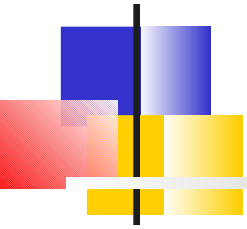
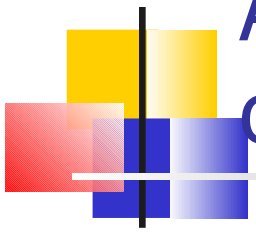


Busca Local

Texto base:

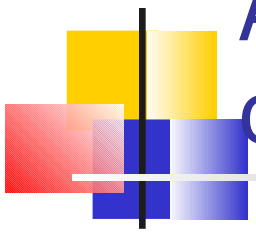
Stuart Russel e Peter Norving - “Inteligência Artificial”

junho/2007



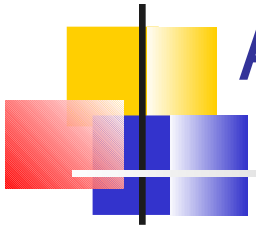
Algoritmos de busca local e problemas de otimização

- 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
  - Outros exemplos:
    - Projeto de CIs
    - Layout de instalações industriais
    - Escalonamento de jornadas de trabalho
    - Otimização de redes de telecomunicações
    - Roteamento de veículos
    - Outras aplicações



Algoritmos de busca local e problemas de otimização

- Alguns espaços de busca são muito grandes para uma busca sistemática
- Operam sobre um único estado corrente, ao invés de vários caminhos
- Em geral se movem apenas para os vizinhos desse estado
- Vantagens:
  - Ocupam pouquíssima memória (normalmente constante)
  - Podem encontrar soluções razoáveis em grandes ou infinitos espaços de estados, para os quais os algoritmos sistemáticos são inadequados

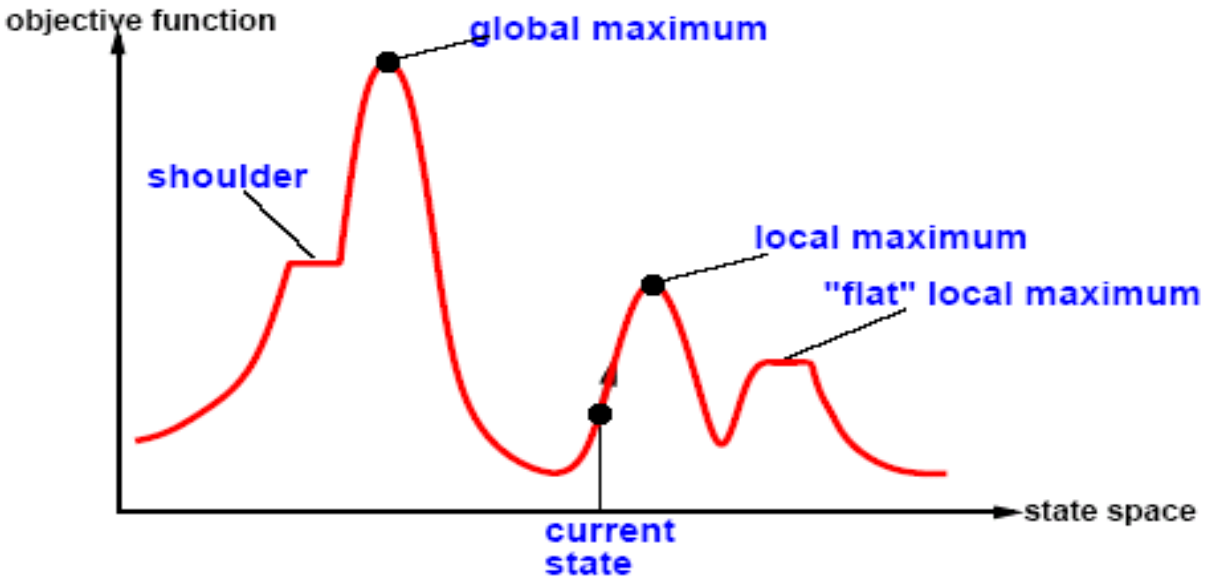


Algoritmos de busca local

- Algoritmos de busca local são úteis para resolver problemas de otimização puros
- Onde o objetivo é encontrar o melhor estado de acordo com uma função objetivo
- Normalmente o espaço de estados é considerado como tendo uma topologia onde existe:
  - Uma posição – definida pelo estado
  - Uma elevação – definida pelo valor da função de custo da heurística ou da função objetivo

Algoritmos de busca local

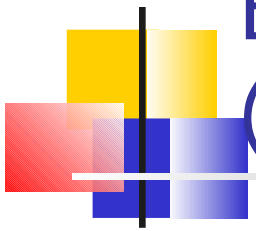
- Elevação = custo -> objetivo = mínimo global
- Elevação = função objetivo -> objetivo = máximo global
- É **completo** se sempre encontra um objetivo
- É **ótimo** se sempre encontra um mínimo/máximo **global**



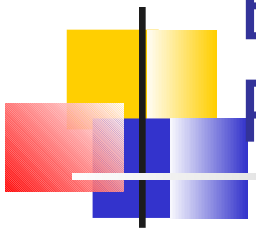
Busca de subida de encosta (*Hill-Climbing*)

```
function HILL-CLIMBING(problema) returns um estado que
   é o máximo local
inputs:problema, um problema
local:atual, um nodo
    vizinho, um nodo

atual <- CRIAR-NÓ(ESTADO_INICIAL[problema])

loop
   vizinho <- um sucessor atual com valor mais alto
   If VALOR[vizinho] <= VALOR[atual] then
    return ESTADO[atual]
   atual <- vizinho
end
```

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



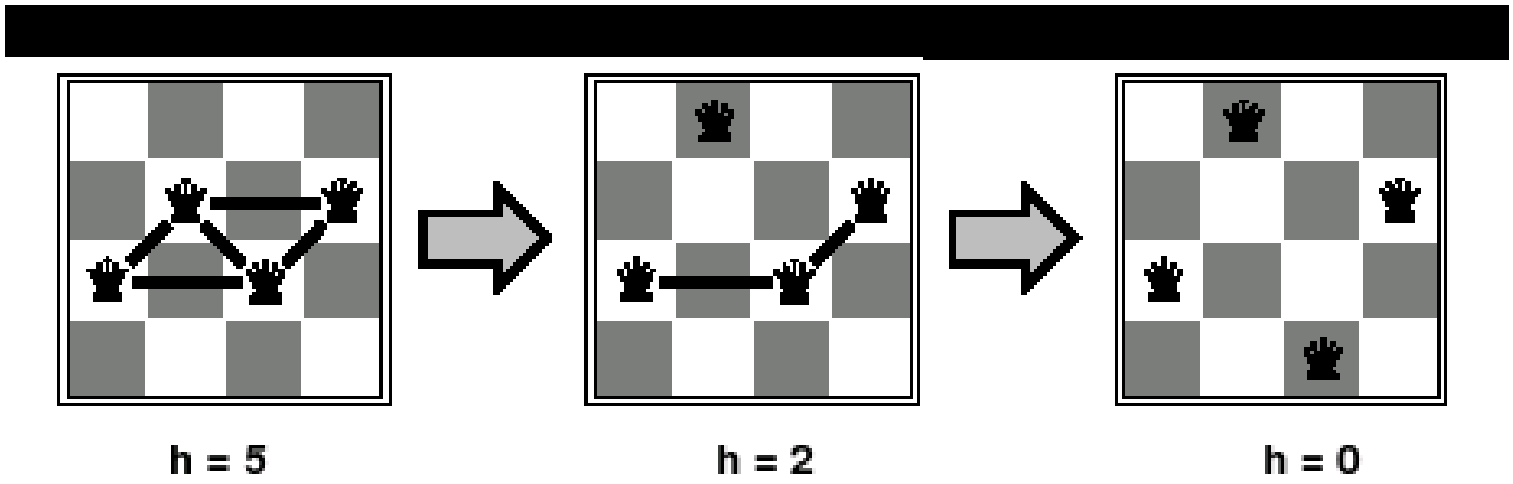
- Se move de forma contínua no sentido do valor crescente
- Termina quando alcança um pico, em que nenhum vizinho tem valor mais alto
- Não mantém árvore de busca, somente o estado e o valor da função objetivo
- Não examina antecipadamente valores de estados além de seus vizinhos imediatos (busca gulosa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



Busca de subida de encosta Para o problema da n-rainhas

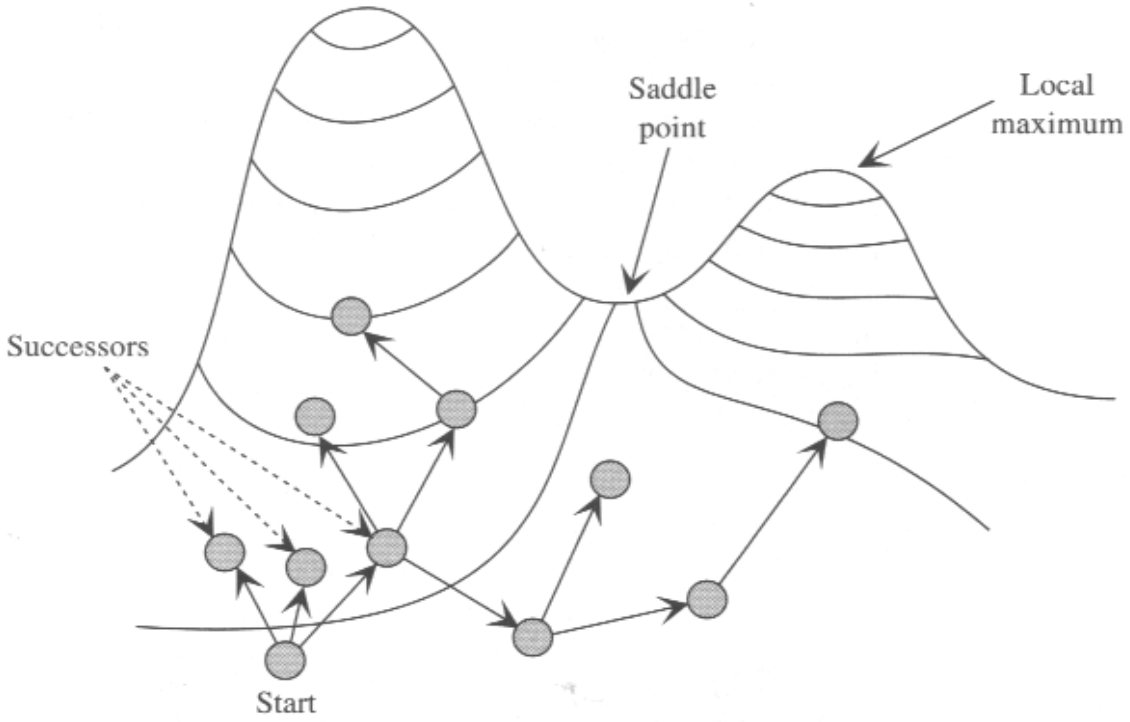
- Algoritmos de busca local utilizam uma formulação de estados completos
  - Cada estado tem n rainhas, 1 por coluna
- Função sucessora gera todos os estados possíveis
  - Gerados pela movimentação de uma única rainha para outro lugar na mesma coluna
- A função heurística é o números de pares de rainhas que estão se atacando umas às outras
  - O mínimo global dessa função é zero, que só ocorre em soluções perfeitas

Busca de subida de encosta
Para o problema da n-rainhas



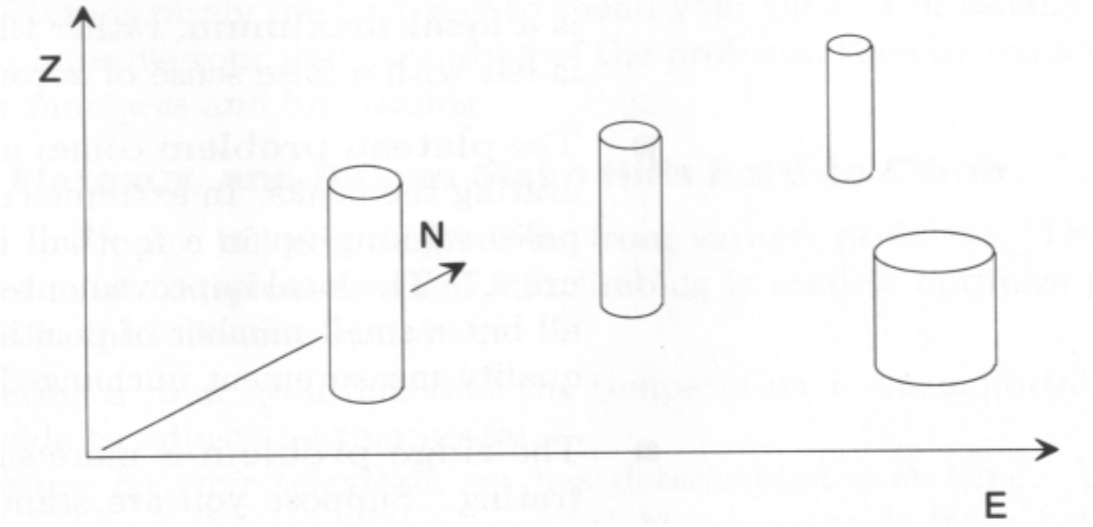
Hill-climbing - problema

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



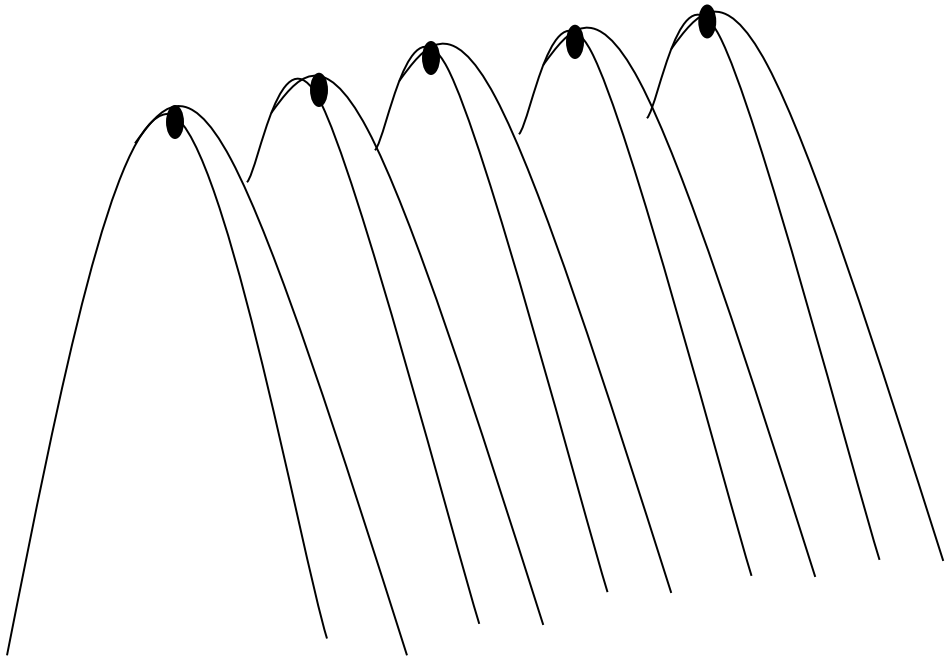
Hill-climbing - problema

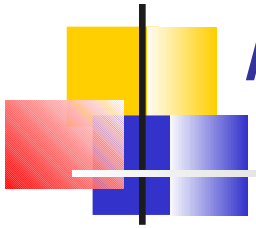
- Podem existir **platôs** fazendo com que em certas áreas a função tenha valores muito próximos.



Hill-climbing - problema

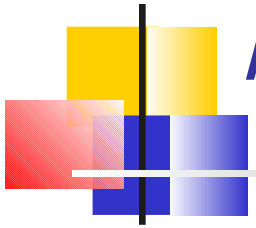
- Podem existir **picos** que fazem com que a função de qualidade oscile entre vários máximos locais.





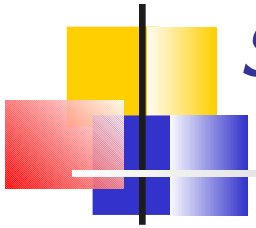
Hill-climbing

- Formas de resolver o problema de máximos locais
  - Ao atingir o máximo fazer **alterações aleatórias** para ver se não há estados melhores (*random-restart-hill-climbing*)
- Pode-se usar também uma função que testa se o estado é solução em vez de procurar somente pelo máximo
- Pode usar *backtracking* para procurar estados melhores
- Término do algoritmo
  - Número fixo de interações
  - Porcentagem de melhoramento
  - Tempo fixo



Hill-climbing

- O sucesso deste tipo de busca depende muito da topologia do espaço de estados
- Muitos problemas reais tem uma topologia mais parecia com uma família de ouriços em um piso plano
- Com ouriços em miniatura vivendo na ponto de cada espinho de um ouriço, *ad infinitum*
- Problemas NP-difíceis têm um número exponencial de máximos locais em que ficam paralisados



Simulated Annealing (Têmpera simulada)

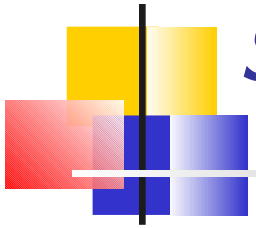
- **Têmpera**: processo usado para temperar ou endurecer metais e vidro aquecendo-os a alta temperatura e depois resfriando gradualmente
- Idéia:
  - Fugir do máximo local permitindo alguns **movimentos "ruins"** para fora do máximo, mas gradualmente decrescendo seu tamanho e freqüência
- A temperatura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

Propriedades do *Simulated Annealing*

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

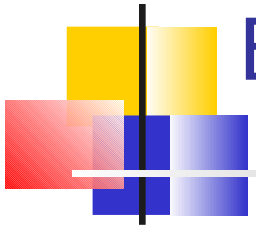
$$e^{\frac{\Delta E}{T}}$$

- T decrescendo suficientemente lento $\Rightarrow$ sempre alcança o melhor estado
- Para valores maiores de T, soluções ruins são permitidas
- T próximo de zero, a probabilidade de se escolher soluções ruins diminui
- ΔE determina qual é a variação entre a solução corrente e a próxima solução



Simulated Annealing

```
function SIMULATED_ANNEALING(problema, escala) returns um estado
   solução
inputs: problema, um problema
        escala, um mapeamento do tempo pela temperatura
local: atual, um nodo
       próximo, um nodo
       T, uma temperatura controlando a probabilidade de dar
   passos pra baixo

atual ← CRIAR-NÓ(ESTADO_INICIAL[problema])

for tempo ←1 to ∞
   T ← escala[tempo]
   if T = 0 then return atual
   próximo ← um sucessor de atual aleatoriamente selecionado
   ΔE ← VALOR[próximo] – VALOR[atual]
   if ΔE > 0 then atual ← próximo
   else atual ← próximo somente com uma probabilidade e^(ΔE/T)
end
```



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
- Problema: os k estados podem rapidamente ficar concentrados em uma pequena região do espaço de estados
- Solução: escolher k sucessores melhores que seus pais ao acaso