1

Buscando Soluções

Busca Heurística

Prof. Patrick Pedreira Silva

2

Busca Heurística ou Informada

- **Estratégias de Busca Cega**
  - encontram soluções para problemas pela geração *sistemática* de novos estados, que são comparados ao objetivo;
  - são *ineficientes* na maioria dos casos:
    - *são capazes de calcular* apenas o custo de caminho *do nó atual ao nó inicial (função* g*), para decidir qual o próximo nó da fronteira a ser expandido.*
    - *essa medida não necessariamente conduz a busca na direção do objetivo.*
  - Como encontrar um barco perdido?
    - *não podemos procurar no oceano inteiro...*

3

Busca Heurística

- **Estratégias de Busca Heurística**
  - utilizam ***conhecimento específico*** do problema na escolha do próximo nó a ser expandido
  - barco perdido
    - *correntes marítimas, vento, etc...*
- **Aplica de uma *função de avaliação* a cada nó na fronteira do espaço de estados**
  - essa função ***estima o custo de caminho*** do nó atual até o objetivo mais próximo utilizando uma *função heurística*
- **Heurística**
  - do grego heuriskein, encontrar, descobrir
  - introduzida em IA por George Polya em 1957 (livro How to Solve It)
  - é conhecimento e, por isso, marcou quebra da IA com a pesquisa operacional

4

Funções Heurísticas

- **Função heurística (h)**
  - estima o custo do caminho mais barato do estado atual até o estado final mais próximo.
  - são específicas para cada problema
- **Exemplo:**
  - encontrar a rota mais curta entre duas cidades (Conquista-Ilhéus)
  - $h_{dd}(n)$ = distância direta entre o nó n e o nó final.
- **Como escolher uma boa função heurística?**
  - ela deve ser admissível, isto é, nunca superestimar o custo real da solução
  - ex. distância direta (h_{dd}) é admissível porque o caminho mais curto entre dois pontos é sempre uma linha reta
- **Classes de algoritmos para busca heurística**
  - Busca pela melhor escolha (Best-First search)
  - Busca com limite de memória
  - Busca com melhora interativa

5

Busca pela Melhor Escolha (BME)
Best-First Search

- **Busca genérica onde o nó de *menor custo "aparente"* na fronteira do espaço de estados é expandido primeiro**
- **Duas abordagens básicas:**
  1. Busca Gulosa (Greedy search)
  2. Algoritmo A* (pronunciado a-**estrela**) e suas variantes
- **Algoritmo:**

Função-Insere **-** ordena nós com base na Função-Avaliação

função Busca-Melhor-Escolha (*problema*,Função-Avaliação)

retorna uma solução

Busca-Genérica (*problema*, Função-Insere)

6

Busca Gulosa

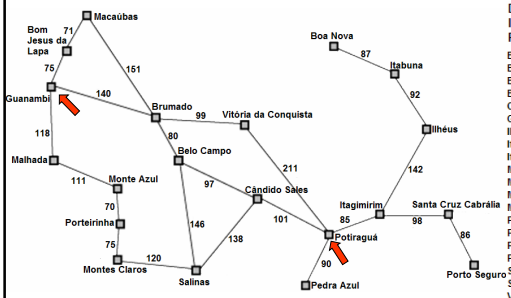
- **Semelhante à busca em profundidade com *backtracking***
- **Tenta expandir o nó mais próximo do nó final com base na estimativa feita pela *função heurística h***
- **Algoritmo:**

função Busca-Gulosa (*problema*)

retorna uma solução ou falha

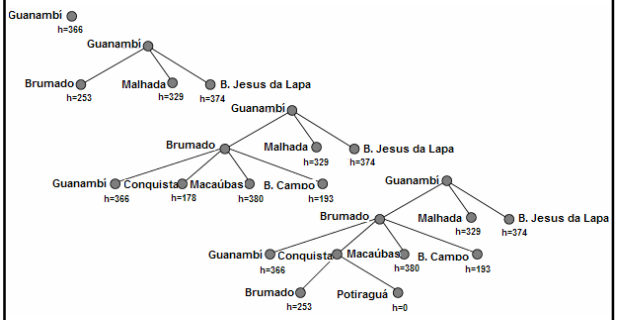
Busca-Melhor-Escolha (*problema*, *h*)

Exemplo: Ir de Guanambi a Potiraguá



Distância em linha reta para Potiraguá	
Belo Campo	193
Boa Nova	234
B. Jesus da Lapa	374
Brumado	253
Cândido Sales	98
Guanambi	366
Ilhéus	199
Itabuna	226
Itagimirim	80
Macaúbas	380
Malhada	329
Monte Azul	244
Montes Claros	242
Pedra Azul	77
Porteirinha	241
Potiraguá	0
Porto Seguro	161
Salinas	160
S. Cruz Cabrália	151
V. da Conquista	178

Busca Gulosa...



Busca Gulosa

9

- **Custo de busca mínimo!**
  - No exemplo, não expande nós fora do caminho
- **Porém *não* é ótima:**
  - No exemplo escolhe o caminho que é mais econômico à primeira vista, via Conquista
    - *Guanambi, Brumado, Conquista, Potiraguá = 450 (140+99+211)*
  - porém, existe um caminho mais curto via Belo Campo
    - *Guanambi, Brumado, Belo campo, Cândido Sales, Potiraguá = 418 (140+80+97+101)*
- **A solução via Conquista foi escolhida por este algoritmo porque:**
  - h_{dd}(Conquista)=178, enquanto h_{dd} (Belo Campo)=193
- **Não é completa:**
  - pode entrar em loop se não detectar a expansão de estados repetidos
  - pode tentar desenvolver um caminho infinito
- **Custo de tempo e memória:** $O(b^d)$
  - Guarda todos os nós expandidos na memória

Algoritmo A*

10

- **É ainda a técnica de busca mais usada**
- **Tenta minimizar o custo total da solução combinando:**
  - Busca Gulosa: econômica, porém não é completa nem ótima
  - Busca de Custo Uniforme (Djikstra): ineficiente, porém completa e ótima
- **Função de avaliação:**
  - f (n) = g (n) + h (n)
  - g (n) = distância de n ao nó inicial
  - h (n) = distância estimada de n ao nó final
- **A* expande o nó de menor valor de f na fronteira do espaço de estados.**

Algoritmo A*

11

- **Se *h* é *admissível*, *f (n)* nunca irá superestimar o custo real da melhor solução através de *n*.**
- **Neste caso, a rota escolhida entre Guanambi e Potiraguá é de fato a mais curta uma vez que:**
  - f(Conquista) = 239 + 178 = 417
  - f(Belo Campo)= 220 + 193 = 413
- **Algoritmo:**

função Busca-A* (*problema*)

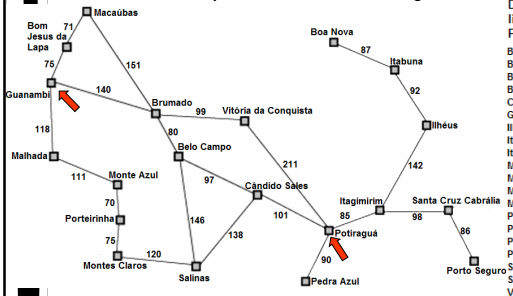
retorna uma solução ou falha

Busca-Melhor-Escolha (*problema*, *g+h*)

Algoritmo A* : exemplo

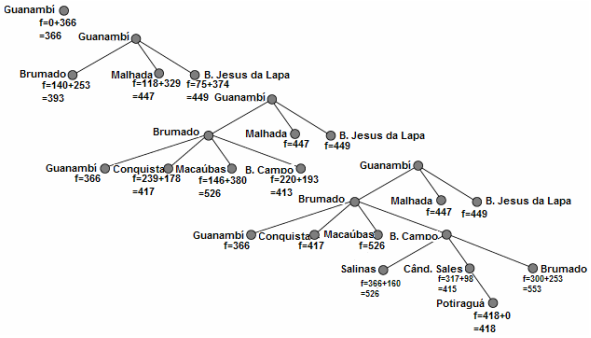
12

Exemplo: Ir de Guanambi a Potiraguá



Distância em linha reta para Potiraguá	
Belo Campo	193
Boa Nova	234
B. Jesus da Lapa	374
Brumado	253
Cândido Sales	98
Guanambi	366
Ilhéus	199
Itabuna	226
Itagimirim	80
Macaúbas	380
Malhada	329
Monte Azul	244
Montes Claros	242
Pedra Azul	77
Porteirinha	241
Potiraguá	0
Porto Seguro	161
Salinas	160
S. Cruz Cabrália	151
V. da Conquista	178

Usando A*



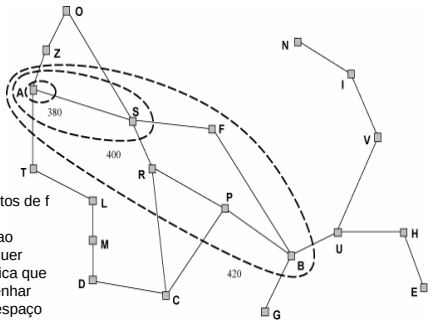
Algoritmo A*: função de avaliação

- **A função heurística h é monotônica se**
  - h (n) ≥ h (sucessor(n))
    - *isso vale para a maioria das funções heurísticas*
- **Transferindo-se esse comportamento para a função de avaliação *f =g+h*, temos que**
  - f (sucessor(n)) ≥ f(n)
    - *uma vez que g é não decrescente*
  - Em outras palavras
    - *o custo de cada nó gerado no mesmo caminho nunca diminui*
- **Se *h* é não monotônica, para se garantir a monotonicidade de *f*, temos:**
  - quando *f(suc(n)) < f (n)*
  - usa-se *f(suc(n)) = max (f(n), g(suc(n)) + h(suc(n)))*

Algoritmo A* : análise do comportamento

- **A estratégia é completa e ótima**
- **Custo de tempo:**
  - exponencial com o comprimento da solução, porém boas funções heurísticas diminuem significativamente esse custo
    - *o fator de expansão fica próximo de 1*
- **Custo memória:** *O* (b^d)
  - guarda todos os nós expandidos na memória
    - *para possibilitar o backtracking*
- **Eficiência ótima**
  - só expande nós com *f(n)* ≤ *f**, onde *f** é o custo do caminho ótimo
    - *f é não decrescente*
  - nenhum outro algoritmo ótimo garante expandir menos nós

A* define Contornos



. O fato de os custos de f serem não-decrescentes ao longo de qualquer caminho significa que podemos desenhar contornos no espaço de estados

Busca com Limite de Memória
Memory Bounded Search

- **IDA* (Iterative Deepening A*) – A* de aprofundamento Interativo**
  - igual ao aprofundamento iterativo, porém seu limite (valor de corte) é dado pela função de avaliação (f(g+h)), e não pela profundidade (d).
  - A cada iteração, o valor de corte é o menor custo de f de qualquer nó que tenha excedido o corte na iteração anterior.
- **BRPM (Busca recursiva pelo melhor)**
  - Semelhante a uma busca recursiva em profundidade, porém em vez de descer indefinidamente pelo caminho atual, ela controla o valor de f do melhor caminho alternativo disponível a partir de qualquer ancestral do nó.
  - Se o nó atual exceder esse limite, a recursão retornará ao caminho alternativo
  - A medida que a recursão se desenrola, o BRPM repõe o valor f de cada nó ao longo do caminho com o melhor valor f de seus filhos

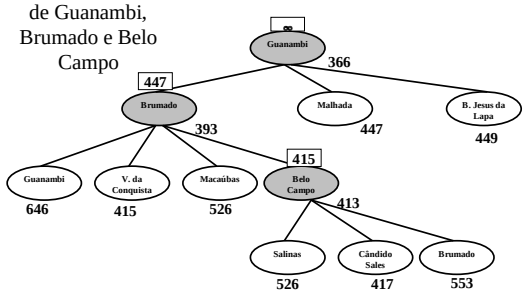
Busca com Limite de Memória
Memory Bounded Search

- **SMA* (Simplified Memory-Bounded A*) – A* limitado pela memória simplificado (LMSA*)**
  - O número de nós guardados em memória é fixado previamente
    - *conforme vai avançando, descarta os piores nós (embora guarde informações a respeito deles) e atualiza os melhores valores dos caminhos*
  - É completa e ótima se a memória alocada for suficiente

19

BRPM

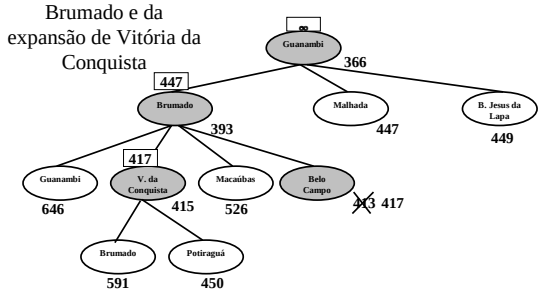
(a) Depois da expansão de Guanambi, Brumado e Belo Campo



20

BRPM

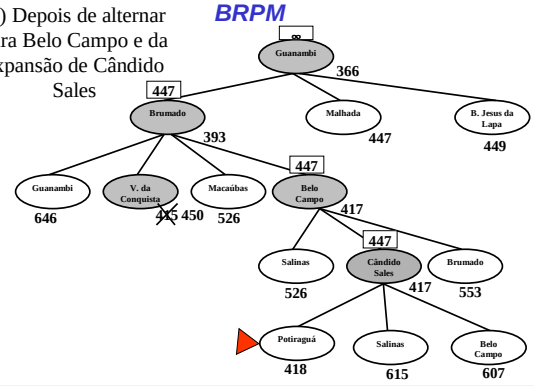
(b) Depois da volta a Brumado e da expansão de Vitória da Conquista



21

BRPM

(c) Depois de alternar para Belo Campo e da expansão de Cândido Sales



22

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

23

(1) Relaxando o problema

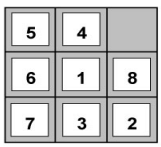
- **Problema Relaxado:**
  - versão simplificada do problema original, onde os operadores são menos restritivos
- **Exemplo: jogo dos 8 números:**
  - operador original: um número pode mover-se de A para B se A é adjacente a B e B está vazio
  - busca exaustiva ≈ 3^{20} estados possíveis
    - *Fator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

4	5	8
	1	6
7	2	3

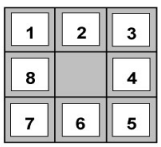
- **Operadores relaxados:**
  1. um número pode mover-se de A para B (*h1*)
  2. um número pode mover-se de A para B se A é adjacente a B (*h2*)

24

Heurísticas para jogo 8 números



Estado inicial



Estado objetivo

- **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)
    - Manhattan Distance d de dois pontos (x,y) e (u,v), d = |x-u| + |y-v|

(2) Usando informação estatística

25

- **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.

(3) Usando atributos/características

26

- **Características do problema podem ser usadas para mensurar o quão se está próximo da solução**
- **ex. quebra-cabeça de 8 peças**
  - Número de blocos mal posicionados (característica $x_1(n)$)
  - Ex.: supondo que tomando 100 configurações do quebra-cabeça ao acaso, obtemos estatísticas sobre os custos reais da solução. Talvez descobríssemos que, quando $x_1(n) = 5$, o custo da solução média é 14. Com esses dados poderíamos usar x_1 para prever h(n). Além de $x_1(n)$ podemos usar várias outras características $x_1(n)$, $x_2(n)$...
    h(n)= $c_1x_1(n) + c_2x_2(n)$
- **Quando não se conhece a importância das características, pode-se aprendê-las ($w_1f_1+w_2f_2+...+w_nf_n$)**

Qualidade da função heurística

27

- **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*
  - $N = 1 + b^* + (b^*)^2 + ... + (b^*)^d$, 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*

Experimento com 100 problemas 8-números

Busca por aprofundamento iterativo

	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.**

Escolhendo Funções Heurísticas

29

- **É sempre melhor usar uma função heurística com valores mais altos, contanto que ela seja *admissível.***
  - ex. *h2* melhor que *h1*
- **h_i <u>domina</u> $h_k \Rightarrow h_i(n) \geq h_k(n)$ $\forall n$ 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

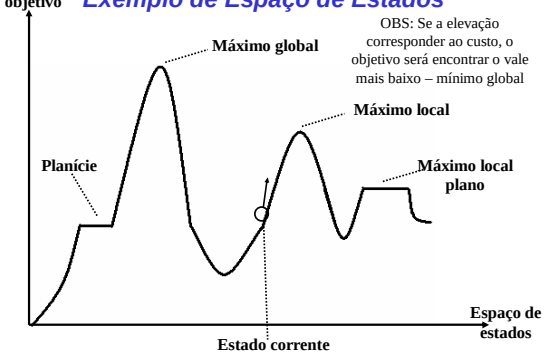
Algoritmos de Melhorias Iterativas (AMI) Iterative Improvement Algorithms

30

- **A idéia é começar com o *estado inicial* (configuração completa, solução aceitável), e melhorá-lo iterativamente.**
  - Imagem da TV
- **Os algoritmos de melhorias interativas são usados em problemas em que o caminho até a solução é irrelevante, o que se importa é a configuração final**
  - Ex.: problema de 8 rainhas
- **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.

31

Exemplo de Espaço de Estados



OBS: Se a elevação corresponder ao custo, o objetivo será encontrar o vale mais baixo – mínimo global

32

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 melhorá-lo no futuro.*

33

Subida da Encosta: algoritmo

- **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)

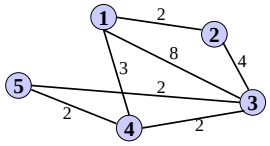
corrente ← Faz-Nó(Estado-Inicial[*problema*])
loop do
  próximo ← **sucessor** de *corrente* **de maior valor** (expande nó *corrente* e seleciona seu melhor filho)
  se Valor[*próximo*] < Valor[*corrente*] (ou >, para minimizar)
    então retorna *corrente* (o algoritmo pára)
  corrente ← *próximo*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

34

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

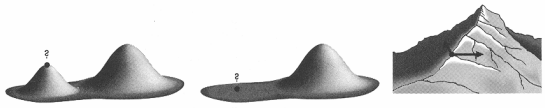
- e1 = {N1, N2, N3, N4, N5}
  f(N1, N2, N3, N4, N5) = 10
- e2 = {N2, N1, N3, N4, N5}
  f(N2, N1, N3, N4, N5) = 14
- e3 = {N2, N1, N4, N3, N5}
  f(N2, N1, N4, N3, N5) = 9!!!



35

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



36

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

37

- **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)

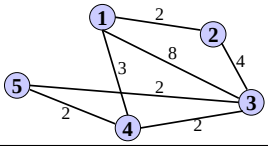
38

- **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

39

- **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 é permutar dois pontos e o caminho resultante não está conectado ({N2, N4, N1, N3, N5}). N2 e N4 não estão conectados!!!



Subida da Encosta

40

- **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

41

- **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

42

- **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

43

- **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

44

- **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

45

- **função** Anelamento-Simulado (*problema,mapeamento*)

  retorna uma solução

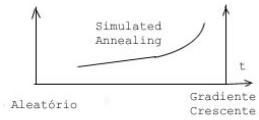
  variáveis locais: *corrente*, *próximo*, *T* (temperatura que controla a probabilidade de passos descendentes)

```
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 ΔE > 0
     então corrente ← próximo
     senão corrente ← próximo com probabilidade = e^(ΔE/T)
```

Anelamento Simulado

46

- **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.



Busca em feixe local

47

- **Mantém o controle de k estados, em vez de somente um.**
- **Começa com k estados gerados aleatoriamente**
- **Em cada passo gera todos os sucessores de todos os k estados**
- **Se qualquer um deles for um objetivo pára.**
- **Caso contrário, ele selecionará os k melhores sucessores da lista completa e repetirá a ação**

Algorítmos genéticos

48

- Em GA, estados sucessores são gerados através da combinação de dois estados antecessores (e não só modificando um único estado).
- Inicia com um conjunto de k estados gerados aleatoriamente, chamado **população**; cada estado (ou **indivíduo**) é representado como uma cadeia sobre um alfabeto finito.
- Indivíduos são avaliados por uma **função de *fitness*** (função de avaliação em AG);
- Indivíduos selecionados geram novos indivíduos por meio de cruzamentos e mutações;
- Repete avaliação/seleção/cruzamento-mutação até que um indivíduo seja avaliado como adequado para solução

AG – exemplo das 8-rainhas

49

- **Representação: cadeia de 8 dígitos representando as 8 colunas do tabuleiro; cada dígito representa a linha onde se encontra a rainha daquela coluna.**

86427531 ➔

- **Fitness: número de pares de rainhas sem ataque mútuo.**
- **• Solução: 12,13,...,18,23,...,28,34,...,38,...,67,68,78 = 28 pares**
- **• Tabuleiro acima: 18 em ataque → valor de fitness = 27**
- **• 24748552 → valor de fitness = 24 (em ataque: 18, 24, 38, 67).**

AG – exemplo das 8-rainhas

50

- **Exemplo de cruzamento:**

- **Exemplo de mutação:**

32748552 ➔ 32748152

Críticas à Busca Heurística

51

- ***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.**

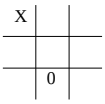
Heurística... por toda IA

52

- **A noção de heurística sempre foi além da busca e de uma formalização via função de um estado**
- **Heurística**
  - escolha, prioridade, estratégia na busca de uma solução razoável onde não há solução ótima ou recurso para determiná-la
  - No dia a dia: heurística para dirigir, namorar, estudar,...
- **Em IA: em todas as áreas como conhecimento de controle**
  - ex. escolha de regras a serem disparadas (SBC)
  - ex. escolha de viés de generalização (aprendizagem)
  - ...

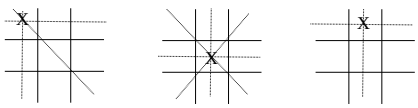
Qual seria uma boa heurística para o jogo da velha?

53



Qual seria uma boa heurística para o jogo da velha?

54



Sugestão: heurística do maior número de vitórias. Alguns estados permitem mais possibilidades de vitória que outros