
UNIVASF
FUNDAÇÃO UNIVERSIDADE FEDERAL
DO VALE DO SÃO FRANCISCO

Busca com informação e exploração

(Capítulo 4 - Russell)

Inteligência Artificial

Professor: Rosalvo Ferreira de Oliveira Neto

Estrutura

- Busca pela melhor escolha
- Busca gulosa pela melhor escolha
- Busca A^*
- Heurísticas
- Algoritmos de busca local
- Hill-climbing search
- Simulated annealing search

Revisão: Árvore de busca

- Uma estratégia de busca é definida pela escolha da ordem de expansão dos nós da árvore

Busca pela melhor escolha

- Ideia: usa uma função de avaliação *f(n)* para cada nó
  - estimativa do nó mais “promissor"

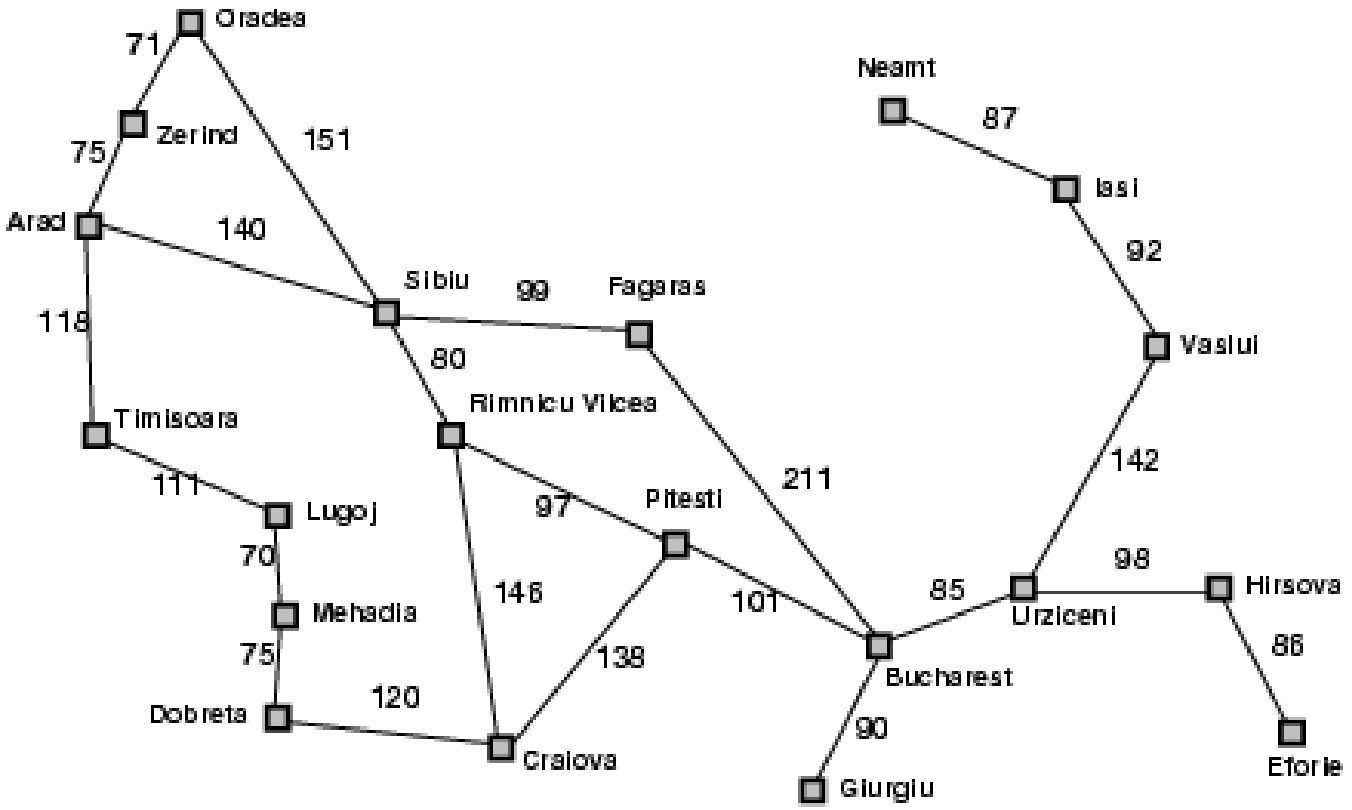
→Expande o nó mais “promissor” ainda não expandido

- Implementação:
  Os nós da borda são ordenados de forma ascendente de valores de *f(n)*.

Casos especiais:

- Busca gulosa
- Busca A*

Romênia com custo do passo em km



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca Gulosa

- Função de avaliação $f(n) = h(n)$ (heurística)
  - Estimativa do custo de *n* até o *objetivo*

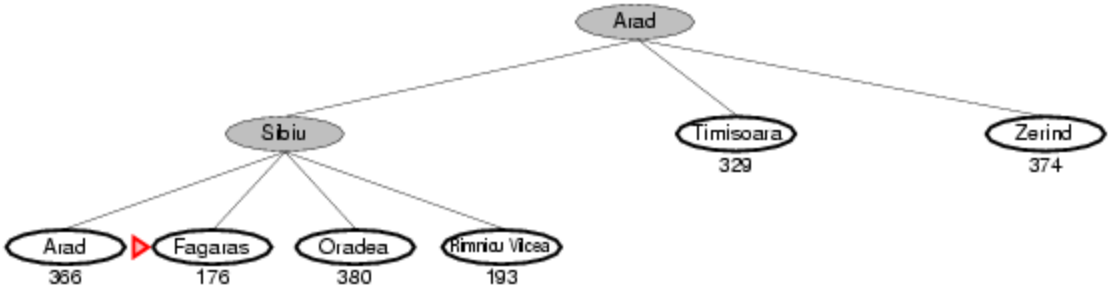
- Exemplo da Romênia. $h_{DLR}(n)$ = distância em linha reta de n até Bucareste

- Busca gulosa expande o nó que parece mais próximo ao objetivo

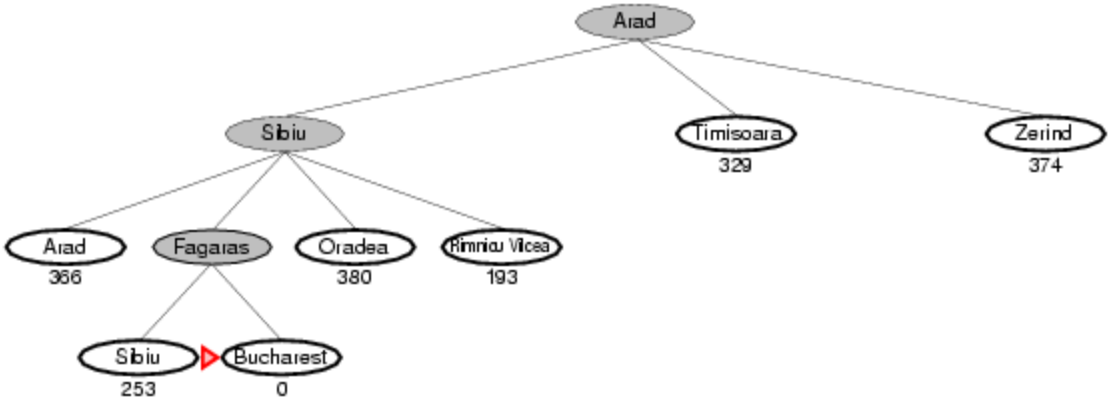
Exemplo de busca gulosa

Exemplo de busca gulosa

Exemplo de busca gulosa



Exemplo de busca gulosa



Propriedades da busca gulosa

- **Completa?**
  - Não – pode ficar presa em loops.
  - Exemplo: Iasi → Neamt → Iasi → Neamt →
- **Ótima?**
  - Não
- **Complexidade de Tempo?**
  - $O(b^m)$, mas uma boa heurística pode reduzir bastante.
- **Espaço?**
  - $O(b^m)$ - Semelhante a busca em profundidade

Busca A*

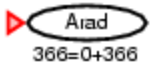
Tipo mais conhecido de busca pela melhor escolha

- Função de avaliação
  - $f(n) = g(n) + h(n)$

Onde

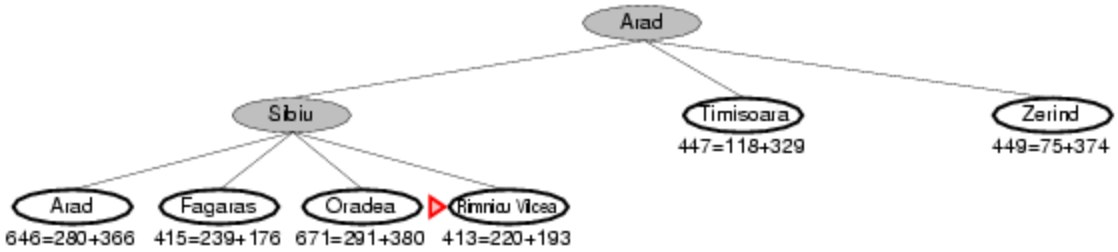
- $g(n)$ = Custo até atingir n
- $h(n)$ = Estimativa de custo do n para o objetivo
- $f(n)$ = Custo total do caminho de n para o objetivo

Exemplo de Busca A*

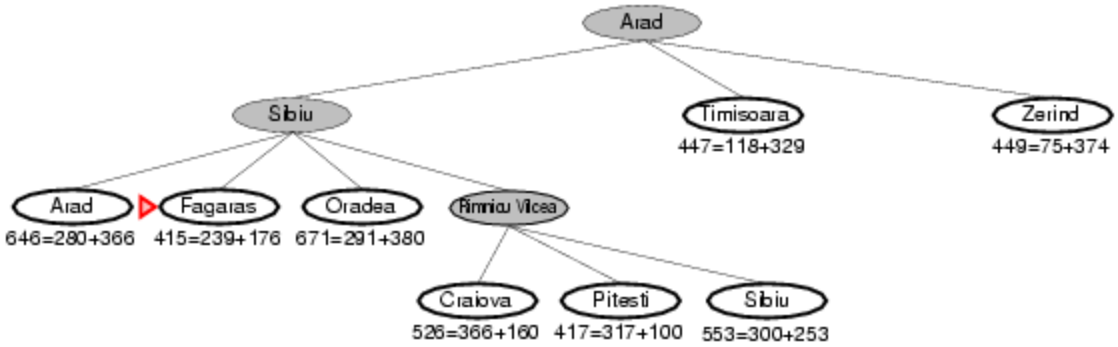


Exemplo de Busca A*

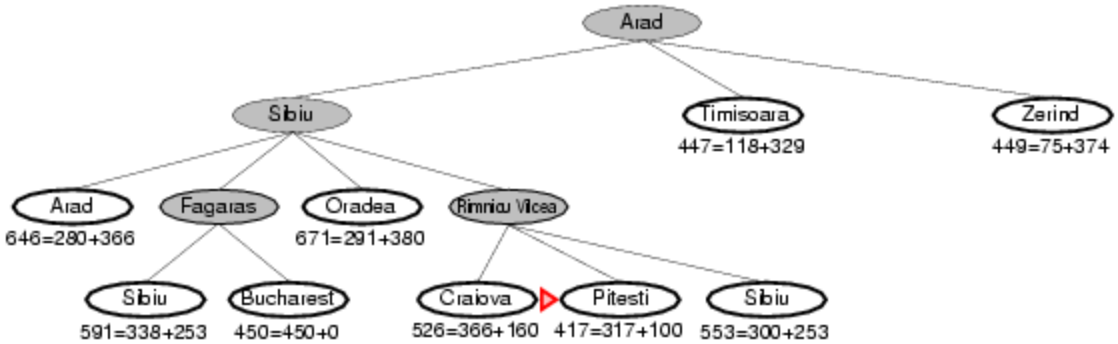
Exemplo de Busca A*



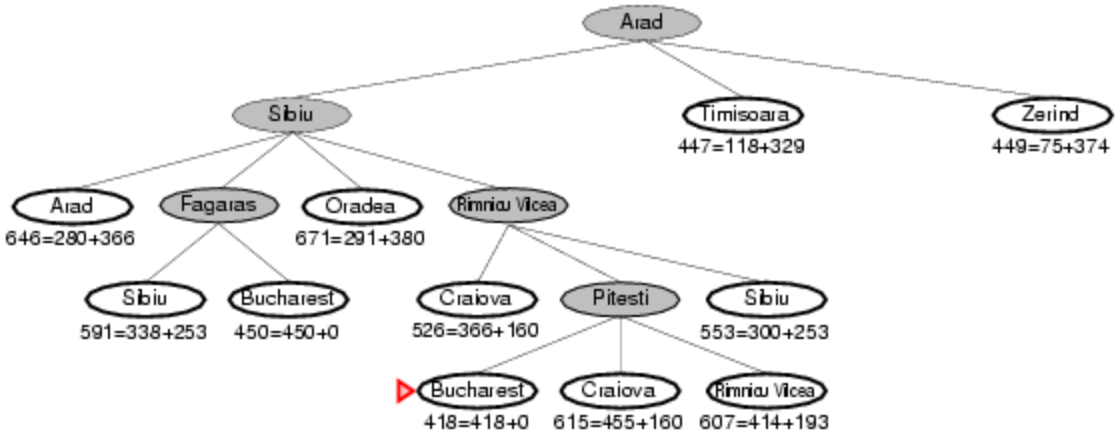
Exemplo de Busca A*



Exemplo de Busca A*



Exemplo de Busca A*



Propriedades da busca A*

- **Completa?**
  - Sim (desde que h(n) tenha propriedade consistente "monotonicidade")
- **Ótima?**
  - Sim
- **Complexidade de Tempo?**
  - *Exponencial*.
- **Espaço?**
  - Mantém todos os nós em memória

Funções heurísticas

- Como escolher uma boa função heurística h?

- h depende de cada problema particular.
- **h deve ser admissível**

O quebra-cabeça de 8 peças foi um dos mais antigos problemas de busca heurísticas.

A busca exaustiva examinaria $3^{22}=3,1 \times 10^{10}$ estados (para um profundidade de 22)

Funções heurísticas admissíveis

- Uma função heurística $h(n)$ é admissível se para todo nó n, $h(n) \leq h^*(n)$, onde $h^*(n)$ é o custo verdadeiro para alcançar o estado objetivo a partir de n.

- Uma heurística admissível nunca superestima o custo para alcançar o objetivo. Ela é ótima.

- Exemplo: $h_{DLR}(n)$ (Nunca superestima o custo real de uma rodovia)

- Teorema: Se $h(n)$ é admissível, a busca em árvore utilizando A* é ótima.

Problema Relaxado

- Versão simplificada do problema original, onde os operadores (ações) são menos restritivos.

- O custo de uma solução ótima para o problema relaxado é uma solução admissível para o problema original.

Problema Relaxado

Exemplo do quebra-cabeça de 8 peças:

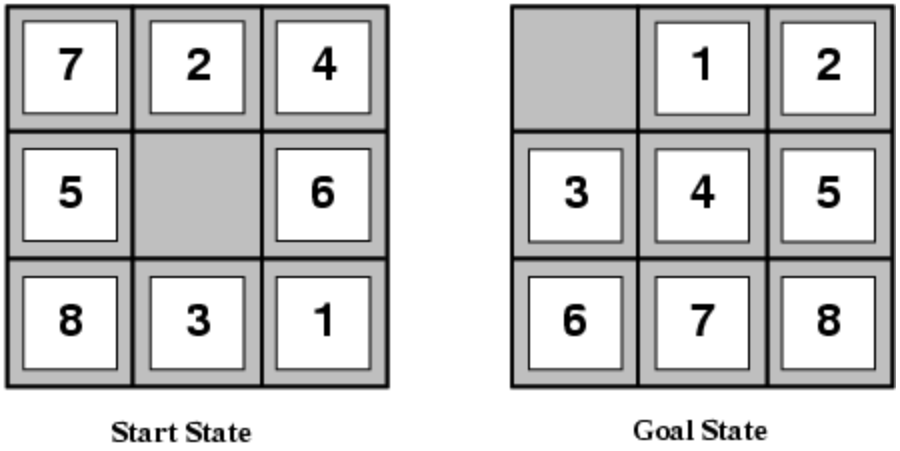
- **Ação Original**
  - Um número pode mover-se de A para B se A é adjacente a B e B está vazio
- **Versão Relaxada**
  - Um número pode mover-se de A para B se A é adjacente a B (h2)
  - Um número pode mover-se de A para B (h1)

Problema Relaxado – Heurísticas derivadas

Exemplo do quebra-cabeça de 8 peças:

- $h_1(n)$ = Número de blocos mal posicionados
- $h_2(n)$ = Distância de Manhattan

(i.e. Pontuação adequada se movêssemos um bloco por vez até o quadrado desejado)



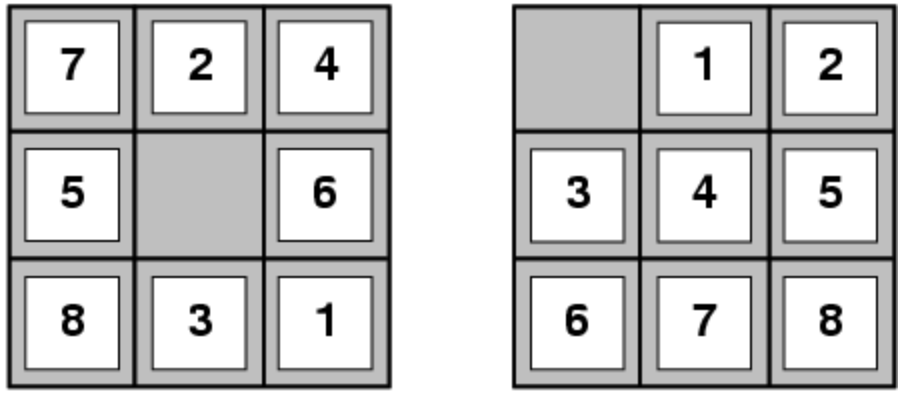
- $h_1(S) = ?$
- $h_2(S) = ?$

Problema Relaxado – Heurísticas derivadas

Exemplo do quebra-cabeça de 8 peças:

- $h_1(n)$ = Número de blocos mal posicionados
- $h_2(n)$ = Distância de Manhattan

(i.e. Pontuação adequada se movêssemos um bloco por vez até o quadrado desejado)



Start State Goal State

- $h_1(S) = 8$
- $h_2(S) = 3 + 1 + 2 + 2 + 3 + 2 + 2 + 3 = 18$

Escolhendo Funções Heurísticas

- Se $h_2(n) \geq h_1(n)$ para todo *nó* (ambas admissível)
- Então h_2 domina h_1
- h_2 é melhor para busca

- É sempre melhor usar uma função heurística com valores mais altos, contanto que ela seja admissível.

Algoritmos de busca local

- Em muitos problemas de otimização, o caminho para o estado objetivo é irrelevante; o próprio estado objetivo é a solução

- Espaço de estado = Conjunto de configurações "completa"
- Encontrar conjunto de configurações que satisfaça a restrição. Exemplo: n-queens

- Em tais casos, nós podemos utilizar algoritmos de busca local
- Armazena um único estado corrente, tenta melhorá-lo

Hill-climbing search

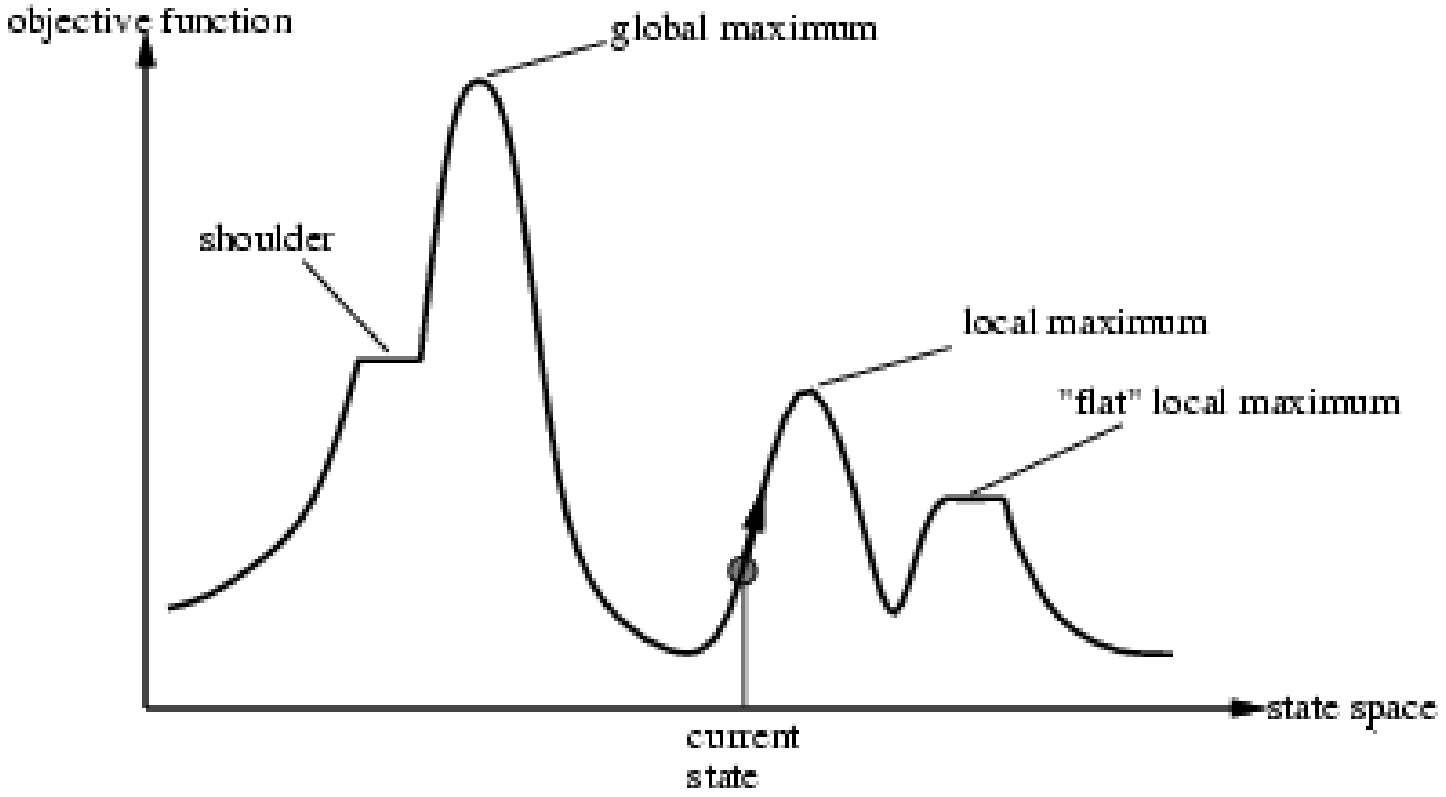
- função **Hill-Climbing (*problema*)** retorna **uma solução**

  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

Hill-climbing search

- Problema: dependendo do estado inicial, pode ficar preso em um máximo local.



Simulated annealing search

- Este algoritmo é semelhante à Subida da Encosta, porém oferece meios para 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

Simulated annealing search

- Analogia com cozimento de vidros ou metais:
  - processo de resfriar um líquido gradualmente até ele se solidificar
- O algoritmo utiliza um mapeamento de resfriamento de instantes de tempo (t) em temperaturas (T).

Simulated annealing search

- função **Têmpera-Simulada** (***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 *ΔE* > 0

  então *corrente* ← *próximo*

  senão *corrente* ← *próximo* com probabilidade = $e^{-\Delta E/T}$

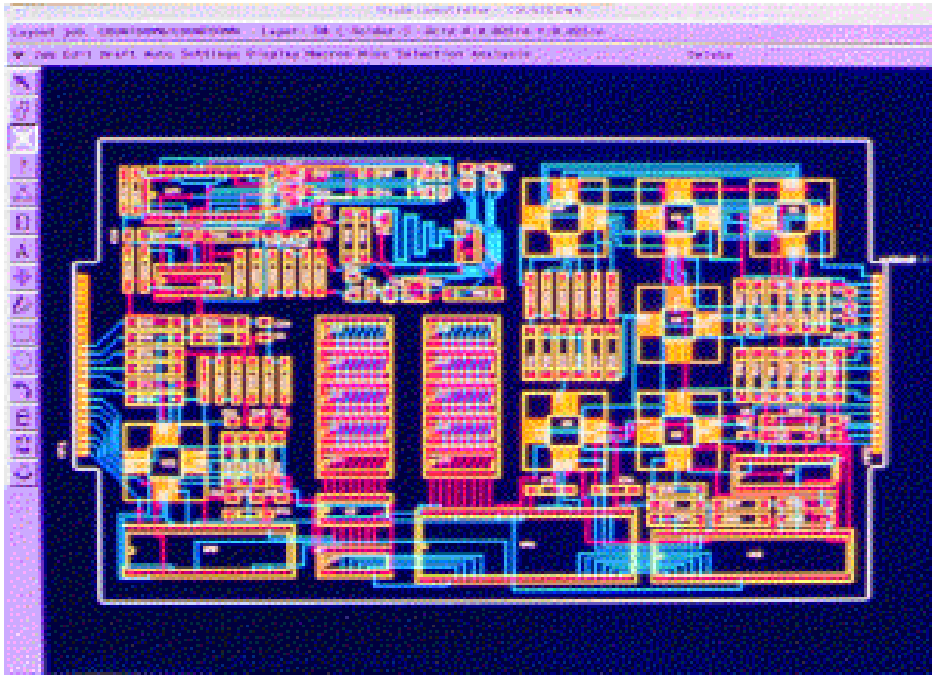
Simulated annealing search

- Para valores de T próximos de zero
  - a expressão $\Delta E/T$ cresce
  - a expressão $e^{-\Delta E/T}$ tende a zero
  - a probabilidade de aceitar um valor de próximo menor que corrente tende a zero
  - o algoritmo tende a aceitar apenas valores de próximo maiores que corrente
- Conclusão
  - com o passar do tempo (diminuição da temperatura), este algoritmo passa a funcionar como Subida da Encosta

Simulated annealing search

Bastante utilizado em:

- Layout VLSI;
- Agendamento de companhia aérea.




UNIVASF
FUNDAÇÃO UNIVERSIDADE FEDERAL
DO VALE DO SÃO FRANCISCO