GRASP - Greedy Randomized Adaptive Search Procedures

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)

Agenda

Introdução

Visão geral:

- O GRASP (Greedy Randomized Adaptive Search Procedures) é uma metaheuristica considerada *multi-start* ou como um processo iterativo.
- O método foi desenvolvido por um pesquisador brasileiro em 1989 na AT & T.
- Na tradução seria algo como, *procedimentos de busca gulosos aleatórios e adaptativos.*
- Na ideia principal, ela aplica o método de busca local repetidamente a partir de soluções construídas por um algoritmo guloso aleatório.
- Então ela consiste basicamente de duas fases:
  - Construção: *constrói uma solução de forma parcialmente gulosa.*
  - Busca Local: *melhora a solução construída por meio da aplicação de um método de busca local.*

Introdução

Visão geral:

- O método tenta viabilizar a diversificação.
- Ou seja, ele vai tentar aplicar a busca local a partir de várias soluções geradas aleatoriamente.
- E buscar por um caminho de melhoria de custo que parte de uma dessas soluções até uma solução ótima ou próxima do ótimo.
- O melhor ótimo local dentre todas as buscas locais é retornado como solução da metaheurística.
- Ele guarda a melhor solução encontrada ao longo das iterações.

Introdução

Pseudo-código:

- O pseudo-código do GRASP básico é assim, supondo um problema de **Minimização**:
- Um exemplo de condição de parada pode ser um número máximo de iterações: $iter < GRASPMax$.
- Um segundo parâmetro importante a considerar o chamado (α) que explico adiante.

Algorithm 1: GRASP Básico

```
1  s_best ← ∞
2  iter ← 0
3  while condicao parada não satisfeita do
4  |    s ← ConstrucaoGulosaAleatoria(α)
5  |    s ← BuscaLocal(s)
6  |    if f(s) < f(s_best) then
7  |    |    s_best ← s
8  |    end
9  |    iter ← iter + 1
10 end
11 return s_best
```

Construção

Visão geral:

- A etapa de **construção** é realizada elemento a elemento.
- Se for uma construção gulosa:
  - Cada candidato é avaliado por uma função gulosa (o melhor naquele momento).
  - O candidato com a melhor avaliação é escolhido.
- Se for uma construção aleatória:
  - Um candidato é escolhido aleatoriamente entre os disponíveis.

Construção

Visão geral:

- A etapa de **construção** é realizada elemento a elemento.
- Se for uma construção gulosa:
  - Cada candidato é avaliado por uma função gulosa (o melhor naquele momento).
  - O candidato com a melhor avaliação é escolhido.
- Se for uma construção aleatória:
  - Um candidato é escolhido aleatoriamente entre os disponíveis.

Construção

Visão geral:

- A etapa de **construção** é realizada elemento a elemento.
- Se for uma construção gulosa:
  - Cada candidato é avaliado por uma função gulosa (o melhor naquele momento).
  - O candidato com a melhor avaliação é escolhido.
- Se for uma construção aleatória:
  - Um candidato é escolhido aleatoriamente entre os disponíveis.

Construção

Visão geral:

- Quais as **vantagens** da ..
- Construção gulosa:
  - Soluções de boa qualidade.
  - Busca local converge rapidamente para um ótimo local.
- Construção aleatória:
  - Diversidade nas soluções geradas.
  - Busca de forma mais ampla pelo espaço de soluções.

Construção

Visão geral:

- Quais as **vantagens** da ..
- Construção gulosa:
  - Soluções de boa qualidade.
  - Busca local converge rapidamente para um ótimo local.
- Construção aleatória:
  - Diversidade nas soluções geradas.
  - Busca de forma mais ampla pelo espaço de soluções.

Construção

Visão geral:

- Quais as **vantagens** da ..
- Construção gulosa:
  - Soluções de boa qualidade.
  - Busca local converge rapidamente para um ótimo local.
- Construção aleatória:
  - Diversidade nas soluções geradas.
  - Busca de forma mais ampla pelo espaço de soluções.

Construção

Visão geral:

- Quais as **DESvantagens** da ..
- Construção gulosa:
  - Pouca ou quase nenhuma diversificação.
  - Soluções são geralmente sub-ótimos.
  - Busca local raramente vai convergir para um ótimo global.
- Construção aleatória:
  - Soluções tem baixa qualidade.
  - Busca local é mais lenta para chegar a um ótimo local.

Construção

Visão geral:

- Quais as **DESvantagens** da ..
- Construção gulosa:
  - Pouca ou quase nenhuma diversificação.
  - Soluções são geralmente sub-ótimos.
  - Busca local raramente vai convergir para um ótimo global.
- Construção aleatória:
  - Soluções tem baixa qualidade.
  - Busca local é mais lenta para chegar a um ótimo local.

Construção

Visão geral:

- Quais as **DESvantagens** da ..
- Construção gulosa:
  - Pouca ou quase nenhuma diversificação.
  - Soluções são geralmente sub-ótimos.
  - Busca local raramente vai convergir para um ótimo global.
- Construção aleatória:
  - Soluções tem baixa qualidade.
  - Busca local é mais lenta para chegar a um ótimo local.

Construção

Visão geral:

- A função **ConstrucaoGulosaAleatoria**(α) tenta então pegar as características boas da
  - Construção gulosa:
    - Soluções de boa qualidade.
    - Convergência rápida para o ótimo local.
  - e Construção aleatória:
    - Diversificação
- E tenta evitar as características ruins da
  - Construção gulosa:
    - Pouca ou quase nenhuma diversificação.
  - e Construção aleatória:
    - Soluções de baixa qualidade.
    - Convergência lenta.

Construção

Visão geral:

- A função **ConstrucaoGulosaAleatoria**(α) tenta então pegar as características boas da
  - Construção gulosa:
    - Soluções de boa qualidade.
    - Convergência rápida para o ótimo local.
  - e Construção aleatória:
    - Diversificação
- E tenta evitar as características ruins da
  - Construção gulosa:
    - Pouca ou quase nenhuma diversificação.
  - e Construção aleatória:
    - Soluções de baixa qualidade.
    - Convergência lenta.

Construção

Fundamento:

- Cria-se uma *lista de candidatos (LC)*.
- Cada candidato é inserido na LC através do seu valor na função gulosa $g()$ (também chamado *custo incremental*).
- Enquanto houver candidatos da LC:
  - Forma-se uma **lista restrita de candidatos (LRC)** com os melhores candidatos segundo a função $g()$.
  - Escolhe-se aleatoriamente um elemento candidato dessa **LRC**.
  - Insere-se esse elemento da **LRC** na solução parcial s.
  - Atualiza-se a **LC**, excluindo-se o elemento inserido na solução parcial s.
- A fase de construção termina retornando uma solução s.

Formação:

- **LC**: Contém todos os elementos que podem ser inseridos.
- **LRC**: Contém apenas os **"melhores"** da LC.
  - Inserir na LRC apenas os p melhores ou os melhores que um certo valor.

Construção

Na formação da Lista Restrita de Canditados (LRC):

- Por cardinalidade:
  - Os p melhores elementos.
  - Os melhores $max\{1, \alpha \times LC.size()\}$
- Por valor:
  - Seja c_{min} e c_{max} o respectivo custo incremental do melhor e pior candidatos da LC.
  - Define-se um intervalo entre: $[c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$
  - Ou seja, $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$

Alpha (α):

- O $\alpha \in [0, 1]$ é um parâmetro que mensura o grau de *aleatoriade/gulosidade* da escolha do candidato $c \in LC$:
  - Se $\alpha = 0$, a escolha será puramente gulosa.
  - Se $\alpha = 1$, a escolha será puramente aleatória.
  - Se $0 < \alpha < 1$, a escola será ***GulosaAleatória***.

Construção

Na formação da Lista Restrita de Canditados (LRC):

- Por cardinalidade:
  - Os p melhores elementos.
  - Os melhores $max\{1, \alpha \times LC.size()\}$
- Por valor:
  - Seja c_{min} e c_{max} o respectivo custo incremental do melhor e pior candidatos da LC.
  - Define-se um intervalo entre: $[c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$
  - Ou seja, $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$

Alpha (α):

- O $\alpha \in [0, 1]$ é um parâmetro que mensura o grau de *aleatoriade*/*gulosidade* da escolha do candidato $c \in LC$:
  - Se $\alpha = 0$, a escolha será puramente gulosa.
  - Se $\alpha = 1$, a escolha será puramente aleatória.
  - Se $0 < \alpha < 1$, a escola será ***GulosaAleatória***.

Construção

Na formação da Lista Restrita de Canditados (LRC):

- Por cardinalidade:
  - Os p melhores elementos.
  - Os melhores $max\{1, \alpha \times LC.size()\}$
- Por valor:
  - Seja c_{min} e c_{max} o respectivo custo incremental do melhor e pior candidatos da LC.
  - Define-se um intervalo entre: $[c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$
  - Ou seja, $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$

Alpha (α):

- O $\alpha \in [0, 1]$ é um parâmetro que mensura o grau de *aleatoriade/gulosidade* da escolha do candidato $c \in LC$:
  - Se $\alpha = 0$, a escolha será puramente gulosa.
  - Se $\alpha = 1$, a escolha será puramente aleatória.
  - Se $0 < \alpha < 1$, a escola será ***GulosaAleatória***.

Construção

Na formação da Lista Restrita de Canditados (LRC):

- Por cardinalidade:
  - Os p melhores elementos.
  - Os melhores $max\{1, \alpha \times LC.size()\}$
- Por valor:
  - Seja c_{min} e c_{max} o respectivo custo incremental do melhor e pior candidatos da LC.
  - Define-se um intervalo entre: $[c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$
  - Ou seja, $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$

Alpha (α):

- O $\alpha \in [0, 1]$ é um parâmetro que mensura o grau de *aleatoriade/gulosidade* da escolha do candidato $c \in LC$:
  - Se $\alpha = 0$, a escolha será puramente gulosa.
  - Se $\alpha = 1$, a escolha será puramente aleatória.
  - Se $0 < \alpha < 1$, a escola será ***GulosaAleatória***.

Construção

Exemplo da Lista Restrita de Canditados (LRC):

- Por cardinalidade: Os melhores $LRC = max\{1, \alpha\% \times LC.size()\}$
- Por valor: $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$
- Exemplo:
  - $LC = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$
  - $g(c) = \{10, 13, 20, 18, 30, 21, 19, 28, 23, 12\}$
  - Seja $\alpha = 0.2 = (20\%)$.
- Por cardinalidade:
  - melhores 2 $(20\% \times 10)$.
  - $LRC = \{c_1, c_{10}\}$.
- Por valor:
  - $c_{min} = 10,\ c_{max} = 30$.
  - $g(c) \leq 10 + 0.2(30 - 10) \Rightarrow\ g(c) \leq 14 \therefore LRC = \{c_1, c_2, c_{10}\}$.

Construção

Exemplo da Lista Restrita de Canditados (LRC):

- Por cardinalidade: Os melhores $LRC = max\{1, \alpha\% \times LC.size()\}$
- Por valor: $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$
- Exemplo:
  - $LC = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$
  - $g(c) = \{10, 13, 20, 18, 30, 21, 19, 28, 23, 12\}$
  - Seja $\alpha = 0.2 = (20\%)$.
- Por cardinalidade:
  - melhores 2 $(20\% \times 10)$.
  - $LRC = \{c_1, c_{10}\}$.
- Por valor:
  - $c_{min} = 10,\ c_{max} = 30$.
  - $g(c) \leq 10 + 0.2(30 - 10) \Rightarrow\ g(c) \leq 14 \therefore LRC = \{c_1, c_2, c_{10}\}$.

Construção

Exemplo da Lista Restrita de Canditados (LRC):

- Por cardinalidade: Os melhores $LRC = max\{1, \alpha\% \times LC.size()\}$
- Por valor: $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$
- Exemplo:
  - $LC = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$
  - $g(c) = \{10, 13, 20, 18, 30, 21, 19, 28, 23, 12\}$
  - Seja $\alpha = 0.2 = (20\%)$.
- Por cardinalidade:
  - melhores 2 $(20\% \times 10)$.
  - $LRC = \{c_1, c_{10}\}$.
- Por valor:
  - $c_{min} = 10,\ c_{max} = 30.$
  - $g(c) \leq 10 + 0.2(30 - 10) \Rightarrow\ g(c) \leq 14 \therefore LRC = \{c_1, c_2, c_{10}\}.$

Construção

Exemplo da Lista Restrita de Canditados (LRC):

- Por cardinalidade: Os melhores $LRC = max\{1, \alpha\% \times LC.size()\}$
- Por valor: $LRC = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$
- Exemplo:
  - $LC = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$
  - $g(c) = \{10, 13, 20, 18, 30, 21, 19, 28, 23, 12\}$
  - Seja $\alpha = 0.2 = (20\%)$.
- Por cardinalidade:
  - melhores 2 $(20\% \times 10)$.
  - $LRC = \{c_1, c_{10}\}$.
- Por valor:
  - $c_{min} = 10,\ c_{max} = 30$.
  - $g(c) \leq 10 + 0.2(30 - 10) \Rightarrow\ g(c) \leq 14 \therefore LRC = \{c_1, c_2, c_{10}\}$.

Construção

Pseudocódigo:

- O pseudocódigo da **fase de construção** seria assim:

Algorithm 2: ConstrucaoGulosaAleatoria(α)

```
1 s ← ∅
2 Cria LC
3 while |LC| > 0 do
4 |   Cria LRC(α)  (Lista Restrita de Candidatos)
5 |   x ← um elemento aleatório ∈ LRC(α)
6 |   s ← s ∪ {x}
7 |   Atualiza LC  (Remove x, atualiza valores)
8 end
9 return s
```

- O pseudocódigo é adaptado para cada tipo de problema abordado.
- As linhas 4, 5 e 6 representam a parte *GulosaAleatória* (*Greedy Randomized*).
- A linha 7 representa a parte *Adaptativa* (*Adaptive*).

Busca Local

Visão geral:

- Na fase da busca local, aplica-se a heurística de refinamento já conhecida por nós.
- Neste caso, temos o $\mathbf{s_0}$ como solução inicial criada pela **ConstrucaoGulosaAleatoria**.

Algorithm 3: Busca Local com Best Improvement

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

- Lembrando das pequenas modificações caso seja um problema de Max.

GRASP

Pseudo-código:

- E como visto, guarda-se a melhor solução encontrada ao longo das iterações.
- Os parâmetros considerados pelo GRASP então são dois: $GRASPMax$ e α.

Algorithm 4: GRASP Básico

```
1  s_best ← ∞
2  iter ← 0
3  while iter < GRASPMax do
4  |    s ← ConstrucaoGulosaAleatoria(α)
5  |    s ← BuscaLocal(s)
6  |    if f(s) < f(s_best) then
7  |    |    s_best ← s
8  |    end
9  |    iter ← iter + 1
10 end
11 return s_best
```

Agenda

Exemplo

TSP Simétrico (todas as ligações) de exemplo:

- Vamos considerar um total de 8 cidades e que a rota começa da cidade 0.

Construção e LRC por valor (iteração 1):

- Solução parcial $s = \{0\}$.
- $LC = \{1, 2, 3, 4, 5, 6, 7\}$
- $g(c) =$ distância do candidato $c \in LC$ à ultima cidade inserida (0).
- $g(c) = \{20, 9, 13, 17, 10, 18, 23\}$
- $c_{min} = 9$, $c_{max} = 23$ e seja $\alpha = 0.4$.
- $LRC(\alpha) = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}$
- $LRC(\alpha) = \{c \in LC \mid g(c) \leq 9 + 0.4 \times (23 - 9)\}$
- $LRC(\alpha) = \{c \in LC \mid g(c) \leq 14.6\}$
- $LRC(\alpha) = \{2, 3, 5\}$

- Vamos supor que aleatoriamente $s \leftarrow s \cup \{3\}$.
- Atualiza $LC \leftarrow LC \setminus \{3\} = \{1, 2, 4, 5, 6, 7\}$

Exemplo

TSP Simétrico (todas as ligações) de exemplo:

Construção e LRC por valor (iteração 2):

- Solução parcial $s = \{0, 3\}$.
- $LC = \{1, 2, 4, 5, 6, 7\}$
- $g(c) =$ distância do candidato $c \in LC$ à ultima cidade inserida (3).
- $g(c) = \{12, 4, 14, 16, 13, 8\}$
- $c_{min} = 4$, $c_{max} = 16$, $\alpha = 0.4$.
- $LRC(\alpha) = \{c \in LC \mid g(c) \leq 4 + 0.4 \times (16 - 4)\}$
- $LRC(\alpha) = \{c \in LC \mid g(c) \leq 8.6\}$
- $LRC(\alpha) = \{2, 7\}$

- Vamos supor que aleatoriamente $s \leftarrow s \cup \{2\}$.
- Atualiza $LC \leftarrow LC \setminus \{2\} = \{1, 4, 5, 6, 7\}$

Exemplo

TSP Simétrico (todas as ligações) de exemplo:

Construção e LRC por valor:

- Solução parcial $s = \{0, 3, 2\}$.
- $LC = \{1, 4, 5, 6, 7\}$
- $g(c) =$ distância do candidato $c \in LC$ à ultima cidade inserida (2).
- E assim por diante ...

- A construção termina quando $LC = \emptyset$.
- A cada aplicação da fase de construção do GRASP novas soluções parcialmente gulosas são geradas, dada a aleatoriedade do processo construtivo.

Considerações do GRASP

Considerações:

- Uma vantagem do GRASP é a facilidade com que pode ser implementada.
- Outra vantagem é a quantidade de parâmetros da sua versão mais básica:
  - *GRASPMax*: Número de iterações até o critério de parada.
  - α: O ajuste que controla o quão gulosa ou o quão aleatória fica a construção.
  - Esses parâmetros devem ser calibrados apropriadamente.
- O *GRASP* também pode ser complementado utilizando-se um **SA** ou **BT** na fase da busca local.

Considerações do GRASP

Considerações:

- Sobre os parâmetros, pode-se usar número de iterações sem melhora também como critério de parada.
- Sobre o α, pode-se usar as estratégias:
  - **Estático:** Um valor fixo entre algum intervalo pré-definido. Geralmente a escolha é próximo da representação gulosa ($\alpha \in [0.1, 0.3]$) para **MIN**, ou ($\alpha \in [0.7, 0.9]$) para **MAX**.
  - **Dinâmico:** A cada nova construção, altera-se o valor de α dinamicamente, dentro de uma distribuição uniforme $[0.5, 0.9]$ por exemplo para **MIN**.
  - **Adaptativo:** Nesta estratégia o valor do α é periodicamente atualizado de acordo com a qualidade das soluções parciais e a melhor solução atual. O nome da estratégia é GRASP reativo (vide seção 2.9.3 do livro do Talbi).

Exercício

1. Faça um *GRASP* para o TSP baseado no Algoritmo 4 da página 15:
   (a) Como critério de parada use um número fixo de iterações (*GRASPMax*).
   (b) Para α pode-se usar um valor estático.
   (c) Na fase de construção, utilize a criação da LRC por valor:

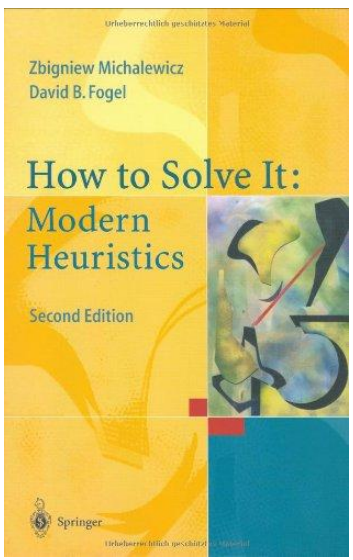
   $$LRC(\alpha) = \{c \in LC \mid g(c) \leq c_{min} + \alpha(c_{max} - c_{min})\}.$$

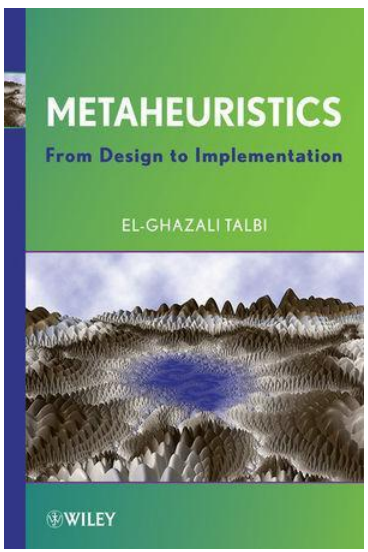
   Obs: Podemos usar a mesmas instâncias utilizadas naqueles exercícios anteriores.

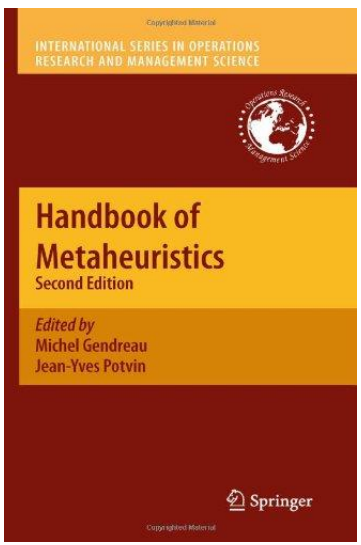
Bibliografias

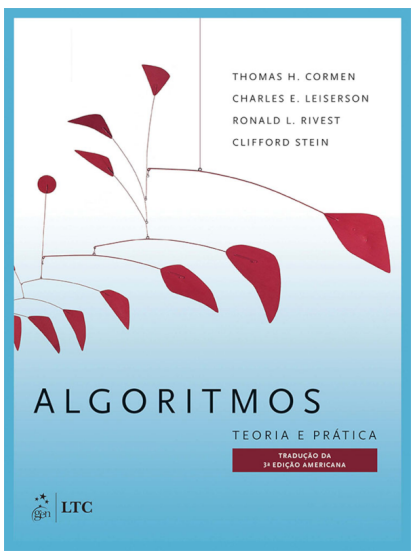
Bibliografia Básica

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.