

Busca la muestra

---

---

---

---

---



## "BUSCA HEURÍSTICA"

- heurística pode ser definida como o "estudo dos dos métodos e regras de descoberta e invenção"
  - interpretações que nem da origem da palavra
- na busca em espaço de estados, heurísticas são formalizadas como regras para escolher aqueles ramos em um espaço de busca que tem maior probabilidade de levarem a uma solução aceitável do problema
- resoluedores de problemas de IA empregam heurísticas em duas situações:
  - ① problemas que podem não ter uma solução exata por causa das ambiguidades

diferentes na formulação do problema ou nos dados disponíveis. Ex:

- diagnóstico médico : Sintomas podem ter diferentes causas
- visão : cenas visuais são ambíguas (ilusões de ótica).

② Problema possui uma solução exata, mas o custo computacional de encontrá-la é muito alto (ou proibitivo). Ex:

- Xadrez

• Infelizmente, as heurísticas podem falhar.

Uma heurística é apenas uma conjectura informada sobre o próximo passo a ser dado na solução de um problema.

→ frequentemente ela é baseada na experiência e intuição

→ uma heurística pode levar um algoritmo a uma solução sub-ótima, ou até mesmo a não encontrar nenhuma solução.

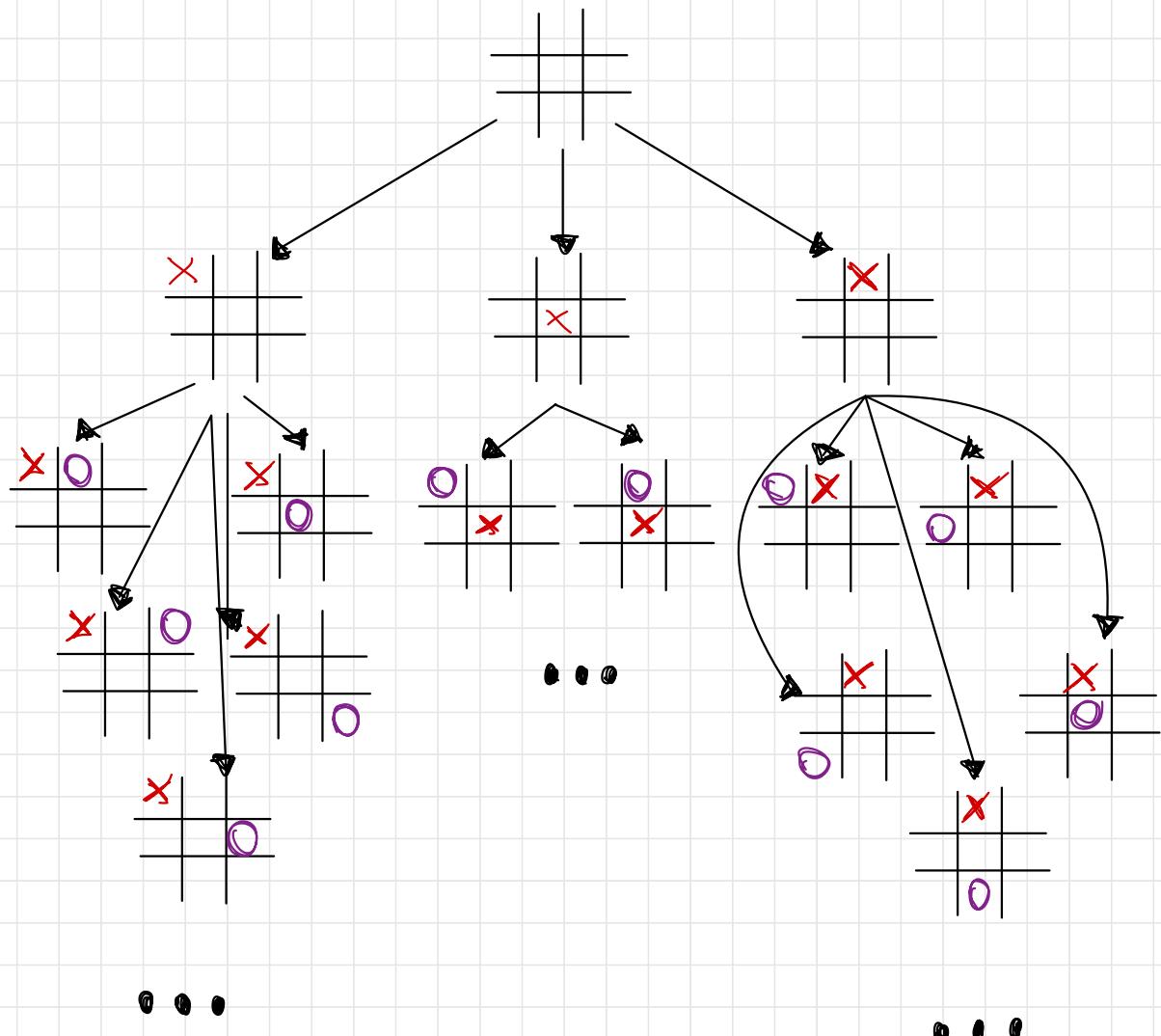
- É útil considerar a busca heurística sob duas perspectivas:
  - \* a medida heurística
  - \* um algoritmo que usa heurísticas para buscar um espoço de estados

- Alguns exemplos:

Subida de Encosta } Heurísticas  
Programação Dinâmica }

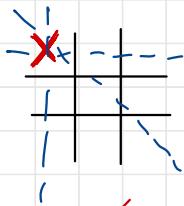
Busca pela Melhor Escolha } Algoritmo

EXEMPLO: 3 primeiros níveis do espaço de estados  
do jogo da velha reduzido por simetria

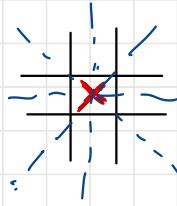


- Usando a representação do espaço por simetria reduz o espaço de estados de  $3^9$  para  $12 \times 7!$
- uma heurística simples pode reduzir ainda mais a busca: podemos nos mover para a configuração na qual X tem mais oportunidades de vitória
- Isto pode ser avaliado na figura 2.

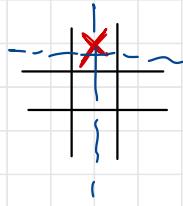
Fig 2: Heurística do "mais vitórias" aplicadas no nível 1 do jogo da velha



três vitórias  
para um X no  
canto



quatro vitórias  
para o X no  
centro



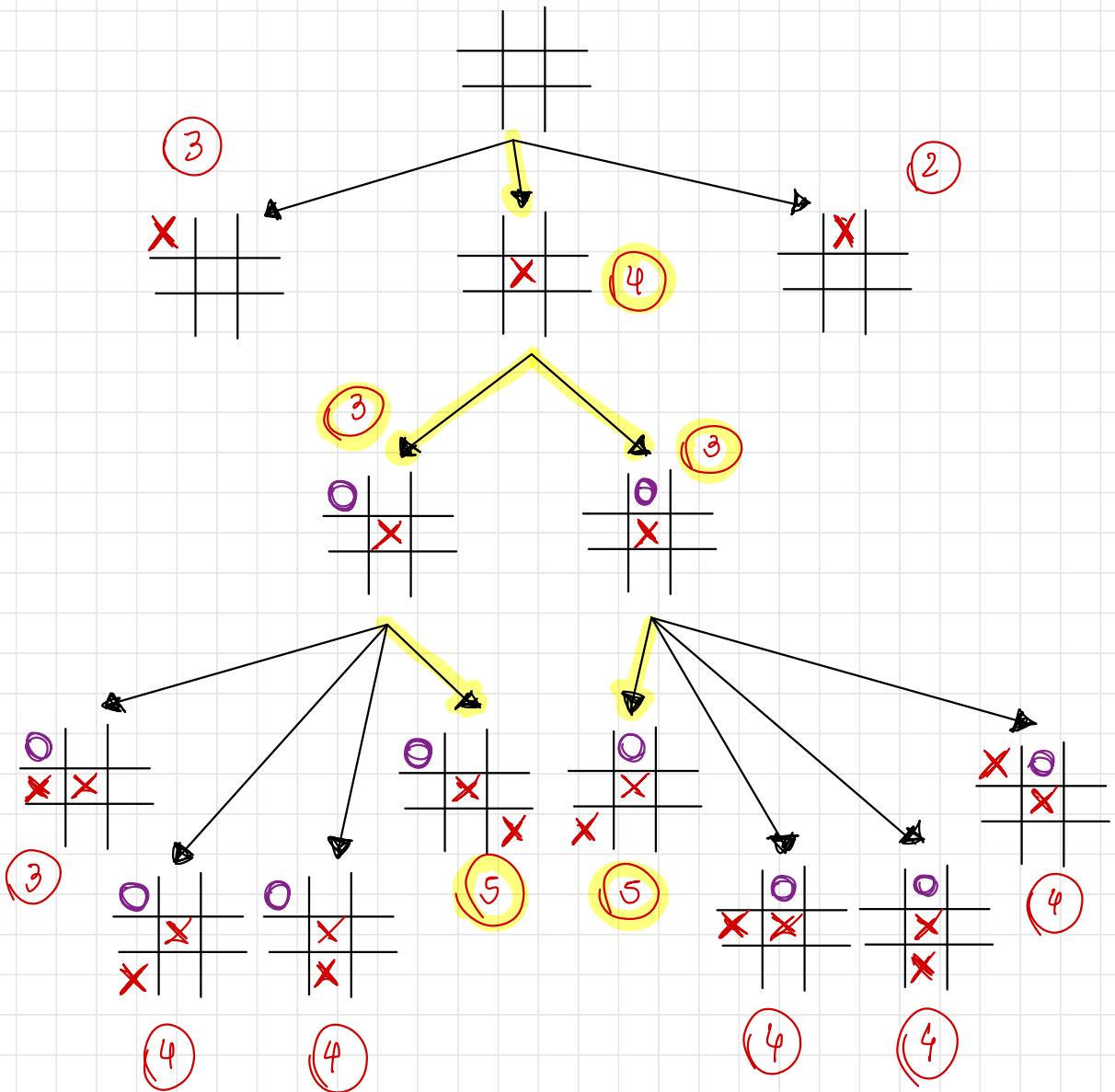
duas vitórias  
para um X no  
centro de um  
lado

- Independentemente de qual movimento seja escolhido, podemos aplicar a heurística ao estado resultante do jogo novamente usando a heurística das "mais vitória" para escolher os movimentos possíveis

- Conforme a busca continua, cada movimento avalia os filhos de um único nó, e uma busca exaustiva não é necessária
- Embora seja difícil calcular o número exato de estados que devem ser examinados, pode-se calcular um limite superior grosseiro, supondo-se um número máximo de 5 movimentos em um jogo com cinco opções por movimento:

$$25 < 9!$$

Fig 3. Espaço de Estados redimensionado teoricamente para o jogo da velha



**Exercício 01:** Fazer o esboço de busca heurística da Figura 3 até um estado objetivo

---

**Exercício 02:** Encontrar e avaliar uma heurística para o problema do quebra-cabeça de 8 peças.

Sugestão  $h_1(x)$ : Número de peças na posição correta

---

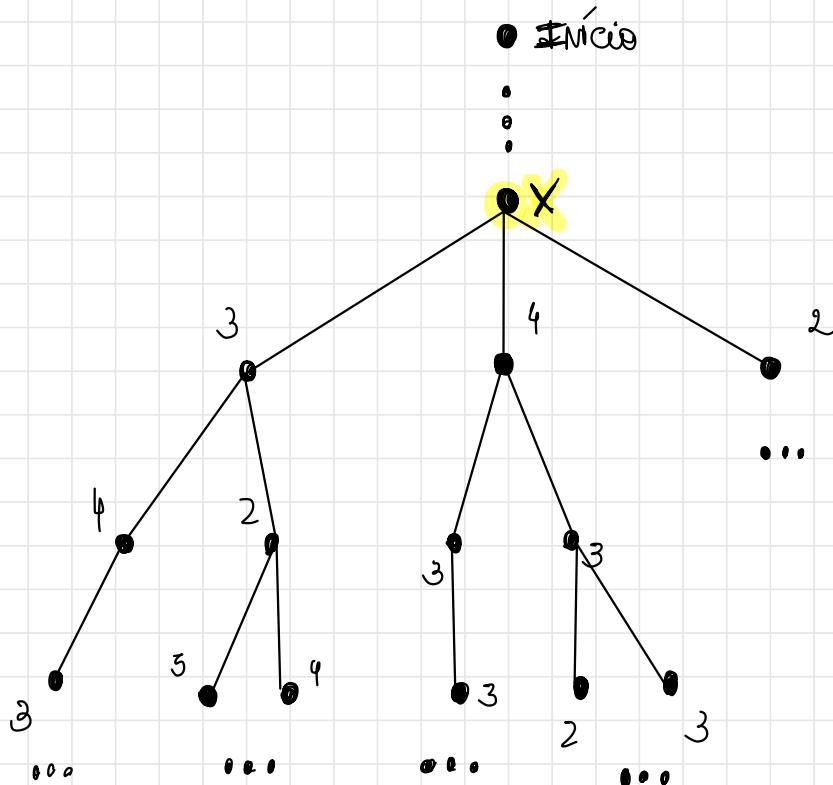
## \* 1. Subida de Encosta (Hill Climbing)

- a maneira mais simples de se implementar a busca heurística é por meio de um algoritmo chamado "subida de encosta"
  - ↳ expande o estado atual da busca e avalia seus filhos
  - ↳ o "melhor" filho é selecionado para uma expansão futura
  - ↳ nem seus irmãos, nem seus genitores são considerados
- analogia: alpinista cego, sempre segue o caminho mais íngreme até não poder avançar mais
- Como o algoritmo não registra o histórico do processo de subida, ele não consegue rever gran de falhas de sua estratégia

- O jogo da velha usando a heurística do caminho com o maior número de vitórias é um exemplo de estratégia de subida de encosta
- um problema da subida de encosta é sua tendência de ficar preso nos máximos locais
  - se ela alcança um estado que tenha uma avaliação melhor que qualquer um de seus filhos, o algoritmo fracassa
  - se esse estado não for um objetivo, mas apenas um máximo local, o algoritmo pode nunca achar a solução
- A figura 4 mostra um exemplo do dilema do máximo local. Suponha que explorando o espaço de busca chegamos ao estado X.

→ as avaliações dos filhos, netos e bisnetos de X demonstraram que a subida de vencosta pode se confundir com a antecipação de vários níveis

Fig 4. O problema do máximo local para a subida de vencosta com antecipações de 3 níveis



- há formas de contornar esse problema, como perturbar aleatoriamente a função de avaliação, mas em geral não há um modo de garantir o desempenho ideal com técnicas de subida de encosta

- \* programa de Damas de Samuel (1959)
  - aplicava busca heurística ao jogo de damas
  - implementava algoritmos para o uso ideal da memória limitada, e uma forma simples de aprendizado
  - programa avaliava estados do tabuleiro como a soma ponderada de várias medidas heurísticas diferentes:

$$\sum_i a_i x_i$$

→ o  $\alpha_i$  representa características do tabuleiro

Com elas:

- vantagem das peças
- local das peças
- controle do centro do tabuleiro
- etc

→ os coeficientes  $\alpha_i$  desses  $\alpha_i$  eram pesos ajustados que modelavam a importância desse fator na avaliação geral do tabuleiro

→ o programa antecipava o número desejado de níveis, ou camadas, e avaliava a melhor jogada

- era uma variação do minimax

**Exercício:** Implemente o algoritmo de Subida de Encosta para o quebra-cabeça de 8 peças.

2	8	3
1	6	4
7		5

estado  
inicial

1	2	3
8		4
7	6	5

estado  
objetivo

## \* 2. Algoritmo de Busca pela Melhor Escolha

↳ apesar de suas limitações, os algoritmos como busca com retrocesso, subida de encosta e programação dinâmica podem ser usados eficientemente se suas funções de avaliação forem suficiente informativas para evitar: máximos locais, becos sem saída e anomalias do espaço de busca

↳ em geral, o uso da busca heurística requer um algoritmo mais flexível

- busca pela melhor escolha
- fila de prioridade

↳ igual DFS e BFS, a busca pela melhor escolha usa listas para manter seus estados

- ABERTOS: registrar a fronteira atual da busca

• **Fechados**: registrar os estados já visitados

↳ uma etapa adicional ordena os estados em ABERTOS de acordo com uma estimativa

“heurística” de sua proximidade com um objetivo

↳ cada iteração do laço considera o estado “mais promissor” na lista ABERTOS

---

ALGORITMO BUSCA PELA MELHOR ESCOLHA

---

busca\_melhor\_escolha (Início)

1.  $\text{ABERTOS} = [\text{Início}]$  // Inicialização
2.  $\text{Fechados} = []$
3. Enquanto  $\text{ABERTOS} \neq []$  faça

- 4.
- retire o estado mais à esquerda de **ABERTOS** e chame-o de X

5. Se  $X = \text{objetivo}$  então:

- 6.
- retorne o caminho de Inicial até X

7. Senão

- 8.
- gera filhos de X

9. Para cada filho de X faça:

10. Caso:

- 11.
- o filho não está em **ABERTOS** ou **FECHADOS**

12. → atribua ao filho um valor heurístico

13. → acrescente o filho a **ABERTOS**

- 14.
- o filho já está um **ABERTOS**

15. → se o filho foi alcançado por um caminho mais certo

16.

→ então dê ao estado em  
**Abertos** o caminho mais  
curto

17.

- o filho já está em **Fechados**  
→ se o filho foi alcançado  
por um caminho mais cur-  
to, então

18.

19.

\* retire o estado de  
**Fechados**

20.

\* Aumente o filho em  
**Abertos**

21.

fim caso

22.

fim para

23.

fim se/senão

24.

- Coloque X em **Fechados**

25.

- reordene estados em **Abrertos** pelo método heurístico (melhor mais à esquerda)

26.

fim enquanto

27.

retorna **FALHA**

// Abrertos está vazio

28. fim algoritmo

---

- a cada iteração o algoritmo **busca melhor escolha** retira o primeiro elemento da lista

**Abrertos**.

→ Se encontrar as condições de objetivo, o algoritmo reforma o caminho de solução que levou ao objetivo

**OBS:** note que cada estado retém informações do ancestral para determinar se ele tinha sido alcançado anteriormente por um caminho

mais curto e permitir que o algoritmo encontre o caminho de solução

→ se o primeiro elemento em Aceitos não for um objetivo, o algoritmo aplica todas as regras ou operadores de produção que combinam para gerar seus descendentes

→ se um estado filho não estiver em Aceitos ou Fechados, o algoritmo aplica uma avaliação heurística a esse estado, e a lista Aceitos é ordenada. Isto leva os melhores estados para o começo da lista

OBS: Note que este "melhor" pode ser de qualquer nível do espaço de busca. A lista Aceitos é mantida como uma fila de prioridades

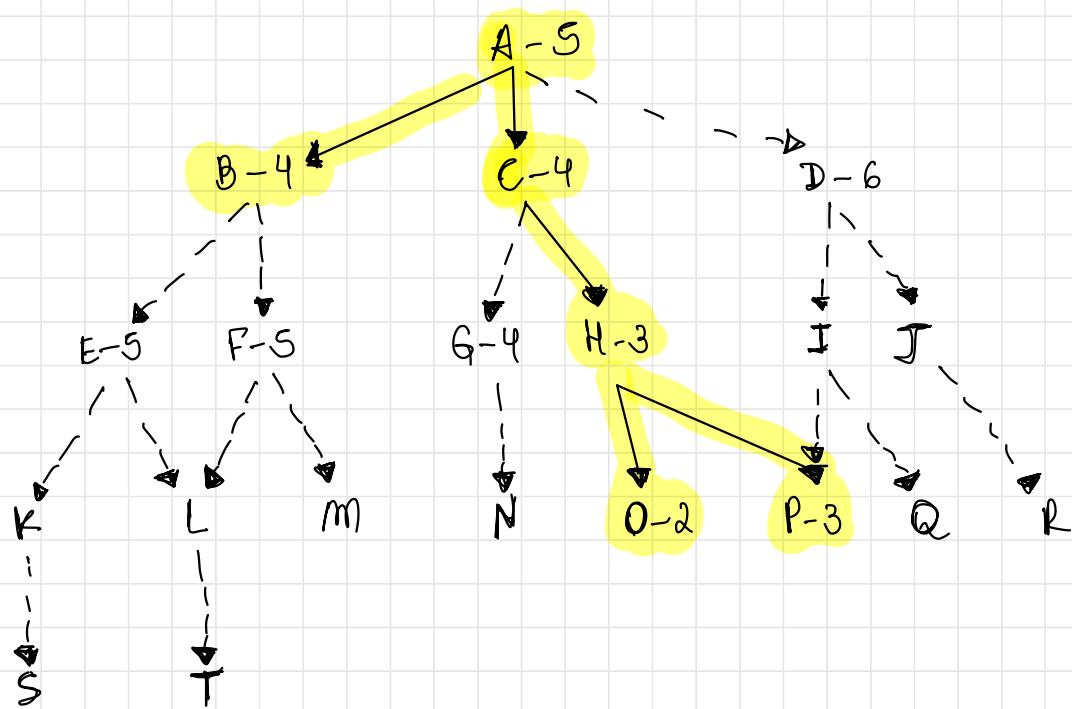
→ se um estado filho já estiver em ~~Abertos~~  
ou ~~Fechados~~, o algoritmo verifica para ter  
certeza de que o estado registra o mais  
curto dos dois caminhos parciais da solução.

Estados duplicados não são refidos.

OBS: atualizando o histórico de caminho dos  
nós em ~~Abertos~~ e ~~Fechados~~ quando eles  
são redescobertos, o algoritmo achará um  
caminho mais curto para um objetivo.

- a Figura 5 mostra um espaço de  
estados hipotéticos com avaliações heurísticas  
ligadas a alguns de seus estados. Os esta-  
dos conectados são gerados pelo processo de  
busca. Os estados deslocados foram expandidos  
pelo algoritmo

Fig 5. Busca heurística de um espaço de estados hipotético



- Note que o algoritmo não explora todo o espaço de busca. O objetivo do algoritmo é encontrar o estado objetivo examinando o mínimo de estados possível  
→ quanto mais “informada” a heurística, menos estados são processados

- Execução do Espaço Hipotético da Figura 5
  - P é o estado objetivo
  - os estados ao longo do caminho até P têm devido a ter valores heurísticos mais baixos
  - a heurística é falível: o estado O tem um valor mais baixo que o próprio objetivo e é examinado primeiro.
  - diferentemente do algoritmo de Subida de Encosta, que não mantém uma fila de prioridades para a seleção dos "próximos" estados, o algoritmo se recupera desse erro e determina o objetivo corretamente.

1. **ABERTOS** = [A5], **FECHADOS** = []

2. Avaliar A5, **ABERTOS** = [B4, C4, D6], **FECHADOS** = [A5]

3. Avaliar B4, **ABERTOS** = [C4, E5, F5, J6]

**FECHADOS** = [B4, A5]

4. Avaliar C<sub>4</sub>, **ABERTOS** = [H<sub>3</sub>, G<sub>4</sub>, E<sub>5</sub>, F<sub>5</sub>, D<sub>6</sub>]  
**FECHADOS** = [C<sub>4</sub>, B<sub>4</sub>, A<sub>5</sub>]

5. Avaliar H<sub>3</sub>, **ABERTOS** = [O<sub>2</sub>, P<sub>3</sub>, G<sub>4</sub>, E<sub>5</sub>, F<sub>5</sub>, D<sub>6</sub>]  
**FECHADOS** = [H<sub>3</sub>, C<sub>4</sub>, B<sub>4</sub>, A<sub>5</sub>]

6. Avaliar O<sub>2</sub>, **ABERTOS** = [P<sub>3</sub>, G<sub>4</sub>, E<sub>5</sub>, F<sub>5</sub>, D<sub>6</sub>]  
**FECHADOS** = [O<sub>2</sub>, H<sub>3</sub>, C<sub>4</sub>, B<sub>4</sub>, A<sub>5</sub>]

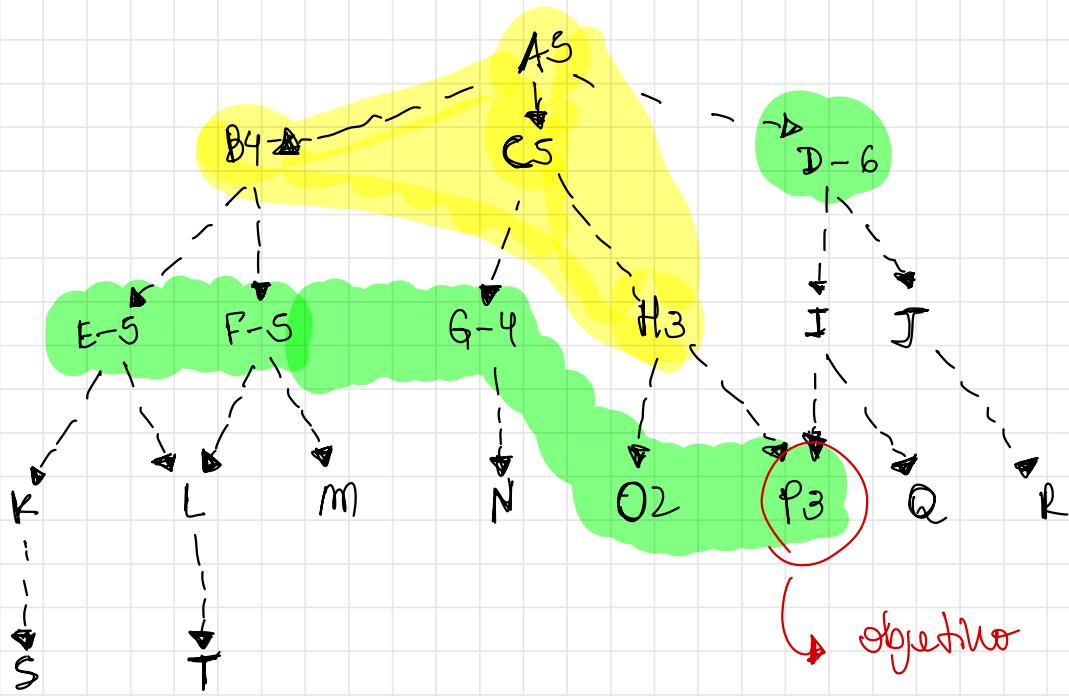
7. Avaliar P<sub>3</sub>: a solução foi encontrada.

---

A Figura 6 mostra o espaço de busca e lista após a quinta iteração. Note que a fronteira é irregular, refletindo a natureza oportunista da busca pela melhor escolha.

- Na busca pela melhor escolha, a lista de ABERTOS permite o retrocesso de caminhos que faltaram em produzir um objetivo

Fig 6. Busca heurística de um espaço de estados hipotético com espaços realçados



Estados Abertos

Estados Fechados

### \* 3. Implementando Funções de Avaliação Heurística

- para o problema do quebra-cabeça de 8 peças, a heurística mais simples conta, em cada estado, o número de peças que estão fora do lugar em relação ao objetivo

→ porém não são usadas todas as informações disponíveis em uma configuração de tabuleiro, porque não leva em conta as distâncias que as peças devem ser movidas

→ uma heurística melhor somaria todas as distâncias das peças fora de lugar, adicionando uma unidade para cada quadrado que uma peça precisa ser movida para alcançar a posição do estado - objetivo

- vestas duas heurísticas não percebem a dificuldade da inversão de peças, isto é, se duas peças forem vizinhas, e o objetivo requer que elas sejam opostas, levará (muito) mais que dois movimentos para colocá-las no lugar desejado

Fig 7. Exemplo de inversões em um estado:  
1 e 2, 5 e 6

2	1	3
8		4
7	5	6

1	2	3
8		4
7	6	5

OBJETIVO

→ uma heurística que leva isso em consideração multiplica um número pequeno (ex: 2) por cada inversão

direta de pegar.

- Um exemplo de aplicação destas três heurísticas a diferentes estados, pode ser vista figura 9.

Fig 8 : Estado do Quebra-Cabeça de 8 páginas e movimentos possíveis

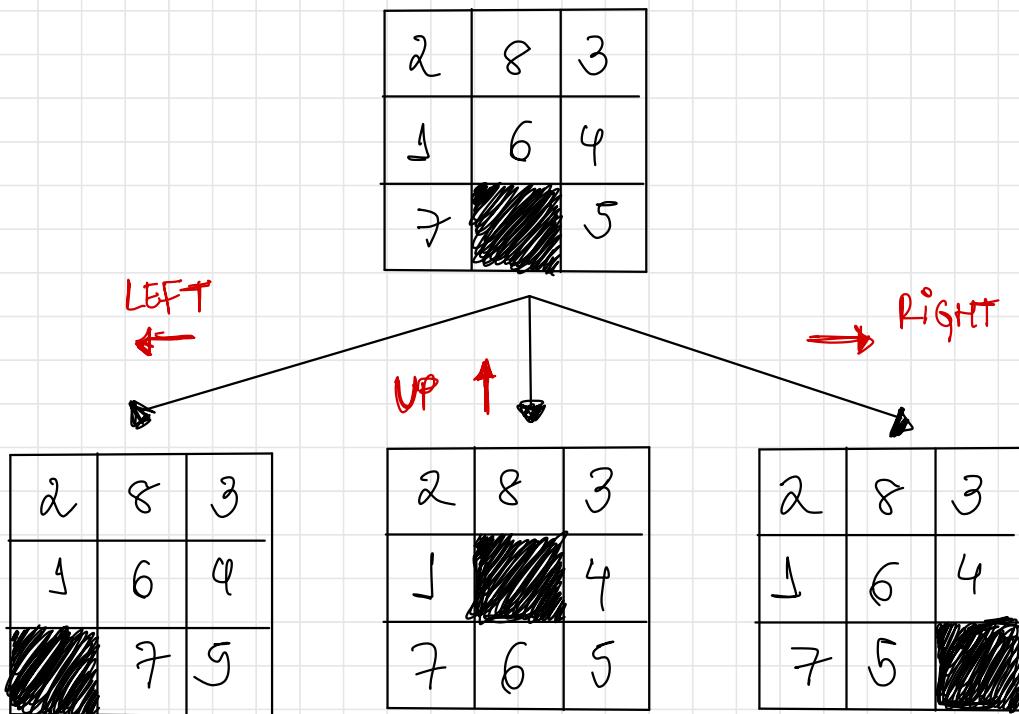


Fig 9: Três heurísticas aplicadas a estados da Fig 8

2	8	3
1	6	4
7	5	

5

6

0

2	8	3
1		4
7	6	5

3

4

0

2	8	3
1	6	4
7	5	

5

6

0

OBJETIVO

1	2	3
8		4
7	6	5

H1: pegar  
fora do  
lugar

H2: Soma das  
distâncias  
fora do lugar

H3: 2x o  
número de  
inversões  
diretas

- Algunas Observações:

- \* a  $H_2$  (soma das distâncias) pode fornecer uma estimativa mais precisa do trabalho, do que somente a contagem
- \* a  $H_3$  (número de inversões) não consegue distinguir entre os estados, porque nenhum tem inversões diretas
- \* uma quarta heurística ( $H_4$ ) poderia conformar os problemas e incluir vários aspectos combinando  $H_2$  e  $H_3$
- \* exemplo ilustra a dificuldade de elaborar boas heurísticas

- Se dois estados tiverem a mesma, ou quase a mesma, avaliação heurística, geralmente é preferível examinar o estado que está mais próximo da raiz do grafo

→ esse estado terá uma maior probabilidade de estar no caminho mais certo

→ A distância do estado inicial até os seus descendentes pode ser medida se mantivermos uma contagem de profundidade para cada estado.

- essa contagem é 0 para o estado inicial
- incrementada em 1 para cada nível de busca
- pode ser acrescentada à avaliação heurística de cada estado para orientar a busca em favor de estados mais superficiais

• Isso faz com que a nossa função de avaliação  $f$  seja a soma de dois componentes:

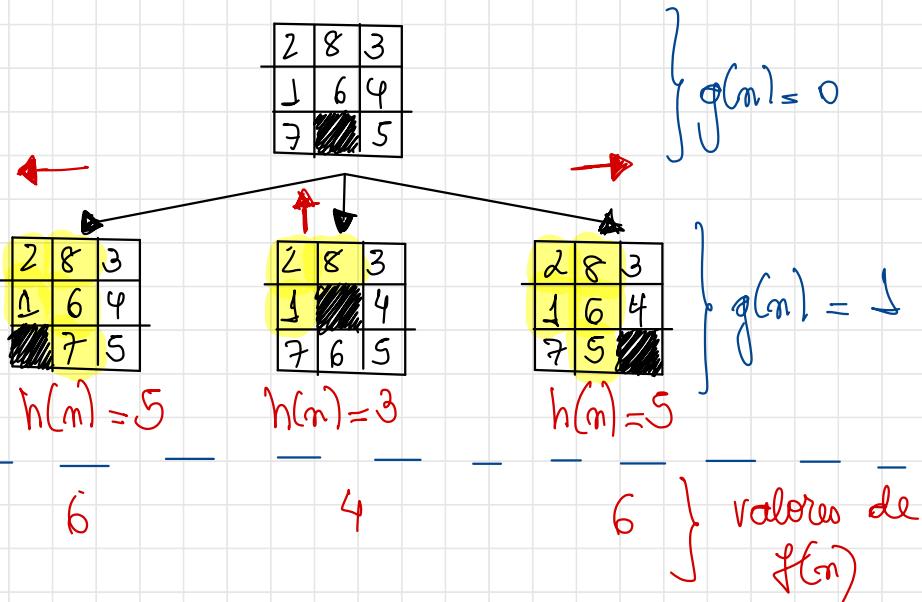
$$f(n) = g(n) + h(n)$$

onde:

- $g(n)$  mede o comprimento real do caminho de um estado  $n$  qualquer até o estado inicial

- $h(n)$  é uma estimativa heurística da distância entre o estado  $n$  e um objetivo
- no quebra-cabeça de 8 peças  $h(n)$  pode ser o número de peças fora do lugar

Fig 10: Heurística  $f$  aplicada no quebra-cabeça de 8 peças



onde:

$$f(n) = g(n) + h(n)$$

$g(n)$  = distância real de  $n$  até o estado inicial

$h(n)$  = número de peças fora do lugar

Fig 11: Espaço de estados gerado pela heurística f

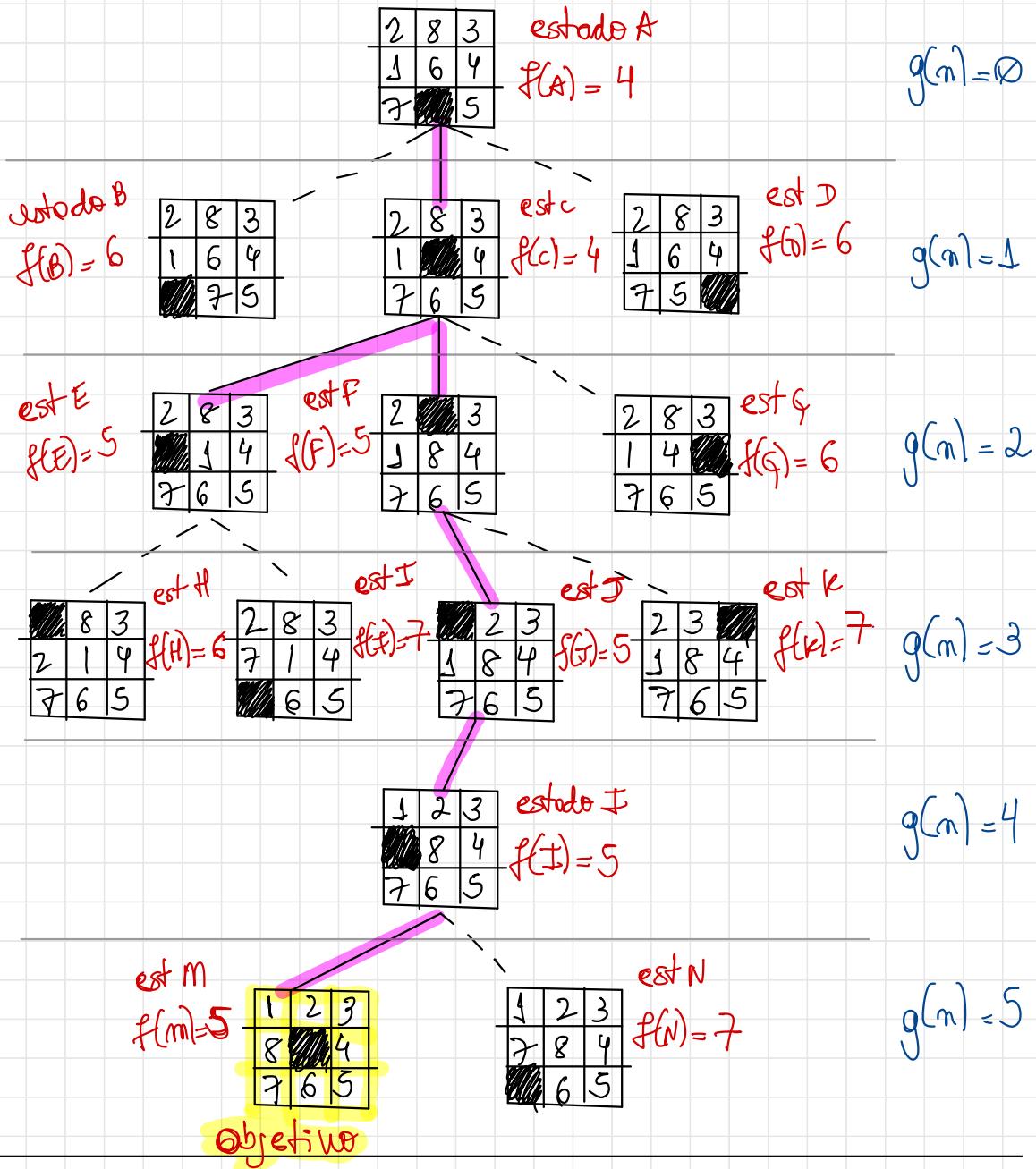
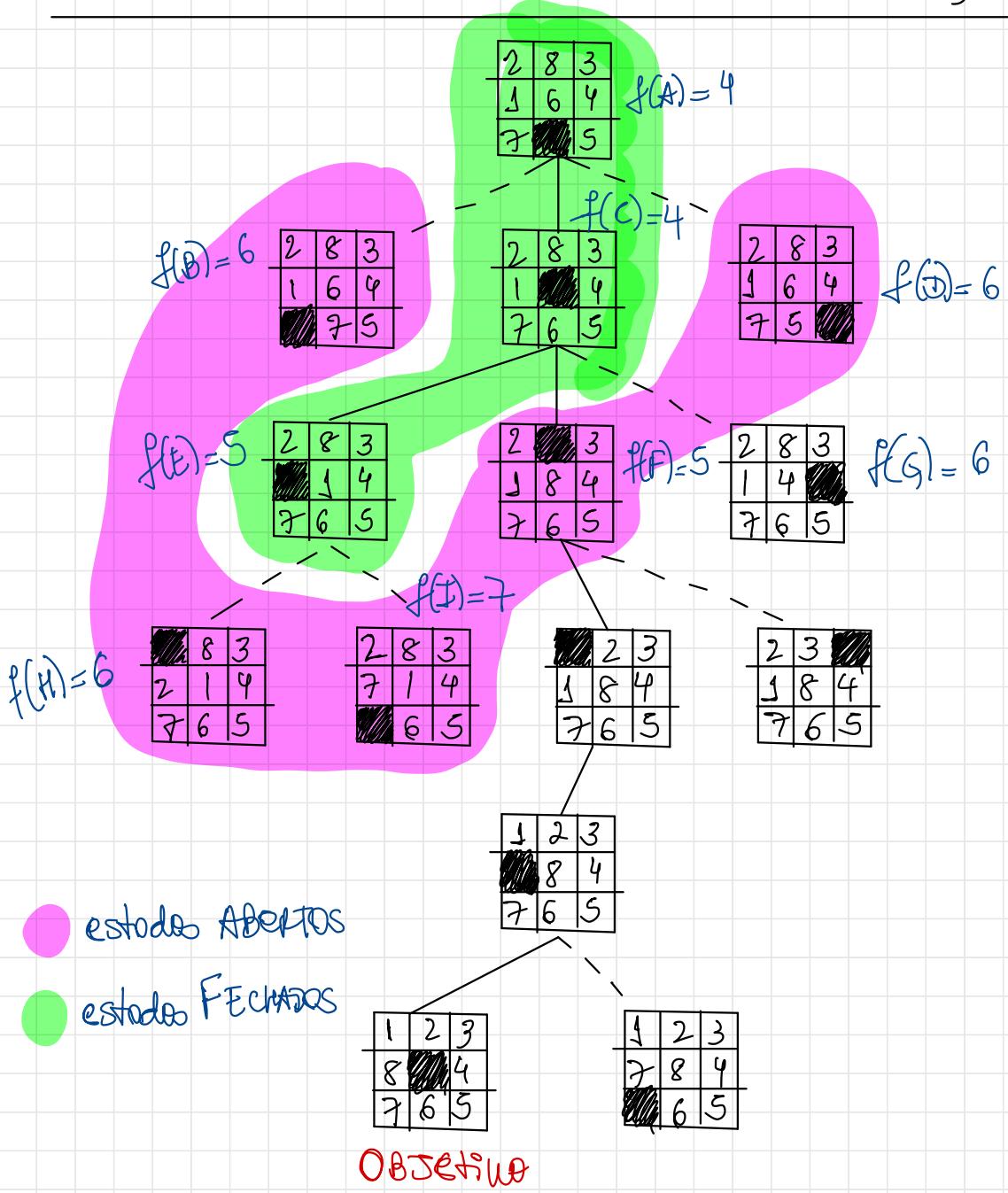


Fig 12: ABERTOS e FECHADOS após a terceira iteração



- A busca pela melhor escolha, usando a heurística  $f(n)$  pode garantir a produção do caminho mais curto até um objetivo;

1. Operações sobre estados geram filhos do estado atualmente examinado
2. Cada estado novo é verificado para ver se ocorreu antes (está em **ABERTOS** ou **FECHADOS**), impedindo assim os laços
3. Cada estado  $n$  recebe um valor  $f$  igual a soma de sua profundidade no espaço de busca  $g(n)$  e uma estimativa heurística de sua distância até um objetivo  $h(n)$ 
  - \*  $f$  guia a busca a estados promissores
  - \*  $g$  impede que a busca persista em um caminho infrutífero
4. Estados em **ABERTOS** são ordenados por seus valores de  $f$
5. A eficiência do algoritmo pode ser melhorada se **ABERTOS** e **FECHADOS** sejam

## Implementadas como Heaps

- a busca pela melhor escolha é um algoritmo genérico para pesquisar, de modo heurístico, qualquer grafo de estados
- 

**EXERCÍCIO:** Implemente a busca pela melhor escolha com uma heurística  $f$  para o problema do quebra-cabeça de 8 peças

---

## \* 4. Admissibilidade, Monotonicidade e Grau de Informação

- podemos avaliar o comportamento de heurísticas ao longo de várias dimensões:
  - \* encontrar solução boa
  - \* menor caminho
  - \* etc
- Heurísticas que encontram o caminho mais curto até um objetivo, sempre que esse existir, são chamadas de **admissíveis**
- Em que sentido uma heurística é "melhor" que outra?
  - Isto é o grau de informação de uma heurística
- **Monotonicidade:** uma garantia de que o mesmo estado não será descoberto mais adiante na busca com um custo menor

## • 4.1 - Medidas de Admissibilidade

↳ um algoritmo de busca é admissível se ele seguramente encontrar um caminho mínimo até uma solução, sempre que tal solução exista.

\* a busca em amplitude é uma estratégia de busca admissível

↳ usando a função de avaliação  $f(n) = g(n) + h(n)$  podemos caracterizar uma classe de estratégias de busca admissíveis. Para isso definimos uma função de avaliação  $f^*$ :

$$f^*(n) = g^*(n) + h^*(n)$$

onde:

•  $g^*(n)$  é o custo do caminho mais curto do nó inicial ao nó  $n$

•  $h^*(n)$  reforma o custo real do menor caminho de  $n$  até o objetivo

- Com isso,  $f^*(n)$  é o custo real do caminho ótimo partindo de um nó inicial até um nó objetivo; passando pelo nó  $n$ .

↳ quando empregamos busca pela melhor escoilha com a função de avaliação  $f^*$ , a estratégia de busca resultante é admissível

\* embora órculos como  $f^*$  não existem para a maioria dos problemas reais, desejamos que  $f$  seja uma estimativa próxima de  $f^*$

\*  $g(n) =$  custo do caminho atual até  $n$  é uma estimativa razoável de  $f$

\*  $h(n) =$  estimativa de custo mínimo até um objetivo.

limite superior

{ Se  $h(n) \leq h^*(n)$ , o algoritmo que usa  $f$  é chamado de  $A^*$

## ALGORITMO A, ADMISSIBILIDADE A\*

- Considere a função de avaliação  $f(n) = g(n) + h(n)$  onde:
  - \*  $g(n)$  é o custo de  $n$  a partir do estado inicial
  - \*  $h(n)$  é a estimativa heurística do custo de ir de  $n$  até um objetivo
- Se essa função for usada com o algoritmo busca pela melhor escolha, o resultado é o chamado **algoritmo A**
- Um algoritmo de busca é **admissível** se, para qualquer grafo, ele sempre terminar no caminho de seleção ótimo, sempre que existir um caminho entre os estados inicial e objetivo
- Se o algoritmo A for usado com uma função de avaliação, na qual  $h(n)$  é menor que o custo do caminho mínimo de  $n$  para um objetivo, de igual a ele, o algoritmo de

busca será chamado de A\* (A ESTRELA)

- Todos os algoritmos A\* são admissíveis
- 

↳ a busca em amplitude pode ser considerada um algoritmo A\* onde  $f(n) = g(n) + \theta$

↳ várias heurísticas para o quebra-cabeça dos 8 fornecem exemplos de algoritmos A\*.

\*  $h_1(n)$ : Contar as peças fora de posição é certamente menor que, ou igual, o número de movimentos necessários para mover-las até sua posição objetivo.

→ é uma heurística admissível

→ garante caminho de solução ótimo, ou o mais curto, quando existir

\*  $h_2(n)$ : Soma das distâncias diretas das peças fora de lugar é também admissível

\*  $h_3(n)$ : uso de multiplicadores para inversão também é !

↳ Muitas embora o custo real do caminho mais curto até um objetivo não possa sempre ser calculável, muitas vezes podemos provar que uma heurística é limitada "por cima" por este valor

- quando isto puder ser feito, a busca resultante terminará descobrindo o caminho mais curto até o objetivo

#### 4.2- Monotonicidade

↳ algoritmos A\* não querem  $g(n) < g^*(n)$ . Isso significa que heurísticas admissíveis podem inicialmente alcançar estados que não são objetivos ao longo de um caminho subótimo, desde que, no final o algoritmo encontre um caminho ótimo

## MONOTONIA

- Uma função heurística  $h$  é monotônica se

1. Para todo estado  $n_i$  e  $n_j$ , onde  $n_j$  é um descendente de  $n_i$

$$h(n_i) - h(n_j) \leq \text{custo}(n_i, n_j)$$

onde  $\text{custo}(n_i, n_j)$  é o custo real (em número de movimentos) para ir do estado  $n_i$  para  $n_j$

2. A avaliação heurística do estado objetivo é zero, ou seja,  $h(\text{objetivo}) = 0$

↳ em outras palavras, a diferença entre as medidas heurísticas para um estado e para qualquer um de seus sucessores é limitada pelo custo real de ir desse estado para o seu sucessor

- ↳ se um algoritmo de busca pela melhor escolha for usado com heurística monotônica, o algoritmo sempre encontra o menor caminho para qualquer estado na primeira vez que ele é descoberto
- ↳ qualquer heurística monotônica é admissível

### 4.3 - Heurísticas Mais Informadas

- Para duas heurísticas  $A^*$   $h_1$  e  $h_2$ , se:  
 $h_2(n) \leq h_1(n)$ , para todos os estados  $n$  do espaço de busca, então diz-se que a heurística  $h_2$  é mais informada que  $h_1$ .
- ↳ em geral, quanto mais informado for um algoritmo  $A^*$ , menos espaço ele precisará expandir até chegar a solução final

