Heurísticas e Metaheuristicas

Simulated Annealing

Prof. Guilherme de Castro Pena
guilherme.pena@ufsj.edu.br
Sala: DCOMP 3.11

Departamento de Ciência da Computação
Universidade Federal de São João del-Rei

Material adaptado do Prof. André (UFV) e Prof. Marcone (UFOP)

Agenda

Heurísticas Clássicas

Visão geral:

- Para relembrar, quando citamos **heurísticas clássicas**, podemos classificá-las de duas formas:
  - **Heurísticas Construtivas:** Consistem em construir uma solução passo a passo, elemento por elemento.
  - **Heurísticas de Refinamento:** Consistem em efetuar modificações na solução construída, de forma a tentar melhorá-la.

Heurísticas Construtivas

Visão geral:

- Das heurísticas construtivas, que constroem uma solução passo a passo, basicamente existem:
  - Construção aleatória.
  - Construção gulosa.
  - Construção parcialmente gulosa, que é uma mistura de ambos.

Heurísticas Construtivas - Construção Aleatória

Funcionamento

- Constrói uma solução, elemento por elemento.
- A cada passo é adicionado um único elemento candidato na solução parcial.
- O candidato a ser adicionado é escolhido **aleatoriamente** dentre os elementos candidatos.
- O método se encerra quando todos os elementos candidatos foram analisados.

Heurísticas Construtivas - Construção Aleatória

Pseudo-código (Aleatório)

Algorithm 1: Template do método de construção aleatória

```
1  s ← ∅
2  Inicializa o conjunto C de elementos candidatos
3  while C ≠ ∅ do
4  |    Aleatoriamente escolha t ∈ C
5  |    if t pode ser inserido em s then
6  |    |    s ← s ∪ {t}
7  |    |    C ← C \ {t}
8  |    end
9  end
10 return s (solução completa)
```

- A ideia principal para o método é que a cada escolha, um candidato aleatório entra na solução e é removido do conjunto C. Podemos considerar que a entrada do problema permite isso para qualquer escolha.

Heurísticas Construtivas - Construção Gulosa

Funcionamento

- Constrói uma solução, elemento por elemento
- A cada passo é adicionado um único elemento candidato na solução parcial
- O candidato escolhido é o "melhor" segundo um certo critério
- O método se encerra quando todos os elementos candidatos foram analisados

Heurísticas Construtivas - Construção Gulosa

Pseudo-código (Guloso)

Algorithm 2: Template do método de construção gulosa

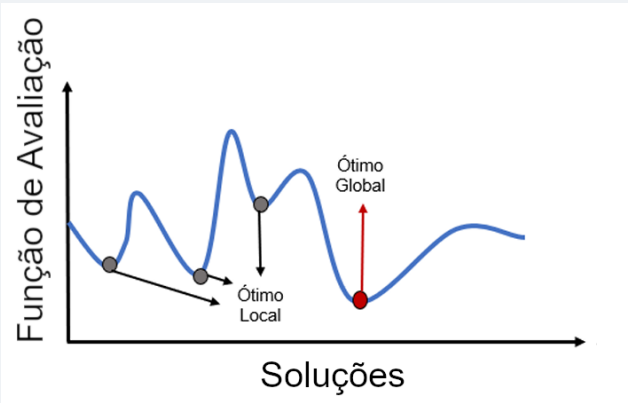
```
1  s ← ∅
2  Inicializa o conjunto C de elementos candidatos
3  while C ≠ ∅ do
4  |   g_best = melhor {g(t)|t ∈ C}
5  |   Seja t_best o valor de t ∈ C associado a g_best
6  |   if t_best pode ser inserido em s then
7  |   |   s ← s ∪ {t_best}
8  |   |   C ← C \ {t_best}
9  |   end
10 end
11 return s (solução completa)
```

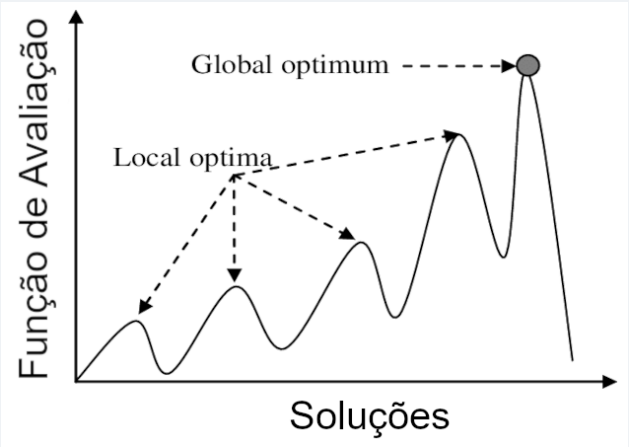
- Nesse caso, a função $g()$ entrega o melhor candidato segundo o critério guloso utilizado. Nesse caso,
- **Melhor = max** se o problema for de maximização.
- **Melhor = min** se o problema for de minimização.

Heurísticas de Refinamento

Visão geral:

- As heurísticas de refinamento são as heurísticas de **busca local**.
- Elas atuam sobre soluções completas.
- E como vimos, elas são baseadas no conceito de vizinhança.
- Dentre os métodos, destacam-se:
  - Método da Descida (*Descent Method*)
  - Método da Subida (*Uphill Method* ou *Hill Climbing*)
  - **Importante:** Sem perda de generalidade, ambos são similares, altera-se o fato de ser um problema de min ou de max.

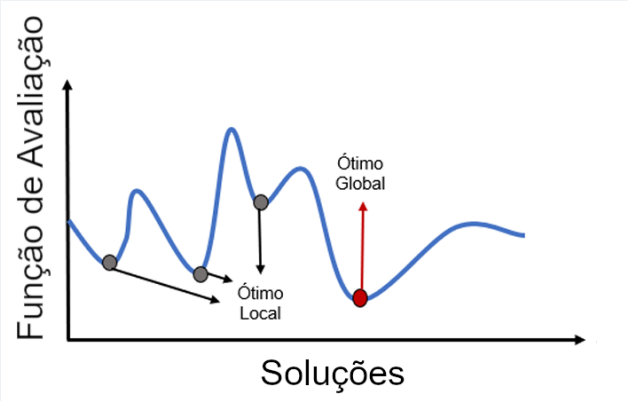


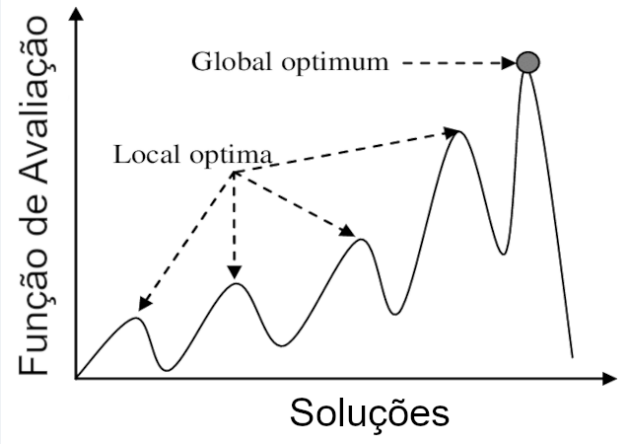


Heurísticas de Refinamento

Visão geral:

- As heurísticas de refinamento são as heurísticas de **busca local**.
- Elas atuam sobre soluções completas.
- E como vimos, elas são baseadas no conceito de vizinhança.
- Dentre os métodos, destacam-se:
  - Método da Descida (*Descent Method*)
  - Método da Subida (*Uphill Method* ou *Hill Climbing*)
  - **Importante:** Sem perda de generalidade, ambos são similares, altera-se o fato de ser um problema de min ou de max.

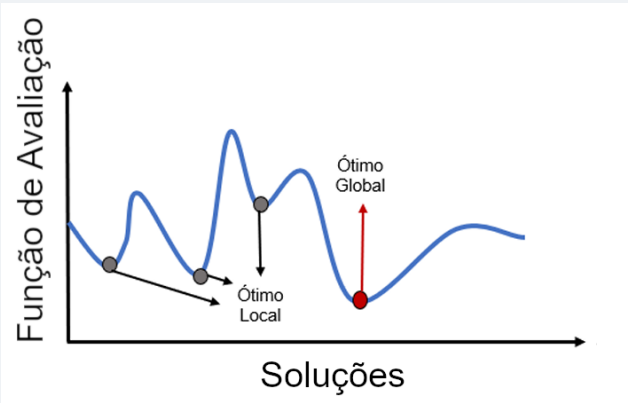


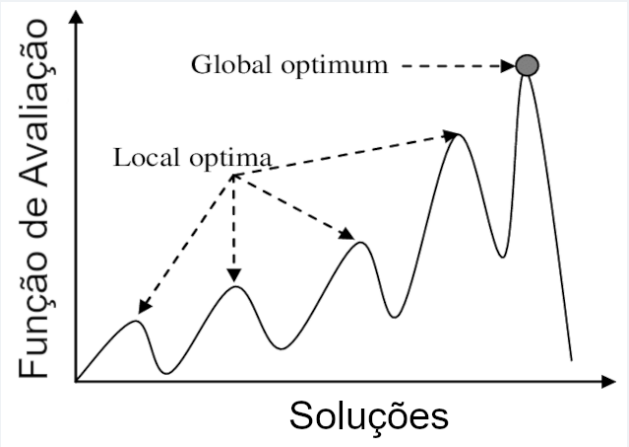


Heurísticas de Refinamento

Visão geral:

- As heurísticas de refinamento são as heurísticas de **busca local**.
- Elas atuam sobre soluções completas.
- E como vimos, elas são baseadas no conceito de vizinhança.
- Dentre os métodos, destacam-se:
  - Método da Descida (*Descent Method*)
  - Método da Subida (*Uphill Method* ou *Hill Climbing*)
  - **Importante:** Sem perda de generalidade, ambos são similares, altera-se o fato de ser um problema de min ou de max.

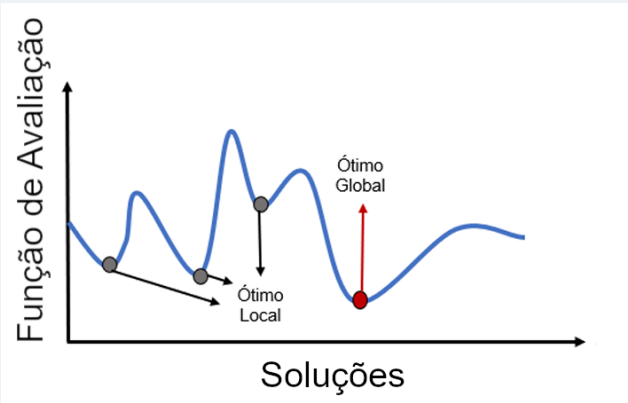


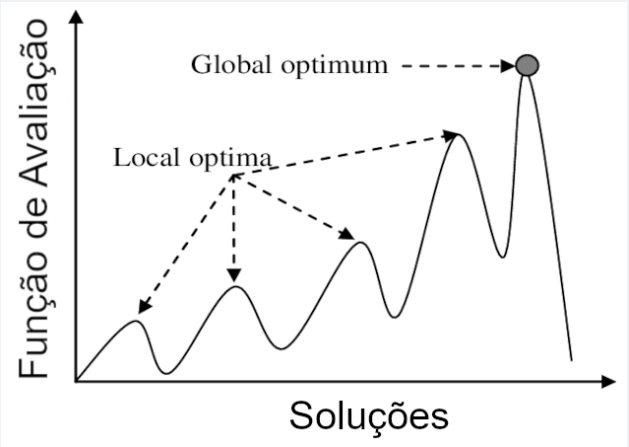


Heurísticas de Refinamento

Visão geral:

- As heurísticas de refinamento são as heurísticas de **busca local**.
- Elas atuam sobre soluções completas.
- E como vimos, elas são baseadas no conceito de vizinhança.
- Dentre os métodos, destacam-se:
  - Método da Descida (*Descent Method*)
  - Método da Subida (*Uphill Method* ou *Hill Climbing*)
  - **Importante:** Sem perda de generalidade, ambos são similares, altera-se o fato de ser um problema de min ou de max.





Heurísticas de Refinamento

Funcionamento

- Partir de uma **solução inicial** qualquer.
- Caminhar, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada, tentando **melhorar** a solução construída.
- Dentre as estratégias de busca temos:
  - Completa: *Best Improvement*
  - Primeira melhora: *First Improvement*
  - Randômica: *Random Improvement*

Heurísticas de Refinamento

Funcionamento

- Partir de uma **solução inicial** qualquer.
- Caminhar, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada, tentando **melhorar** a solução construída.
- Dentre as estratégias de busca temos:
  - Completa: *Best Improvement*
  - Primeira melhora: *First Improvement*
  - Randômica: *Random Improvement*

Heurísticas de Refinamento

Funcionamento

- Partir de uma **solução inicial** qualquer.
- Caminhar, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada, tentando **melhorar** a solução construída.
- Dentre as estratégias de busca temos:
  - Completa: *Best Improvement*
  - Primeira melhora: *First Improvement*
  - Randômica: *Random Improvement*

Heurísticas de Refinamento

Funcionamento

- Partir de uma **solução inicial** qualquer.
- Caminhar, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada, tentando **melhorar** a solução construída.
- Dentre as estratégias de busca temos:
  - Completa: *Best Improvement*
  - Primeira melhora: *First Improvement*
  - Randômica: *Random Improvement*

Heurísticas de Refinamento

Procedimento - *Best Improvement*

Algorithm 3: Template da Descida/Subida com Best Improvement

1 $s \leftarrow s_0$ (Solucao inicial)
2 $V = \{s' \in N(s) | f(s') < f(s)\}$ (Vizinhos de s)
3 **while** $|V| > 0$ **do**
4 | Seleciona $s' \in V$, onde $s' = argmin\{f(s') | s' \in V\}$ (Melhor vizinho)
5 | $s \leftarrow s'$
6 | $V = \{s' \in N(s) | f(s') < f(s)\}$ (Gera Vizinhos do novo s)
7 **end**
8 **return** s

- A ideia é selecionar o melhor vizinho da solução corrente s até não haver mais, i.e., $|V| = 0$.
- Nesse caso, para ser o método da subida altera-se as vizinhanças para $f(s') > f(s)$ nas linhas 2 e 6.
- E altera-se para *argmax* na linha 4.

Heurísticas de Refinamento

Procedimento - *First Improvement*

Algorithm 4: Template da Descida/Subida com First Improvement

```
1  s ← s_0 (Solucao inicial)
2  Melhora ← true
3  while Melhora do
4  |   Melhora ← false
5  |   for each s' ∈ N(s) do
6  |   |   if f(s') < f(s) then
7  |   |   |   s ← s'
8  |   |   |   Melhora ← true
9  |   |   |   break;
10 |   |   end
11 |   end
12 end
13 return s
```

- A ideia é selecionar o primeiro melhor vizinho da solução corrente s até não haver mais, quando a variável *Melhora* se manter *false*.
- Nesse caso, para ser o método da subida altera-se o teste do *if* para $f(s') > f(s)$ nas linha 6.

Heurísticas de Refinamento

Procedimento - Estratégia Randômica

- Variante do Método de Descida/Subida.
- Evita a pesquisa exaustiva pelo melhor vizinho.
- Consiste em escolher aleatoriamente um vizinho e o aceitar somente se ele for de melhora.
- Se o vizinho escolhido não for de melhora, a solução corrente permanece inalterada e outro vizinho é aleatoriamente gerado.
- O procedimento é interrompido após um certo número fixo de iterações sem melhora no valor da melhor solução obtida até então.
- A solução final não é necessariamente um ótimo local.

Heurísticas de Refinamento

Procedimento - Estratégia Randômica

Algorithm 5: Template da Descida Randômica (*Random Descent Method*)

```
1  Iter ← 0 (Contador de solucoes sem melhora)
2  s ← s_0 (Solucao inicial)
3  while Iter < IterMAX do
4  |   Iter ← Iter + 1
5  |   Seleciona aleatoriamente s' ∈ N(s)
6  |   if f(s') < f(s) then
7  |   |   Iter ← 0
8  |   |   s ← s'
9  |   end
10 end
11 return s
```

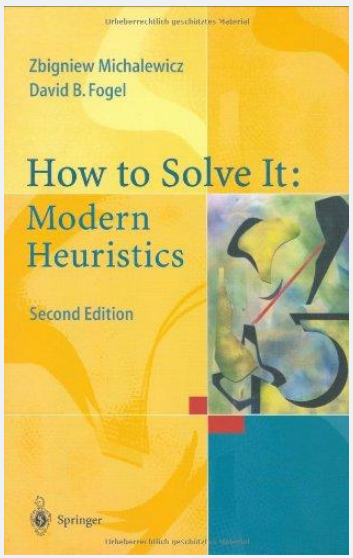
- A ideia é selecionar soluções vizinhas aleatórias que tem valor objetivo melhor, até atingir um máximo de iterações sem melhora.
- Nesse caso, para ser o método da subida altera-se o teste do *if* para $f(s') > f(s)$ nas linha 6.

Agenda

Referência

Livro

- Esse conteúdo está baseado no livro:
- MICHLEWICZ, Zbigniew; FOGEL, David B. How to solve it: modern heuristics. 2nd. ed. Berlin: Springer c2010 554 p. ISBN 9783642061349 (Capítulo 5, seção 5.1).



Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais**.

Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais**.

Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais**.

Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais**.

Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais**.

Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais.**

Introdução

Visão geral:

- Até agora vimos como métodos de busca, a Busca Exaustiva e a Busca Local:
  - Um é caro computacionalmente
  - Outro fica preso em ótimo local
- Como resolver isso?
  - Não há muita esperança de acelerar (alg. polinomial) a Busca Exaustiva.
  - A opção que resta é um algoritmo capaz de escapar de ótimo local
- O **Simulated Annealing** é uma *meta-heurística* que tenta **escapar de ótimos locais**.

Introdução

Visão geral:

- De forma breve vamos ver um comparativo entre os procedimentos:

Algorithm 6: Busca Local

```
1 s ← s_0 (Solucao inicial)
2 while melhora(s) ≠ 'nao' do
3 |   s ← melhora(s)
4 end
5 return s
```

Algorithm 7: Simulated Annealing

```
1 s ← s_0 (Solucao inicial)
2 while Nao condicao de termino do
3 |   s ← melhora?(s, T)
4 |   atualiza(T)
5 end
6 return s
```

- ***melhora(s)*** retorna uma solução melhor ou um **nao** (em caso de ótimo local).
- No SA, a função ***melhora(s)*** é subustituída pela ***melhora?(s, T)***, por alguns motivos.

Introdução

Visão geral:

- Existem 3 diferenças importantes entre eles:
  1. **A forma como os métodos param:** O SA tem uma condição de parada externa (num. iterações por exemplo); A busca local, para ao atingir um ótimo local.
  2. A função ***melhora?*** não necessariamente retorna uma sol. vizinha melhor, porque uma sol. pior pode ser aceita segundo o valor da temperatura T.
  3. A **temperatura T** é atualizada periodicamente, e esse valor influencia diretamente em ***melhora?***. A busca local, não tem isso.

Introdução

Visão geral:

- Existem 3 diferenças importantes entre eles:
  1. **A forma como os métodos param:** O SA tem uma condição de parada externa (num. iterações por exemplo); A busca local, para ao atingir um ótimo local.
  2. A função ***melhora?*** não necessariamente retorna uma sol. vizinha melhor, porque uma sol. pior pode ser aceita segundo o valor da temperatura T.
  3. A **temperatura T** é atualizada periodicamente, e esse valor influencia diretamente em ***melhora?***. A busca local, não tem isso.

Introdução

Visão geral:

- Existem 3 diferenças importantes entre eles:
  1. **A forma como os métodos param:** O SA tem uma condição de parada externa (num. iterações por exemplo); A busca local, para ao atingir um ótimo local.
  2. A função ***melhora?*** não necessariamente retorna uma sol. vizinha melhor, porque uma sol. pior pode ser aceita segundo o valor da temperatura T.
  3. A **temperatura T** é atualizada periodicamente, e esse valor influencia diretamente em ***melhora?***. A busca local, não tem isso.

Introdução

Visão geral:

- Existem 3 diferenças importantes entre eles:
  1. **A forma como os métodos param:** O SA tem uma condição de parada externa (num. iterações por exemplo); A busca local, para ao atingir um ótimo local.
  2. A função ***melhora?*** não necessariamente retorna uma sol. vizinha melhor, porque uma sol. pior pode ser aceita segundo o valor da temperatura T.
  3. A **temperatura T** é atualizada periodicamente, e esse valor influencia diretamente em ***melhora?***. A busca local, não tem isso.

Metodologia

Visão geral - *Hill-Climbing*:

- Vamos chegar no pseudocódigo do **Simulated Annealing** partindo do *Hill-Climbing* (Lembrando que o Hill-Climbing é a busca local).

Algorithm 8: Método da Subida com Best Improvement - Hill Climbing

```
1 s ← s_0 (Solucao inicial aleatoria)
2 V = {s' ∈ N(s)|f(s') > f(s)} (Vizinhos de s)
3 while |V| > 0 do
4 |   s' = argmax{f(s')|s' ∈ V} (Melhor vizinho)
5 |   s ← s'
6 |   V = {s' ∈ N(s)|f(s') > f(s)} (Gera Vizinhos do novo s)
7 end
8 return s
```

- O próximo passo é o método ***Iterated Hill-Climbing***:
  - Ele inclui um número máximo de iterações (MAX).
  - E vários pontos de início para tentar buscar por melhores soluções no espaço.

Metodologia

Visão geral - *Hill-Climbing*:

- Vamos chegar no pseudocódigo do **Simulated Annealing** partindo do *Hill-Climbing* (Lembrando que o Hill-Climbing é a busca local).

Algorithm 9: Método da Subida com Best Improvement - Hill Climbing

```
1 s ← s_0 (Solucao inicial aleatoria)
2 V = {s' ∈ N(s)|f(s') > f(s)} (Vizinhos de s)
3 while |V| > 0 do
4 |   s' = argmax{f(s')|s' ∈ V} (Melhor vizinho)
5 |   s ← s'
6 |   V = {s' ∈ N(s)|f(s') > f(s)} (Gera Vizinhos do novo s)
7 end
8 return s
```

- O próximo passo é o método ***Iterated Hill-Climbing***:
  - Ele inclui um número máximo de iterações (MAX).
  - E vários pontos de início para tentar buscar por melhores soluções no espaço.

Metodologia

Visão geral - *Iterated Hill-Climbing*:

Algorithm 10: Iterated Hill Climbing

```
1  t ← 0
2  s_best ← ∅
3  while t < MAX do
4  |   s ← s_0 (Solucao inicial aleatoria)
5  |   V = {s' ∈ N(s)|f(s') > f(s)} (Vizinhos de s)
6  |   while |V| > 0 do
7  |   |   s' = argmax{f(s')|s' ∈ V} (Melhor vizinho)
8  |   |   s ← s'
9  |   |   V = {s' ∈ N(s)|f(s') > f(s)} (Gera Vizinhos do novo s)
10 |   end
11 |   if f(s) > f(s_best) then
12 |   |   s_best ← s
13 |   end
14 |   t ← t + 1
15 end
16 return s_best
```

Metodologia

Visão geral - *Stochastic Hill-Climbing*:

- O próximo passo é o método *Iterated* → ***Stochastic Hill-Climbing***:
  - Em vez de verificar todos os vizinhos de s e escolher o melhor, selecionar somente um vizinho s' qualquer.
  - Em vez de aceitá-lo somente se for melhor, **aceitá-lo com certa probabilidade**, dependendo da diferença relativa entre os méritos (avaliações).

Algorithm 11: Stochastic Hill Climbing

1 $t \leftarrow 0$
2 $s \leftarrow s_0$ (Solucao inicial aleatoria)
3 **while** $t < MAX$ **do**
4 $\quad s' \in N(s)$ (Seleciona um vizinho qualquer de s)
5 $\quad s \leftarrow s'$ com uma certa probabilidade $\frac{1}{1+e^{\frac{f(s)-f(s')}{T}}}$
6 $\quad t \leftarrow t+1$
7 **end**
8 **return** s

Metodologia

Visão geral - *Stochastic Hill Climbing*:

- Probabilidade de aceitação (p): $p = \frac{1}{1+e^{\frac{f(s)-f(s')}{T}}}$
  - Depende da diferença entre a avaliação da solução corrente (s) e da próxima (s').
  - Depende de certo valor constante T.

Metodologia

Visão geral - *Stochastic Hill Climbing*:

- Probabilidade de aceitação (p): $p = \frac{1}{1+e^{\frac{f(s)-f(s')}{T}}}$
- p é muito alta para T pequeno (para T=1 praticamente *Hill-Climbing*)
- p é cerca de 50% para T grande (praticamente aleatório)

- Suponha $f(s) = 107$ e $f(s') = 120$
- i.e., a nova solução é melhor por 13 un.
- Nesse caso $f(s) - f(s') = -13$

T	$e^{-13/T}$	p
1	0.000002	1.00
5	0.0743	0.93
10	0.2725	0.78
20	0.52	0.66
50	0.77	0.56
1010	0.9999	0.5

Probabilidade de aceitação em função de T

Metodologia

Visão geral - *Stochastic Hill Climbing*:

- Probabilidade de aceitação (p):
  $p = \frac{1}{1+e^{\frac{f(s)-f(s')}{T}}}$
- Note que p é:
  - alta para soluções boas
  - baixa para soluções ruins
  - 50% para soluções de mesmo valor (tanto faz)

- Suponha $f(s) = 107$ e $T = 10$
- A probabilidade de aceitação depende do valor da nova solução

$f(s')$	$f(s) - f(s')$	$e^{(f(s)-f(s'))/10}$	p
80	27	14.88	0.06
100	7	2.01	0.33
107	0	1.00	0.50
120	-13	0.27	0.78
150	-43	0.01	0.99

Probabilidade de aceitação em função de $f(s')$

Metodologia

Visão geral - *Simulated Annealing (SA)*:

- *Stochastic Hill Climbing* $\rightarrow$ *Simulated Annealing*
- O valor do parâmetro T muda durante a execução:
  - Começa alto: quase uma busca randômica pura.
  - Termina baixo: quase um hill-climber puro.
- Uma solução **melhor é sempre aceita**.

Metodologia

Visão geral - *Simulated Annealing (SA)*:

Algorithm 12: Simulated Annealing

```
1  T ← t_0 (Temperatura inicial)
2  s ← s_0 (Solucao inicial aleatoria)
3  s_best ← ∅
4  while nao condicao parada do
5      while nao condicao termino do
6          s' ∈ N(s) (Seleciona um vizinho qualquer de s)
7          if f(s') > f(s) then
8              s ← s'
9              if f(s') > f(s_best) then
10                 s_best ← s' (Melhor sol. até agora)
11             end
12         end
13         else if random[0,1) < e^((f(s')−f(s))/T) then
14             s ← s'
15         end
16     end
17     T ← α.T (Atualiza temperatura)
18 end
19 return s_best
```

Metodologia

Visão geral - *Simulated Annealing (SA)*:

- Note que, então, na linha 7, se uma sol. vizinha for melhor ela é aceita.
- Caso seja pior, ela pode ser aceita segundo a probabilidade, que considera o valor da temperatura atual.
- Esse mecanismo de aceitar piores soluções traz a característica do SA de escapar de possíveis ótimos locais.
- A melhor solução encontrada até então, sempre será guardada no s_{best}.

Observações:

- Esse algoritmo representa um problema de **max**.
- Se fosse um de minimização, as mudanças seriam:
  - Linha 8: $f(s') < f(s)$
  - Linha 12: $\text{random}[0,1) < e^{\frac{f(s)-f(s')}{T}}$, porque o e deve ficar elevado a um valor negativo, para gerar uma probabilidade entre 0 e 1.

Metodologia

Visão geral - *Simulated Annealing (SA)*:

- Note que, então, na linha 7, se uma sol. vizinha for melhor ela é aceita.
- Caso seja pior, ela pode ser aceita segundo a probabilidade, que considera o valor da temperatura atual.
- Esse mecanismo de aceitar piores soluções traz a característica do SA de escapar de possíveis ótimos locais.
- A melhor solução encontrada até então, sempre será guardada no s_{best}.

Observações:

- Esse algoritmo representa um problema de **max**.
- Se fosse um de minimização, as mudanças seriam:
  - Linha 8: $f(s') < f(s)$
  - Linha 12: $\text{random}[0,1) < e^{\frac{f(s)-f(s')}{T}}$, porque o e deve ficar elevado a um valor negativo, para gerar uma probabilidade entre 0 e 1.

Metodologia

Observações:

- Em relação às condições de término e parada, geralmente pode ser utilizado o seguinte:
  - **Condição de parada:** Relaciona-se com a Temperatura T atual, $T > 0$, para quando a temperatura diminui para 0, processo de resfriamento usando algum $\alpha < 1$.
  - **Condição de termino:** Pode utilizando um número pré-definido de iterações, denominado SA_{max}, que caracteriza, um número fixo de iterações para cada temperatura.
- Com tais mudanças o algoritmo fica assim, próximo slide.

Metodologia

Visão geral - *Simulated Annealing (SA)*:

Algorithm 13: Simulated Annealing

```
1  T ← t_0 (Temperatura inicial)
2  s ← s_0 (Solucao inicial aleatoria)
3  s_best ← ∅
4  i ← 0
5  while T > 0 do
6      while i < SA_max do
7          i ← i + 1
8          s' ∈ N(s) (Seleciona um vizinho qualquer de s)
9          if f(s') > f(s) then
10             s ← s'
11             if f(s') > f(s_best) then
12                 s_best ← s' (Melhor sol. até agora)
13             end
14         end
15         else if random[0, 1) < e^((f(s')−f(s))/T) then
16             s ← s'
17         end
18     end
19     T ← α.T (Atualiza temperatura)
20     i ← 0
21 end
22 return s_best
```

Considerações

Considerações:

- *Simulated Annealing* significa Recozimento Simulado.
- A ideia básica do SA é uma analogia da termodinâmica para recozimento de metais (recristalização).
- Para produzir um material cristalino, sem imperfeições:
  - Aquecer o material até um estado fundido
  - Resfriar bem lentamente
- Se não aquecer o suficiente no início ou se resfriar rapidamente, irregularidades acabam ficando dentro da estrutura

Comparativo:

- No SA a analogia está na temperatura T que deve iniciar alta, e o seu processo de resfriamento pela fórmula $T \leftarrow \alpha.T$ que deve ser lento.
- Além disso, à medida que T diminui, o aceite de soluções piores também diminuirá.

Considerações

Considerações:

- *Simulated Annealing* significa Recozimento Simulado.
- A ideia básica do SA é uma analogia da termodinâmica para recozimento de metais (recristalização).
- Para produzir um material cristalino, sem imperfeições:
  - Aquecer o material até um estado fundido
  - Resfriar bem lentamente
- Se não aquecer o suficiente no início ou se resfriar rapidamente, irregularidades acabam ficando dentro da estrutura

Comparativo:

- No SA a analogia está na temperatura T que deve iniciar alta, e o seu processo de resfriamento pela fórmula $T \leftarrow \alpha.T$ que deve ser lento.
- Além disso, à medida que T diminui, o aceite de soluções piores também diminuirá.

Considerações

Questões:

- Como qualquer outro algoritmo de busca:
  - Como representar uma solução?
  - Como determinar uma solução inicial?
  - Como avaliar o custo de uma solução?
  - Quem são os vizinhos de uma dada solução?

Questões específicas do SA:

- Como determinar a temperatura inicial T ?
- Como determinar o condição de término (mudança de temperatura) ?
- Como determinar a taxa de resfriamento α ?
- Como determinar o critério de parada (*ponto de congelamento*) ?

Considerações

Questões:

- Como qualquer outro algoritmo de busca:
  - Como representar uma solução?
  - Como determinar uma solução inicial?
  - Como avaliar o custo de uma solução?
  - Quem são os vizinhos de uma dada solução?

Questões específicas do SA:

- Como determinar a temperatura inicial T ?
- Como determinar o condição de término (mudança de temperatura) ?
- Como determinar a taxa de resfriamento α ?
- Como determinar o critério de parada (*ponto de congelamento*) ?

Considerações - Variações do SA

Temperatura inicial - Parâmetro a ser calibrado:

- Muito alta fica randômico, muita baixa fica busca local
- *Accept all*: alta o bastante para aceitar qualquer vizinho no início
- *Acceptance ratio*: experimento preliminar sobre taxa de aceitação
- Pode-se gerar várias soluções iniciais e definir como o MAIOR valor encontrado dentre todas as avaliações.

Condição de término (mudança de temperatura):

- SA_{max}: Número máximo de iterações em uma dada temperatura.
- **Static:** determinado antes
- **Adaptive:** definido durante a busca
- Exemplo: Caixeiro Viajante, $SA_{max} = k.n$, n = num de cidades, k = constante a ser calibrada.

Considerações - Variações do SA

Temperatura inicial - Parâmetro a ser calibrado:

- Muito alta fica randômico, muita baixa fica busca local
- *Accept all*: alta o bastante para aceitar qualquer vizinho no início
- *Acceptance ratio*: experimento preliminar sobre taxa de aceitação
- Pode-se gerar várias soluções iniciais e definir como o MAIOR valor encontrado dentre todas as avaliações.

Condição de término (mudança de temperatura):

- SA_{max}: Número máximo de iterações em uma dada temperatura.
- **Static:** determinado antes
- **Adaptive:** definido durante a busca
- Exemplo: Caixeiro Viajante, $SA_{max} = k.n$, n = num de cidades, k = constante a ser calibrada.

Considerações - Variações do SA

Taxa de Resfriamento - Parâmetro a ser calibrado:

- Deve-se balancear custo computacional (tempo) e qualidade da solução.
- Linear: $T \leftarrow T - \beta$
- Geométrico: $T \leftarrow \alpha.T \mid \alpha \in (0,1)$, (mais usado)
- Logarítmico: $T_i \leftarrow T_0/\log(i)$

Critério de parada (*ponto de congelamento*):

- Temperatura final (estado de "congelamento", temperatura bem baixa).
- T_{final} = zero da máquina.
- $T_{final} = 0.01$ (ou 0.001).
- Número de iterações sem melhora.

- **Obs:** Os valores dos parâmetros mais adequados são dependentes do problema e da instância.

Considerações - Variações do SA

Taxa de Resfriamento - Parâmetro a ser calibrado:

- Deve-se balancear custo computacional (tempo) e qualidade da solução.
- Linear: $T \leftarrow T - \beta$
- Geométrico: $T \leftarrow \alpha.T \mid \alpha \in (0, 1)$, (mais usado)
- Logarítmico: $T_i \leftarrow T_0 / \log(i)$

Critério de parada (*ponto de congelamento*):

- Temperatura final (estado de "congelamento", temperatura bem baixa).
- T_{final} = zero da máquina.
- $T_{final} = 0.01$ (ou 0.001).
- Número de iterações sem melhora.

- **Obs:** Os valores dos parâmetros mais adequados são dependentes do problema e da instância.

Considerações - Variações do SA

Taxa de Resfriamento - Parâmetro a ser calibrado:

- Deve-se balancear custo computacional (tempo) e qualidade da solução.
- Linear: $T \leftarrow T - \beta$
- Geométrico: $T \leftarrow \alpha.T \mid \alpha \in (0, 1)$, (mais usado)
- Logarítmico: $T_i \leftarrow T_0 / \log(i)$

Critério de parada (*ponto de congelamento*):

- Temperatura final (estado de "congelamento", temperatura bem baixa).
- T_{final} = zero da máquina.
- $T_{final} = 0.01$ (ou 0.001).
- Número de iterações sem melhora.

- **Obs:** Os valores dos parâmetros mais adequados são dependentes do problema e da instância.

Exercício

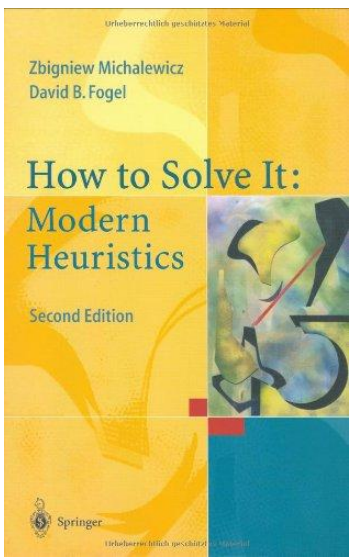
1. Faça um *Simulated Annealing* para o TSP baseado no Algoritmo 13 da página 30:
   (a) Gere soluções iniciais de forma aleatória.
   (b) Defina uma vizinhança à sua escolha.
   (c) Baseado na aula, defina uma temperatura inicial, o número de iterações por temperatura (SA_{max}), um valor α para um resfriamento lento e uma condição de parada.

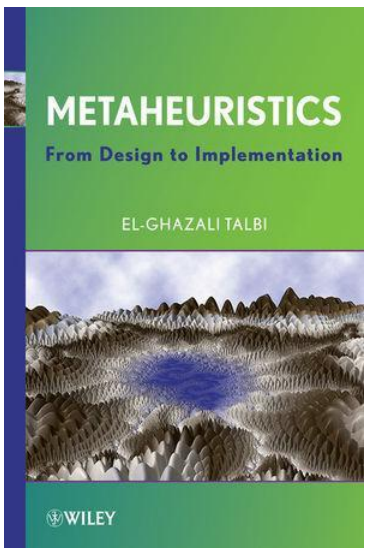
   Obs: Podemos usar a mesmas instâncias utilizadas na busca local anteriormente.

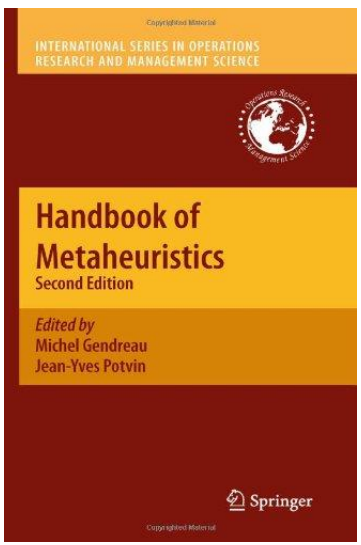
Bibliografias

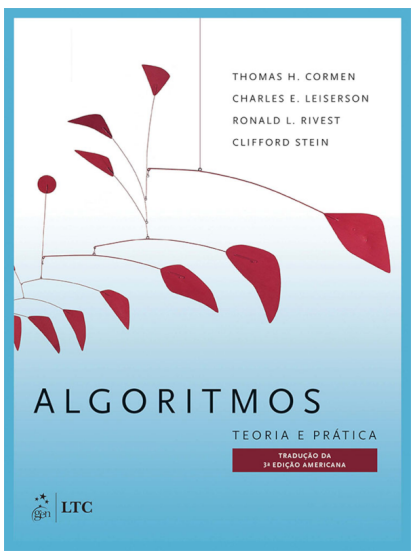
Bibliografia Básica

1. MICHLEWICZ, Zbigniew; FOGEL, David B. How to solve it: modern heuristics. 2nd. ed. Berlin: Springer c2010 554 p. ISBN 9783642061349.
2. Talbi, El-Ghazali; Metaheuristics: From Design to Implementation, Wiley Publishing, 2009.
3. GENDREAU, Michel. Handbook of metaheuristics. 2.ed. New York: Springer 2010 648 p. (International series in operations research & management science ; 146).
4. T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, 3rd edition, 2009 (**Pergamum**).









Bibliografias

Bibliografia Complementar

1. GLOVER, Fred; KOCHENBERGER, Gary A. (ed.). Handbook of metaheuristics. Boston: Kluwer, 2003. 556 p. (International series in operations research & management science ; 57).
2. BLUM, Christian Et Al. Hybrid metaheuristics: an emerging approach to optimization. Berlin: Springer 2008 289 p. (Studies in Computational intelligence; 114).
3. DOERNER, Karl F. (ed.) Et Al. Metaheuristics: progress in complex systems optimization. New York: Springer 2007 408 p. (Operations research / computer science interfaces series).
4. GLOVER, Fred; LAGUNA, Manuel. Tabu search. Boston: Kluwer Academic, 1997. 382 p.
5. AARTS, Emile. Local search in combinatorial optimization. Princeton: Princeton University Press, 2003 512 p.
6. Gaspar-Cunha, A.; Takahashi, R.; Antunes, C.H.; Manual de Computação Evolutiva e Metaheurística; Belo Horizonte: Editora UFMG; Coimbra: Imprensa da Universidade de Coimbra; 2013.