

Aula 4 – Algoritmos de Busca Local

Algoritmos de busca local

- ▲ Em muitos problemas de otimização o caminho ao objetivo é irrelevante.
 - ▢ N-rainhas: N rainha \rightarrow solução: $\{(1,3); (2,4); \dots (N,1)\}$
 - ▢ Caixeiro viajante: N ponto \rightarrow solução = $\{1,2,3,\dots,N,1\}$
 - ▢ Problema da Mochila

- ▲ Para esses casos podemos usar algoritmos de busca local.
 - ▢ Objetivo: encontrar a configuração ótima.
 - ▢ Solução: manter um simples estado e tentar melhorar.

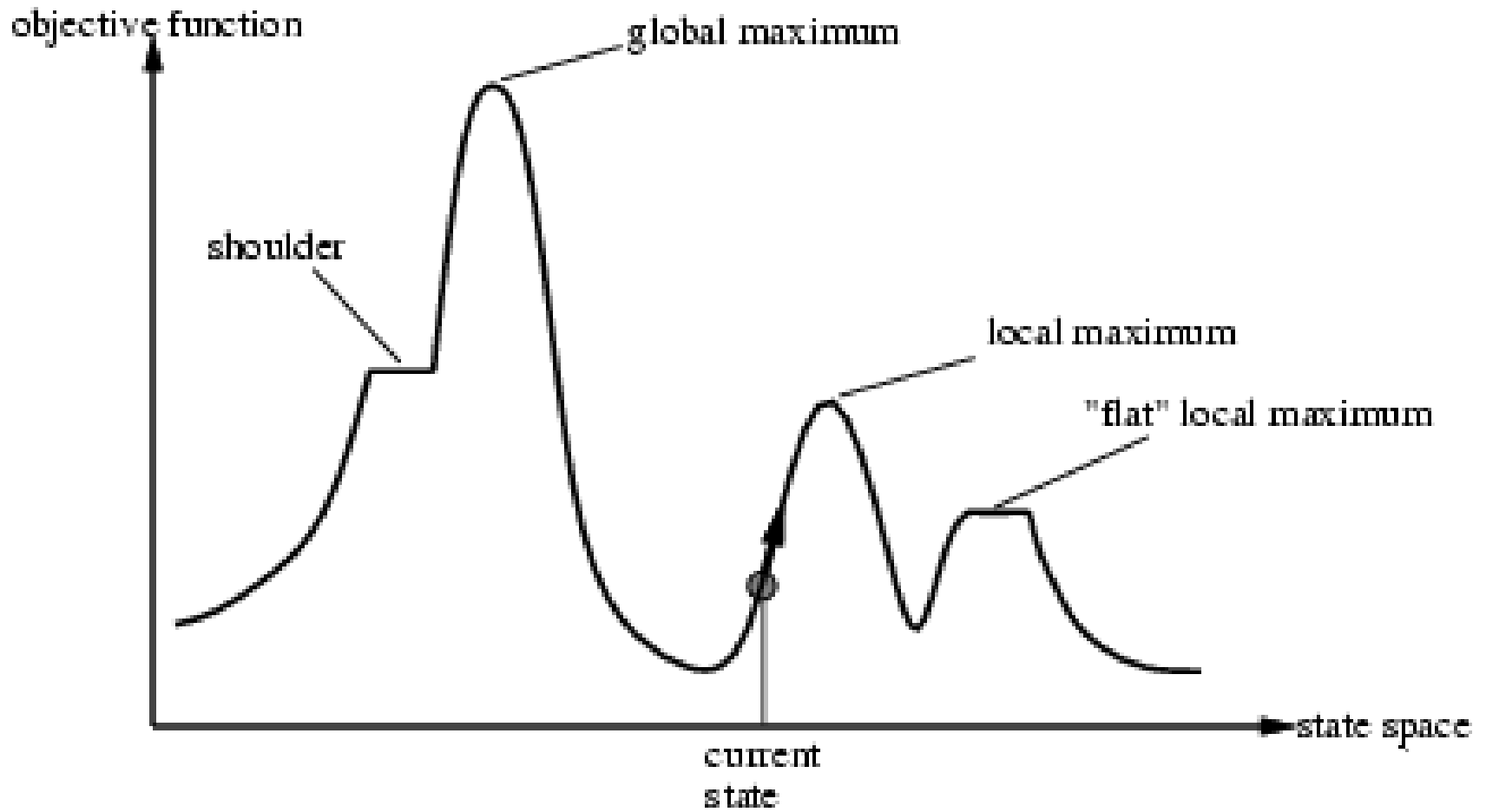
- ▲ Vantagens:
 - ▢ Usam pouca memória.
 - ▢ Podem encontrar soluções **razoáveis** em grandes ou infinitos espaços de estados.
 - ▢ São úteis para resolver problemas de otimização puros.

- ▲ Estados estão representados sobre uma superfície:
 - ▢ A altura de qualquer ponto corresponde à função de avaliação do estado naquele ponto.

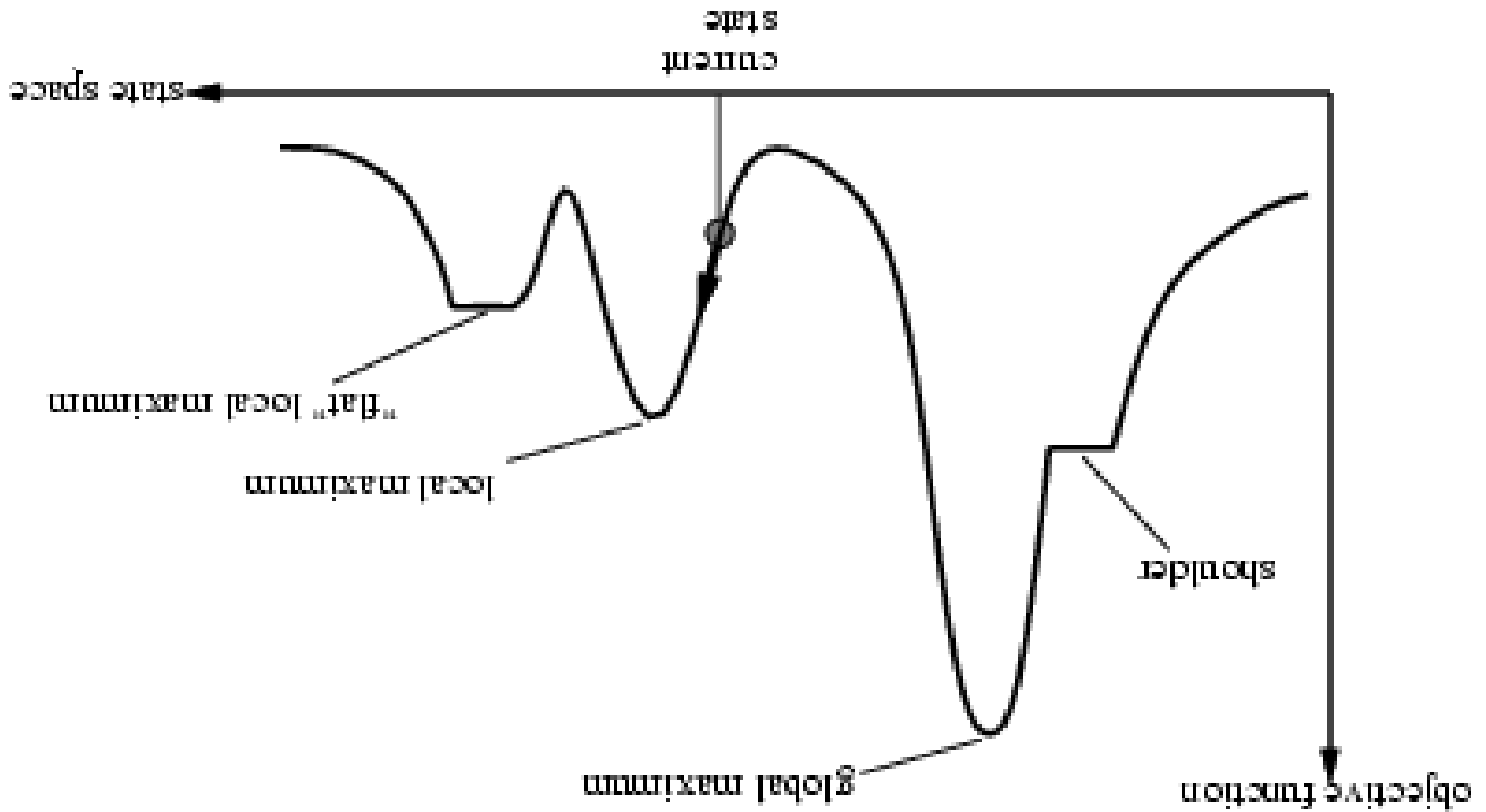
Busca Local x Busca Baseada em Árvore

	Busca Baseada em Árvore	Busca Local
Estado Inicial	Nó = posição inicial	Solução Inicial
Estado Final	Nó = posição final	Solução Final
Função Sucessora	Gera novos estados / nós / posições	Gera uma nova solução

Topologia do espaço de estados



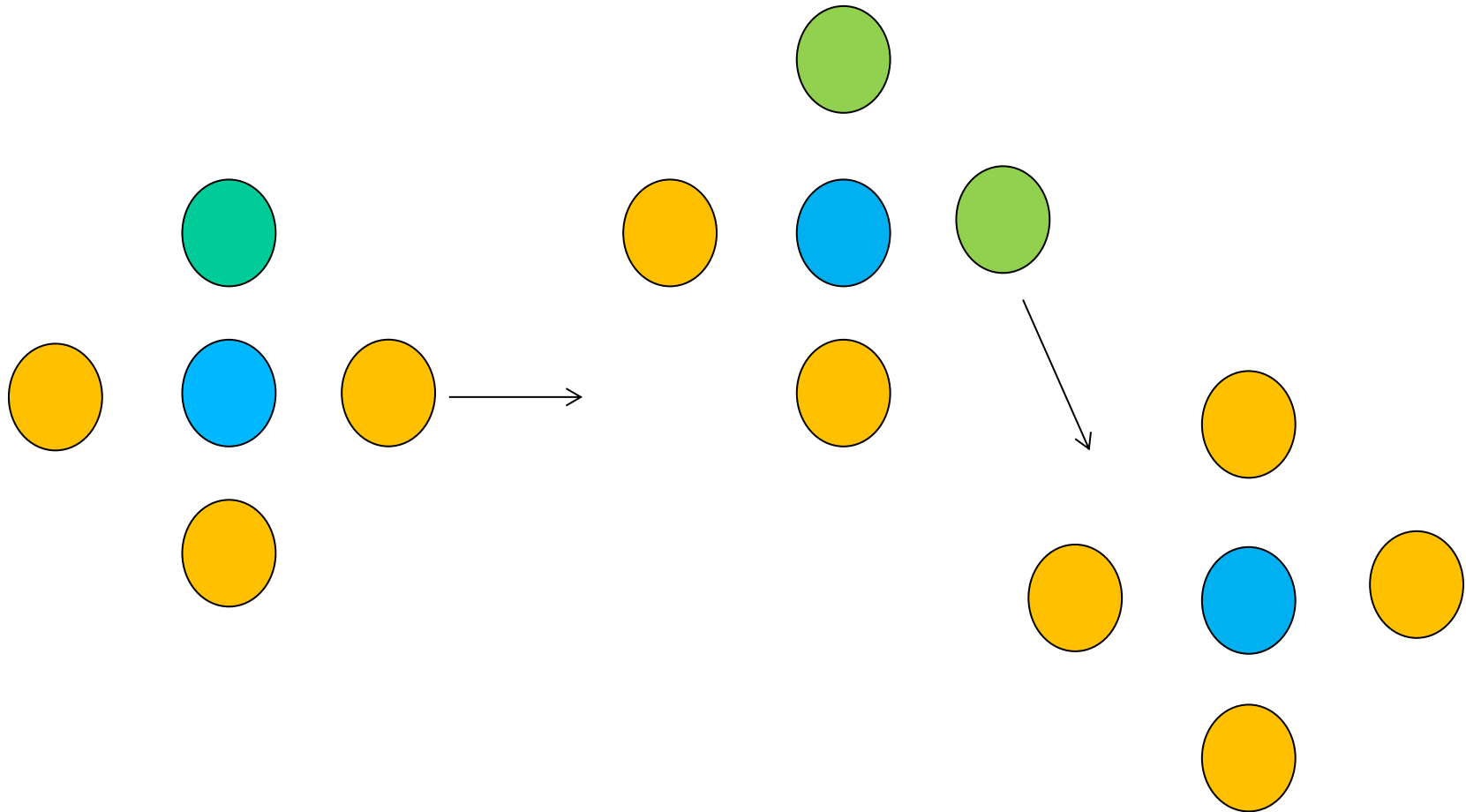
Topologia do espaço de estados



Subida de Encosta (Hill Climbing)









- ⤴ “Como alcançar o cume do Monte Evereste em meio a um nevoeiro denso durante uma crise de amnésia?”.
- ⤴ É um laço simples que se move em direção de um valor crescente
- ⤴ Não examina valores de estados além dos vizinhos imediatos do estado atual.
- ⤴ Geralmente faz uma escolha aleatória entre o conjunto de melhores sucessores (caso exista mais de um).
- ⤴ Problemas com o método:
 - Máximos locais.
 - Picos.
 - Platôs.

Subida de Encosta (Hill Climbing)



Subida de encosta

- h = número de pares de rainhas que atacam umas às outras.
- Exemplo: $h = 17$.

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

Subida de encosta

Função Subida-de-Encosta (N) retorna estado que é a solução

variáveis locais atual: solução

 próximo: solução

atual ← **SOLUÇÃO_INICIAL(N)**

va = AVALIA(atual)

Repita

novo ← **SUCESORES(ATUAL)***

vn = AVALIA(novo);

 se $(vn \geq va)$ então retornar atual

 senão atual ← novo;

 va ← vn;

Fim-Repita

* retorna, dentre os sucessores gerados, o melhor.

EXEMPLO

- Supondo o PCV, com $N=6$. Dados:

- Rotina Solução inicial: $\{1,2,3,4,5,6\}$
- Matriz de Adjacências:

	C1	C2	C3	C4	C5	C6
C1	0	7	3	6	2	2
C2	1	0	4	1	3	5
C3	2	2	0	4	6	8
C4	3	5	6	0	2	8
C5	3	4	3	6	0	4
C6	3	5	3	2	1	0

- Função sucessores: selecionar uma cidade aleatoriamente e trocar de posição com as demais; cada troca representa um novo sucessor.
- Função avalia: soma do custo do caminho considerando a Matriz de Adjacências.

EXEMPLO

- atual: {1,2,3,4,5,6}
- $va = 7 + 4 + 4 + 2 + 4 + 3 = 24$
- Iteração 1:
 - Sucessores (cidade = 4):
 - **Sucessor: [4, 2, 3, 1, 5, 6] Valor: 19 (melhor) = 5+4+2 +2+4+2**
 - **Sucessor: [1, 4, 3, 2, 5, 6] Valor: 24 = 6+6+2+3+4+3**
 - **Sucessor: [1, 2, 4, 3, 5, 6] Valor: 27**
 - **Sucessor: [1, 2, 3, 5, 4, 6] Valor: 34**
 - **Sucessor: [1, 2, 3, 6, 5, 4] Valor: 29**
 - novo = {4, 2, 3, 1, 5, 6}
 - $vn = 19$
 - vn é maior ou igual a que va
 - **atual = {4,2,3,1,5,6}**
 - **va = 19**

EXEMPLO

- atual: {4,2,3,1,5,6}
- va = 19
- Iteração 2:
 - Sucessores (cidade = 2):
 - **Sucessor 1 : [2, 4, 3, 1, 5, 6] Valor: 20**
 - **Sucessor 2 : [4, 3, 2, 1, 5, 6] Valor: 17**
 - **Sucessor 3 : [4, 1, 3, 2, 5, 6] Valor: 17**
 - **Sucessor 4 : [4, 5, 3, 1, 2, 6] Valor: 21**
 - **Sucessor 5 : [4, 6, 3, 1, 5, 2] Valor: 20**
 - novo = {4, 3, 2, 1, 5, 6}
 - vn = 17
 - vn é maior ou igual a que va
 - **atual = {4,3,2,1,5,6}**
 - **va = 17**

EXEMPLO

- atual: {4,3,2,1,5,6}
- va = 17
- Iteração :
 - Sucessores (cidade = 5):
 - **Sucessor 1 : [5, 3, 2, 1, 4, 6] Valor: 21**
 - **Sucessor 2 : [4, 5, 2, 1, 3, 6] Valor: 20**
 - **Sucessor 3 : [4, 3, 5, 1, 2, 6] Valor: 29**
 - **Sucessor 4 : [4, 3, 2, 5, 1, 6] Valor: 18**
 - **Sucessor 5 : [4, 3, 2, 1, 6, 5] Valor: 18**
- novo = {4, 3, 2, 5, 1, 6}
 - vn = 18
 - vn é maior ou igual a que va
 - **break - sair do loop**
- Retornar como resposta atual: atual: {4,3,2,1,5,6} com valor = 17.

Subida de encosta

Função Subida-de-Encosta (p) retorna estado que é a solução

variáveis locais atual: um nó

 próximo: um nó

TMAX \leftarrow 3 (definido pelo usuário)

t \leftarrow 0

atual \leftarrow **SOLUÇÃO_INICIAL()**

va = AVALIA(atual);

Repita

novo \leftarrow **MELHOR sucessor de atual**

 vn = AVALIA(novo);

 se (vn \geq va) então

 se (t \leq TMAX) t++

 senão retornar atual

 senão atual \leftarrow novo; va \leftarrow vn ; t \leftarrow 0;

Fim-Repita

EXEMPLO

- Supondo o PCV, com $N=6$. Dados:

- Rotina Solução inicial: $\{1,2,3,4,5,6\}$
- Matriz de Adjacências:

	C1	C2	C3	C4	C5	C6
C1	0	7	3	6	2	2
C2	1	0	4	1	3	5
C3	2	2	0	4	6	8
C4	3	5	6	0	2	8
C5	3	4	3	6	0	4
C6	3	5	3	2	1	0

- Função sucessores: selecionar uma cidade aleatoriamente e trocar de posição com as demais; cada troca representa um novo sucessor.
- Função avalia: soma do custo do caminho considerando a Matriz de Adjacências.

EXEMPLO

- $TMAX = 3$
- $t = 0$
- atual: {1,2,3,4,5,6}
- $va = 24$
- Iteração 1:
 - Sucessores (cidade = 4):
 - **Sucessor: [4, 2, 3, 1, 5, 6] Valor: 19**
 - **Sucessor: [1, 4, 3, 2, 5, 6] Valor: 24**
 - **Sucessor: [1, 2, 4, 3, 5, 6] Valor: 27**
 - **Sucessor: [1, 2, 3, 5, 4, 6] Valor: 34**
 - **Sucessor: [1, 2, 3, 6, 5, 4] Valor: 29**
 - novo = {4, 2, 3, 1, 5, 6}
 - $vn = 19$
 - vn é maior ou igual a que va
 - atual = {4,2,3,1,5,6}
 - $va = 19$

EXEMPLO

- $t = 0$
- atual: {4,2,**3**,1,5,6}
- $va = 19$
- Iteração 2:
 - Sucessores (cidade = 3):
 - **Sucessor 1 : [3, 2, 4, 1, 5, 6] Valor: 15**
 - **Sucessor 2 : [4, 3, 2, 1, 5, 6] Valor: 17**
 - **Sucessor 3 : [4, 2, 1, 3, 5, 6] Valor: 21**
 - **Sucessor 4 : [4, 2, 5, 1, 3, 6] Valor: 24**
 - **Sucessor 5 : [4, 2, 6, 1, 5, 3] Valor: 22**
 - novo = [3,2,4,1,5,6]
 - $vn = 15$
 - vn é maior ou igual a que va
 - atual = {3,2,4,1,5,6}
 - $va = 15$

EXEMPLO

- $t = 0$
- atual: {3,2,4,1,5,6}
- $va = 15$
- Iteração 3:
 - Sucessores (cidade = 3):
 - **Sucessor 1 : [2, 3, 4, 1, 5, 6] Valor: 22**
 - **Sucessor 2 : [4, 2, 3, 1, 5, 6] Valor: 19**
 - **Sucessor 3 : [1, 2, 4, 3, 5, 6] Valor: 27**
 - **Sucessor 4 : [5, 2, 4, 1, 3, 6] Valor: 20**
 - **Sucessor 5 : [6, 2, 4, 1, 5, 3] Valor: 22**
 - novo = [4,2,3,1,5,6]
 - $vn = 19$
 - vn é maior ou igual a que va
 - t é menor ou igual a TMAX
 - $t=1$

EXEMPLO

- $t = 1$
- atual: {3,2,4,1,5,6}
- $va = 15$
- Iteração 4:
 - Sucessores (cidade = 6):
 - **Sucessor 1 : [6, 2, 4, 1, 5, 3] Valor: 22**
 - **Sucessor 2 : [3, 6, 4, 1, 5, 2] Valor: 23**
 - **Sucessor 3 : [3, 2, 6, 1, 5, 4] Valor: 24**
 - **Sucessor 4 : [3, 2, 4, 6, 5, 1] Valor: 18**
 - **Sucessor 5 : [3, 2, 4, 1, 6, 5] Valor: 12**
 - novo = [3,2,4,1,6,5]
 - $vn = 12$
 - vn é maior ou igual a que va
 - atual= [3,2,4,1,6,5]
 - $va=12$
 - $t=0$

EXEMPLO

- $t = 0$
- atual: [3,2,4,1,6,5]
- $va = 12$
- Iteração 5:
 - Sucessores (cidade = 2):
 - **Sucessor 1 : [2, 3, 4, 1, 6, 5] Valor: 18**
 - **Sucessor 2 : [3, 4, 2, 1, 6, 5] Valor: 16**
 - **Sucessor 3 : [3, 1, 4, 2, 6, 5] Valor: 22**
 - **Sucessor 4 : [3, 6, 4, 1, 2, 5] Valor: 26**
 - **Sucessor 5 : [3, 5, 4, 1, 6, 2] Valor: 26**
 - novo = [3,4,2,1,6,5]
 - $vn = 16$
 - vn é maior ou igual a que va
 - t é menor ou igual a TMAX
 - $t=1$

EXEMPLO

- $t = 1$
- atual: [3,2,4,1,6,5]
- $va = 12$
- Iteração 6:
 - Sucessores (cidade = 1):
 - **Sucessor 1 : [1, 2, 4, 3, 6, 5] Valor: 26**
 - **Sucessor 2 : [3, 1, 4, 2, 6, 5] Valor: 22**
 - **Sucessor 3 : [3, 2, 1, 4, 6, 5] Valor: 21**
 - **Sucessor 4 : [3, 2, 4, 6, 1, 5] Valor: 19**
 - **Sucessor 5 : [3, 2, 4, 5, 6, 1] Valor: 15**
 - novo = [3,2,4,5,6,1]
 - $vn = 15$
 - vn é maior ou igual a que va
 - t é menor ou igual a TMAX
 - $t=2$

EXEMPLO

- $t = 2$
- atual: [3,2,4,1,6,5]
- $va = 12$
- Iteração 7:
 - Sucessores (cidade = 5):
 - **Sucessor 1 : [5, 2, 4, 1, 6, 3] Valor: 19**
 - **Sucessor 2 : [3, 5, 4, 1, 6, 2] Valor: 26**
 - **Sucessor 3 : [3, 2, 5, 1, 6, 4] Valor: 18**
 - **Sucessor 4 : [3, 2, 4, 5, 6, 1] Valor: 15**
 - **Sucessor 5 : [3, 2, 4, 1, 5, 6] Valor: 15**
 - novo = [3,2,4,5,6,1]
 - $vn = 15$
 - vn é maior ou igual a va
 - t é menor ou igual a TMAX
 - $t=3$

EXEMPLO

- $t = 3$
- atual: [3,2,4,1,6,5]
- $va = 12$
- Iteração 7:
 - Sucessores (cidade = 1):
 - **Sucessor 1 : [1, 2, 4, 3, 6, 5] Valor: 26**
 - **Sucessor 2 : [3, 1, 4, 2, 6, 5] Valor: 22**
 - **Sucessor 3 : [3, 2, 1, 4, 6, 5] Valor: 21**
 - **Sucessor 4 : [3, 2, 4, 6, 1, 5] Valor: 19**
 - **Sucessor 5 : [3, 2, 4, 5, 6, 1] Valor: 15**
 - novo = [3,2,4,5,6,1]
 - $vn = 15$
 - vn é maior ou igual a que va
 - t é menor ou igual a TMAX
 - $t=4$

EXEMPLO

- $t = 4$
- atual: [3,2,4,1,6,5]
- $va = 12$
- Iteração 7:
 - Sucessores (cidade = 1):
 - **Sucessor 1 : [1, 2, 4, 3, 6, 5] Valor: 26**
 - **Sucessor 2 : [3, 1, 4, 2, 6, 5] Valor: 22**
 - **Sucessor 3 : [3, 2, 1, 4, 6, 5] Valor: 21**
 - **Sucessor 4 : [3, 2, 4, 6, 1, 5] Valor: 19**
 - **Sucessor 5 : [3, 2, 4, 5, 6, 1] Valor: 15**
 - novo = [3,2,4,5,6,1]
 - $vn = 15$
 - vn é maior ou igual a que va
 - t é menor ou igual a TMAX

Têmpera simulada (simulated annealing)

- ▲ Procura combinar subida de encosta com um percurso aleatório que resulte de algum modo em eficiência e completeza:
 - ▢ Têmpera: processo de aquecimento de metais e de vidros a altas temperaturas seguido de resfriamento lento e gradual, permitindo que o material seja misturado em um estado cristalino de baixa energia.
- ▲ Idéia: “Imagine a tarefa de colocar uma bola de pingue-pongue na fenda mais profunda de uma superfície acidentada”.
 - ▢ Agitar com força e depois reduzir gradualmente a intensidade da agitação.
- ▲ Fuga de máximos locais permitindo-se alguns movimentos “ruins” mas gradualmente diminuindo-se o tamanho e freqüência destes.
- ▲ Necessário a definição de alguns parâmetros como: temperatura inicial, temperatura final..
- ▲ Escolha de movimentos aleatórios em vez de melhores. Se ele melhorar a situação ele é executado, senão é feito um movimento com uma probabilidade que é <1 .

Têmpera simulada

Função TÊMPERA-SIMULADA()

atual \leftarrow SOLUÇÃO_INICIAL()

va \leftarrow AVALIA(atual)

temp = VALOR_MAXIMO_TEMP

ft_red = VALOR_INICIAL_FT

ENQUANTO(temp > VALOR_MINIMO_TEMP)

nov \leftarrow um sucessor de ATUAL selecionado ao acaso

vn \leftarrow AVALIA(novo)

$\Delta E \leftarrow vn - va$

SE($\Delta E < 0$) ENTÃO

 atual \leftarrow novo; va \leftarrow vn;

SENÃO

 ale \leftarrow random(); aux \leftarrow exp(- ΔE /temp);

SE (ale \leq aux) ENTÃO

 atual \leftarrow novo; va \leftarrow vn;

 temp = temp * ft_red

FIM-ENQUANTO

Retornar atual

Busca em feixe local

▲ Descrição do método:

- Mantém o controle de k estados, inicialmente gerados aleatoriamente.
- Em cada passo são gerados todos os sucessores de cada estado.
- Se qualquer um deles for um objetivo, o algoritmo irá parar. Caso contrário, seleciona os k melhores estados da lista total de sucessores gerados.

▲ Problema:

- Pode ficar concentrada em uma pequena região do espaço de estados.

▲ Solução atenuante: busca em feixe estocástica.

- Escolhe k sucessores ao acaso, com a probabilidade de escolher um determinado sucessor que seja a função crescente de seu valor.
- Apresenta alguma semelhança com o processo de seleção natural.