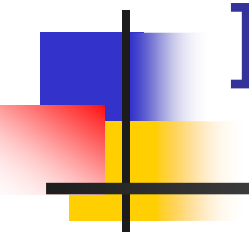


Inteligência Artificial

Prof. Tiago A. E. Ferreira

Aula 10 – Busca com Heurísticas

(Parte II)

Inventando Funções Heurísticas

- Como escolher uma boa função heurística *h*?
- *h* depende de cada problema particular.
- *h* deve ser *admissível*
  - não superestimar 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).

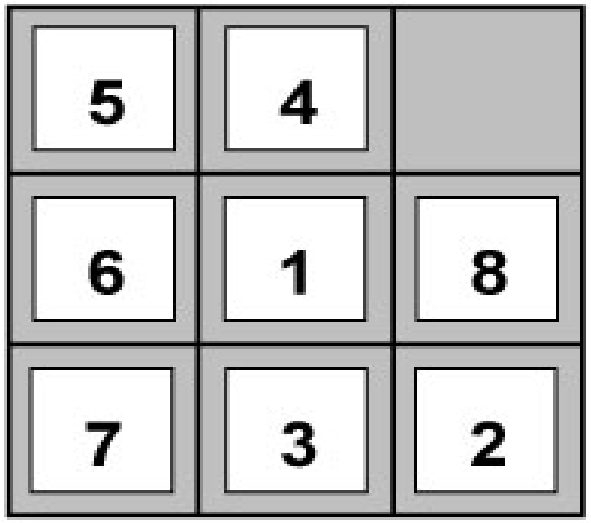
Relaxando Restrições do Problema

- Problema Relaxado:
  - versão simplificada do problema original, onde os operadores são menos restritivos
- Exemplo: jogo dos 8 números - operador original
  - um número pode mover-se de A para B se A é adjacente a B e B está vazio
  - busca exaustiva $\approx 3^{20}$ estados possíveis

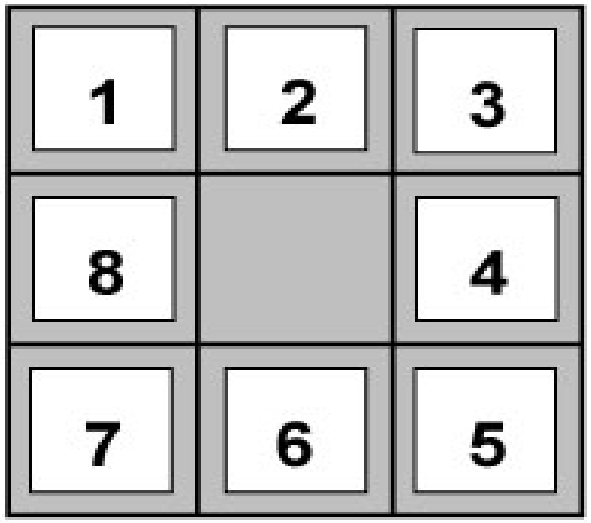
4	5	8
	1	6
7	2	3

- Operadores relaxados:
  1. um número pode mover-se de A para B se A é adjacente a B (*h2*)
  2. um número pode mover-se de A para B se B está vazio
  3. um número pode mover-se de A para B (*h1*)

Heurísticas para jogo 8 números



Start State



Goal State

Heurísticas possíveis

h1 = no. de elementos fora do lugar (*h1*=7)

h2 = soma das distâncias de cada número à posição 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 distância real da solução é 18,
  - então, quando o algoritmo encontrar 14 para o resultado da função, vai substituir esse valor por 18.

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

Escolhendo Funções Heurísticas

- É sempre melhor usar uma função heurística com valores mais altos, contanto que ela seja *admissível.*
  - ex. *h2* melhor que *h1*
- h_i domina $h_k \Rightarrow h_i(n) \geq h_k(n) \; \forall n$ no espaço de estados
  - h_2 domina h_1 no exemplo anterior
- Caso existam muitas funções heurísticas para o mesmo problema, e nenhuma delas domine as outras, usa-se uma *heurística composta*:
  - $h(n) = max\,(h_1(n), h_2(n), \ldots, h_m(n))$
  - Assim definida, h é *admissível* e *domina* cada função h_i individualmente

Qualidade da função heurística

- Qualidade da função heurística: **medida através do fator de expansão efetivo (*b**).**
  - *b** é o fator de expansão de uma árvore uniforme com *N* nós e nível de profundidade *d*
  - $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$, onde

    N = total de nós expandidos para uma instância de problema

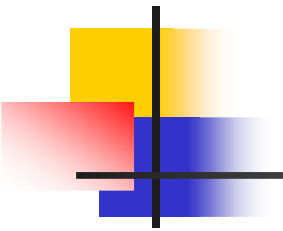
    d = profundidade da solução;
- Mede-se empiricamente a qualidade de *h* a partir do conjunto de valores experimentais de *N* e *d*.
  - uma boa função heurística terá o *b** muito próximo de 1.
- Se o custo de execução da função heurística for maior do que expandir nós, então ela *não* deve ser usada.
  - uma boa função heurística deve ser *eficiente* e *econômica*.

Experimento com 100 problemas

	Search Cost			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	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	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

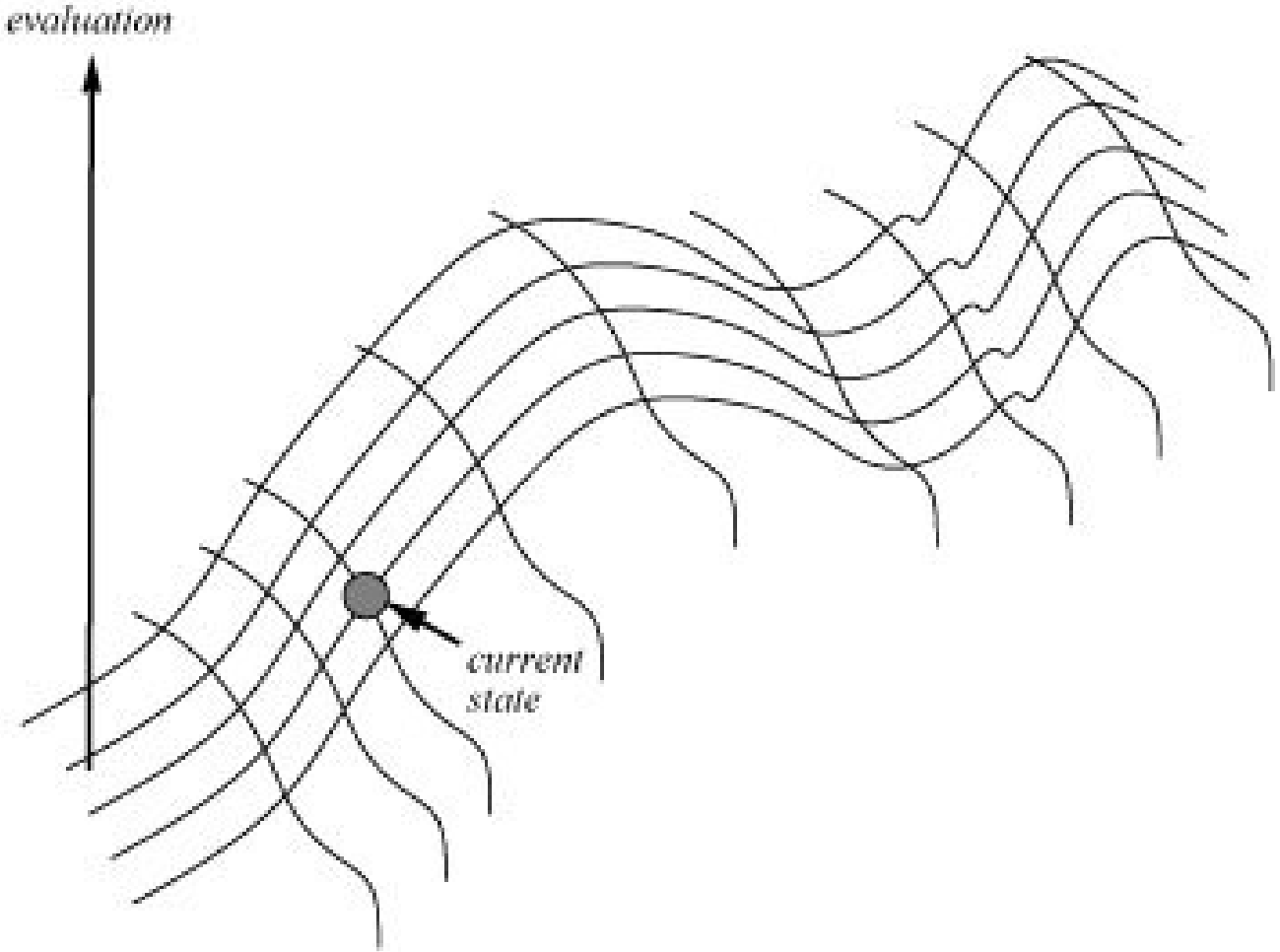
Uma boa função heurística terá o b^* muito próximo de 1.

Algoritmos de Melhorias Iterativas (AMI)
Iterative Improvement Algorithms



- A idéia é começar com o *estado inicial* (configuração completa, solução aceitável), e melhorá-lo iterativamente.
  - Imagem da TV
- Os estados estão representados sobre uma superfície (gráfico)
  - a altura de qualquer ponto na superfície corresponde à *função de avaliação* do estado naquele ponto
- O algoritmo se "move" pela superfície em busca de pontos mais altos (objetivos)
  - o ponto mais alto (máximo global) corresponde à solução ótima
    - nó onde a função de avaliação atinge seu valor máximo
- Aplicações: problemas de otimização
  - por exemplo, linha de montagem, rotas, etc.

Exemplo de Espaço de Estados



Algoritmos de Melhorias Iterativas

- Esses algoritmos guardam apenas o estado atual, e não vêem além dos vizinhos imediatos do estado.
- Contudo, muitas vezes são os melhores métodos para tratar problemas reais muito complexos.
- Duas classes de algoritmos:
  - **Hill-Climbing**: Subida da Encosta ou Gradiente Ascendente
    - só faz modificações que melhoram o estado atual.
  - **Simulated Annealing**: Anelamento Simulado
    - pode fazer modificações que pioram o estado temporariamente, para possivelmente o melhorará no futuro.

Subida da Encosta: algoritmo Hill-Climbing

- O algoritmo *não* mantém uma árvore de busca:
  - guarda apenas o estado atual e sua avaliação
  - É simplesmente um "loop" que se move na direção *crescente* (para maximizar) ou *decrescente* (para minimizar) da função de avaliação.
- Algoritmo:

função **Hill-Climbing (*problema*)** retorna **uma solução**

variáveis locais: *corrente* (o nó atual), *próximo* (o próximo nó)

corrente ← Faz-Nó(Estado-Inicial[*problema*])
loop do
próximo ← **sucessor** de *corrente* **de maior valor** (expande nó *corrente* e seleciona seu melhor filho)
se Valor[*próximo*] < Valor[*corrente*] (ou >, para minimizar)
então retorna *corrente* (o algoritmo pára)
corrente ← *próximo*
end

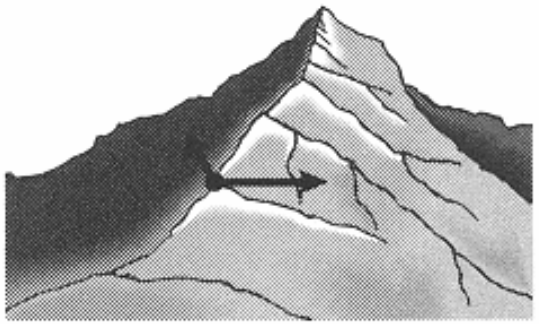
Exemplo de Subida da Encosta

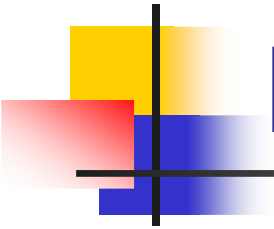
- Cálculo da menor rotas com 5 nós:
  - **estado inicial** = (N1, N2, N3, N4, N5)
  - **f** = soma das distâncias diretas entre cada nó, na ordem escolhida (*admissível!*)
  - **operadores** = permutar dois nós quaisquer do caminho
  - **restrição** = somente caminhos conectados são estados válidos
  - **estado final** = nó onde valor de f é mínimo

  - e1 = {N1, N2, N3, N4, N5}
    f(N1, N2, N3, N4, N5) = 10
  - e2 = {N2, N1, N3, N4, N5}
    f(N2, N1, N3, N4, N5) = 14
  - e3 = {N2, N1, N4, N3, N5}
    f(N2, N1, N3, N4, N5) = 9!!!

Subida da Encosta

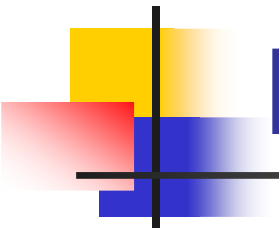
- O algoritmo move-se sempre na direção que apresenta maior taxa de variação para f
- Isso pode acarretar em 3 problemas:
  1. Máximos locais
  2. Planícies (platôs)
  3. Encostas e picos





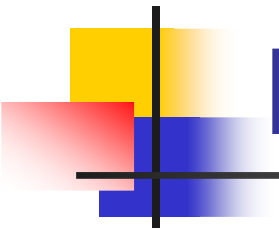
Máximos locais

- Definição
  - Em contraste com máximos globais, são picos mais baixos do que o pico mais alto no espaço de estados (solução ótima)
- A função de avaliação leva a um valor máximo para o caminho sendo percorrido
  - a função de avaliação é menor para todos os estados filhos do estado atual, apesar de o objetivo estar em um ponto mais alto
    - essa função utiliza informação "local"
  - e.g., xadrez: eliminar a Rainha do adversário pode levar o jogador a perder o jogo.



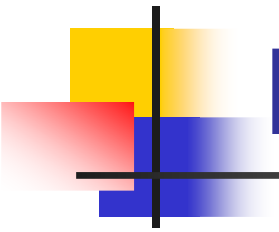
Máximos locais

- O algoritmo pára no máximo local
  - pois só pode mover-se com taxa crescente de variação
    - restrição do algoritmo
  - e.g., 8-números: mover uma peça para fora da sua posição correta para dar passagem a outra peça que está fora do lugar tem taxa de variação negativa!!!



Platôs (Planícies)

- Uma região do espaço de estados onde a função de avaliação dá o mesmo resultado
  - todos os movimentos locais são iguais (taxa de variação zero)
    - f(n) = f(filhos(n))
  - o algoritmo pára depois de algumas tentativas
    - restrição do algoritmo
  - ex. jogo 8-números: nenhum movimento possível vai influenciar no valor de f, pois nenhum número vai chegar ao seu local objetivo.



Encostas e Picos

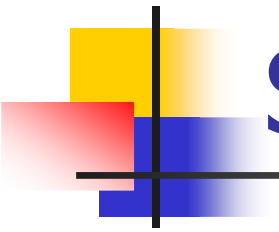
- Apesar de estar em uma direção que leva ao pico, nenhum dos operadores válidos conduz o algoritmo nessa direção
  - os movimentos possíveis têm taxa de variação zero ou negativa
    - restrição do problema e do algoritmo
  - ex. rotas: quando permutar dois pontos e o caminho resultante não está conectado.

Subida da Encosta

- Nos casos acima, o algoritmo chega a um ponto de onde não faz mais progresso.
- *Solução: reinício aleatório (random restart)*
  - O algoritmo realiza uma série de buscas a partir de estados iniciais gerados aleatoriamente.
- Cada busca é executada
  - até que um número máximo estipulado de iterações seja atingido, ou
  - até que os resultados encontrados não apresentem melhora significativa.
- O algoritmo escolhe o melhor resultado obtido com as diferentes buscas.
  - Objetivo!!!

Subida da Encosta: análise

- O algoritmo é completo?
  - **SIM,** para problemas de *otimização*
    - uma vez que cada nó tratado pelo algoritmo é sempre um estado completo (uma solução)
  - **NÃO,** para problemas onde os nós não são estados completos
    - e.g., jogo dos 8-números
    - semelhante à busca em profundidade
- O algoritmo é ótimo?
  - **TALVEZ,** para problemas de *otimização*
    - quando iterações suficientes forem permitidas...
  - **NÃO,** para problemas onde os nós não são estados completos

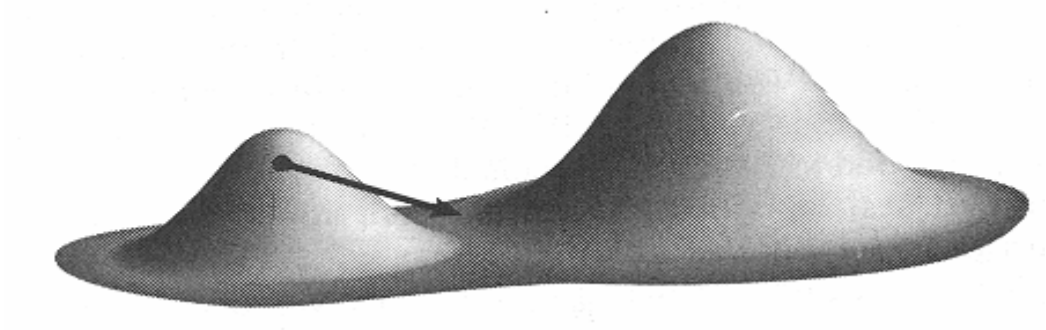


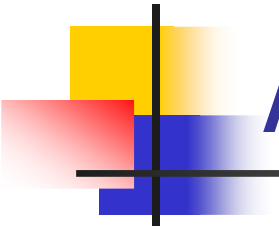
Subida da Encosta: análise

- O sucesso deste método depende muito do formato da superfície do espaço de estados:
  - se há poucos máximos locais, o reinício aleatório encontra uma boa solução rapidamente
  - caso contrário, o custo de tempo é exponencial.

Anelamento Simulado

- Este algoritmo é semelhante à Subida da Encosta, porém oferece meios para se escapar de máximos locais.
  - quando a busca fica "presa" em um máximo local, o algoritmo não reinicia a busca aleatoriamente
  - ele retrocede para escapar desse máximo local
  - esses retrocessos são chamados de *passos indiretos*
- Apesar de aumentar o tempo de busca, essa estratégia consegue escapar dos máximos locais
- Analogia com cozimento de vidros ou metais:
  - processo de resfriar um líquido gradualmente até ele se solidificar





Anelamento Simulado

- 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 a piora causada pelo movimento, $e^{-\Delta E/T}$, onde:

  ΔE = *Valor[próximo-nó] - Valor[nó-atual]*

  T = *Temperatura*

Anelamento Simulado: algoritmo

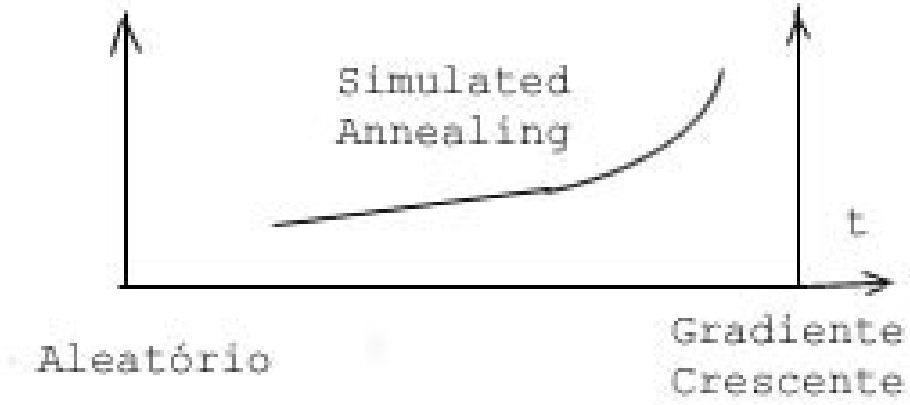
- função **Anelamento-Simulado** (***problema,mapeamento***)
  retorna **uma solução**

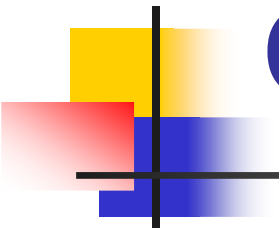
  variáveis locais: *corrente*, *próximo*, *T* (temperatura que controla a probabilidade de passos para trás)

  corrente ← Faz-Nó(Estado-Inicial[*problema*])
  for *t* ← 1 **to** ∞ **do**
    T ← mapeamento[*t*]
    Se *T* = 0
      então retorna *corrente*
    próximo ← um sucessor de *corrente* escolhido aleatoriamente
    ΔE ← Valor[*próximo*] - Valor[*corrente*]
    Se $\Delta E > 0$
      então *corrente* ← *próximo*
      senão *corrente* ← *próximo com probabilidade* = $e^{-\Delta E/T}$

Anelamento Simulado

- Com o tempo (diminuição da temperatura), este algoritmo passa a funcionar como Subida da Encosta.
- O algoritmo é ótimo 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 vai encontrar um máximo global ótimo.





Críticas à Busca Heurística

- *Solução de problemas* usando técnicas de *busca heurística:*
  - dificuldades em definir e usar a *função de avaliação*
  - não consideram conhecimento genérico do mundo (ou *"senso comum")*
- Função de avaliação: compromisso (conflito) entre
  - tempo gasto na seleção de um nó e
  - redução do espaço de busca
- Achar o melhor nó a ser expandido a cada passo pode ser tão difícil quanto o problema da busca em geral.