gene (gerando 2 filhos) e os 2 últimos após o 5º gene (mais 2 filhos). Mostrar a população atual.

4. Faça a mutação (cromossomo, novo valor, gene): (1º, 1, 6º), (3º, 2, 3º), (4º, 7, 8º). Mostre a nova população.

Exemplo: 8 Rainhas

- Recombinação:



Exemplo: 8 Rainhas

- Indivíduos com maiores aptidões possuem mais chances de ser selecionados

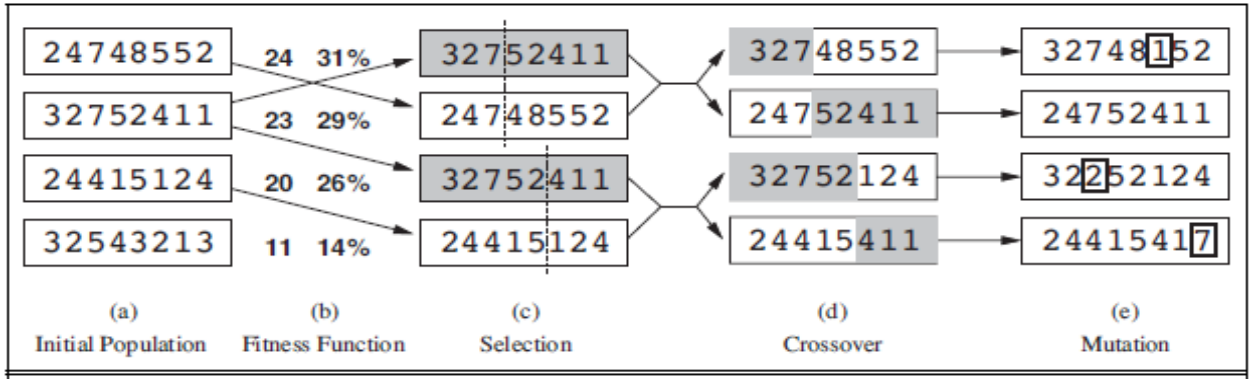


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

- Função de fitness: número de pares de rainhas que não estão se atacando (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc