Heurísticas e Metaheuristicas

Busca Exaustiva e Busca Local

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

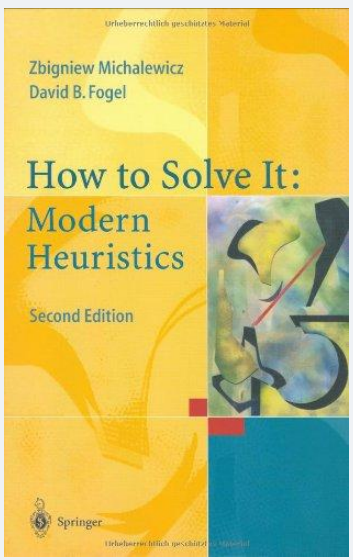
Departamento de Ciência da Computação
Universidade Federal de São João del-Rei

Material adaptado do Prof. André (UFV)

Referência

Livro

- Esse conteúdo está baseado no livro texto:
- MICHLEWICZ, Zbigniew; FOGEL, David B. How to solve it: modern heuristics. 2nd. ed. Berlin: Springer c2010 554 p. ISBN 9783642061349.
- Capítulos 2 e 3, seções 2.6, 3.1 - 3.2;



Métodos Tradicionais

Visão Geral:

- Dos métodos tradicionais existentes, alguns deles trabalham com **soluções parciais**:
  - Solução incompleta (apenas algumas variáveis definidas).
  - ou solução completa de um problema reduzido (mais simples).
  - Vantagem: mais fácil tratar o *subproblema*.
  - Desvantagem: saber decompor o problema e avaliar uma solução parcial.

Exemplos:

- Métodos com **solução parcial**:
  - SAT – fixar duas variáveis (ex.: X_1 e X_2 *true*) e resolver para o restante.
  - TSP – fixar uma sequência (ex.: $7 \rightarrow 11 \rightarrow 2 \rightarrow 16$) e permutar as demais
  - **Torcer para a solução ótima ter esses valores!**

Métodos Tradicionais

Visão Geral:

- Dos métodos tradicionais existentes, alguns deles trabalham com **soluções parciais**:
  - Solução incompleta (apenas algumas variáveis definidas).
  - ou solução completa de um problema reduzido (mais simples).
  - Vantagem: mais fácil tratar o *subproblema*.
  - Desvantagem: saber decompor o problema e avaliar uma solução parcial.

Exemplos:

- Métodos com **solução parcial**:
  - SAT – fixar duas variáveis (ex.: X_1 e X_2 *true*) e resolver para o restante.
  - TSP – fixar uma sequência (ex.: $7 \rightarrow 11 \rightarrow 2 \rightarrow 16$) e permutar as demais
  - **Torcer para a solução ótima ter esses valores!**

Métodos Tradicionais

Visão Geral:

- Dos métodos tradicionais existentes, alguns deles trabalham com **soluções parciais**:
  - Solução incompleta (apenas algumas variáveis definidas).
  - ou solução completa de um problema reduzido (mais simples).
  - Vantagem: mais fácil tratar o *subproblema*.
  - Desvantagem: saber decompor o problema e avaliar uma solução parcial.

Exemplos:

- Métodos que decompõem o problema (problema reduzido):
  - SAT – considerar k variáveis dentre as n originais e tentar resolver o problema menor.
  - TSP – considerar apenas k das n cidades e encontrar a menor rota por elas.
  - **Torcer para ser fácil juntar os resultados e retornar ao problema original!**

Métodos Tradicionais

Visão Geral:

- Outros trabalham com **soluções completas**:
  - Incluem todas as variáveis de decisão.
  - Pode-se facilmente comparar duas soluções.
  - Vantagem: já tem alguma solução em qualquer momento que parar a busca.

Exemplos:

- Métodos com solução completa:
  - Busca Exaustiva
  - Busca Local
  - Simplex
  - Metaheurísticas em geral

Métodos Tradicionais

Visão Geral:

- Outros trabalham com **soluções completas**:
  - Incluem todas as variáveis de decisão.
  - Pode-se facilmente comparar duas soluções.
  - Vantagem: já tem alguma solução em qualquer momento que parar a busca.

Exemplos:

- Métodos com solução completa:
  - Busca Exaustiva
  - Busca Local
  - Simplex
  - Metaheurísticas em geral

Agenda

1. Busca Exaustiva
   - Introdução
   - Exemplos
2. Busca Local
   - Introdução
   - Métodos de Busca Local

Introdução

Visão geral:

- A **busca exaustiva** ela é a estratégia que tenta verificar TODAS as soluções do espaço de soluções até encontrar a melhor.
- Podemos denominar também de **força bruta**.
- O único requisito é gerar todas as soluções de uma forma sistemática.
- A pergunta básica é:
  - Como implementar uma sequência que contém todas as possíveis soluções?
  - Note que a ordem não interessa, já que precisa gerar todas.
- A resposta é que: **Depende da representação!**

Observação interessante

- Existem métodos para alguns problemas de otimização que constroem soluções completas a partir de soluções parciais (*branch and bound* ou o algoritmo $A*$), que são baseados em busca exaustiva.

Introdução

Visão geral:

- A **busca exaustiva** ela é a estratégia que tenta verificar TODAS as soluções do espaço de soluções até encontrar a melhor.
- Podemos denominar também de **força bruta**.
- O único requisito é gerar todas as soluções de uma forma sistemática.
- A pergunta básica é:
  - Como implementar uma sequência que contém todas as possíveis soluções?
  - Note que a ordem não interessa, já que precisa gerar todas.
- A resposta é que: **Depende da representação!**

Observação interessante

- Existem métodos para alguns problemas de otimização que constroem soluções completas a partir de soluções parciais (*branch and bound* ou o algoritmo $A*$), que são baseados em busca exaustiva.

Introdução

Visão geral:

- A **busca exaustiva** ela é a estratégia que tenta verificar TODAS as soluções do espaço de soluções até encontrar a melhor.
- Podemos denominar também de **força bruta**.
- O único requisito é gerar todas as soluções de uma forma sistemática.
- A pergunta básica é:
  - Como implementar uma sequência que contém todas as possíveis soluções?
  - Note que a ordem não interessa, já que precisa gerar todas.
- A resposta é que: **Depende da representação!**

Observação interessante

- Existem métodos para alguns problemas de otimização que constroem soluções completas a partir de soluções parciais (*branch and bound* ou o algoritmo $A*$), que são baseados em busca exaustiva.

Introdução

Visão geral:

- A **busca exaustiva** ela é a estratégia que tenta verificar TODAS as soluções do espaço de soluções até encontrar a melhor.
- Podemos denominar também de **força bruta**.
- O único requisito é gerar todas as soluções de uma forma sistemática.
- A pergunta básica é:
  - Como implementar uma sequência que contém todas as possíveis soluções?
  - Note que a ordem não interessa, já que precisa gerar todas.
- A resposta é que: **Depende da representação!**

Observação interessante

- Existem métodos para alguns problemas de otimização que constroem soluções completas a partir de soluções parciais (*branch and bound* ou o algoritmo $A*$), que são baseados em busca exaustiva.

Exemplos

Exemplos:

- SAT: Enumerar todos os 2^n strings binários.
- TSP: Enumerar todas as $(n-1)!$ permutações.

Exemplos

SAT:

- Como enumerar todos os 2^n strings binários?
  - $< 0 \ldots 000 >$ até $< 1 \ldots 111 >$
- Algumas soluções:
  - Contar de 0 a $2^N - 1$ em decimal, transformando em binário
  - *Backtracking* (busca em profundidade numa árvore binária)
- Alguma leva vantagem?
  - Dependendo da abordagem, o *Backtracking* pode parar uma busca no meio do caminhamento (se testarmos que vai chegar em uma solução **false**).

- Suponha que a fórmula do SAT tenha o trecho: $\cdots \wedge (x_1 \vee \bar{x}_3 \vee x_4) \wedge \ldots$
- As atribuições: $x_1 = 0; x_2 = 1; x_3 = 1; x_4 = 0$ ou
  $x_1 = 0; x_2 = 0; x_3 = 1; x_4 = 0$
- Levam a uma resposta final FALSE independente do valor de outras variáveis, então o *Backtracking* pode buscar outro caminho.

Exemplos

SAT:

- Como enumerar todos os 2^n strings binários?
  - $< 0 \ldots 000 >$ até $< 1 \ldots 111 >$
- Algumas soluções:
  - Contar de 0 a $2^N - 1$ em decimal, transformando em binário
  - *Backtracking* (busca em profundidade numa árvore binária)
- Alguma leva vantagem?
  - Dependendo da abordagem, o *Backtracking* pode parar uma busca no meio do caminhamento (se testarmos que vai chegar em uma solução **false**).

- Suponha que a fórmula do SAT tenha o trecho: $\cdots \wedge (x_1 \vee \bar{x}_3 \vee x_4) \wedge \ldots$
- As atribuições: $x_1 = 0; x_2 = 1; x_3 = 1; x_4 = 0$ ou
  $x_1 = 0; x_2 = 0; x_3 = 1; x_4 = 0$
- Levam a uma resposta final FALSE independente do valor de outras variáveis, então o *Backtracking* pode buscar outro caminho.

Exemplos

SAT:

- Como enumerar todos os 2^n strings binários?
  - $< 0 \ldots 000 >$ até $< 1 \ldots 111 >$
- Algumas soluções:
  - Contar de 0 a $2^N - 1$ em decimal, transformando em binário
  - *Backtracking* (busca em profundidade numa árvore binária)
- Alguma leva vantagem?
  - Dependendo da abordagem, o *Backtracking* pode parar uma busca no meio do caminhamento (se testarmos que vai chegar em uma solução **false**).

- Suponha que a fórmula do SAT tenha o trecho: $\cdots \wedge (x_1 \vee \bar{x}_3 \vee x_4) \wedge \ldots$
- As atribuições: $x_1 = 0; x_2 = 1; x_3 = 1; x_4 = 0$ ou
  $x_1 = 0; x_2 = 0; x_3 = 1; x_4 = 0$
- Levam a uma resposta final FALSE independente do valor de outras variáveis, então o *Backtracking* pode buscar outro caminho.

Exemplos

SAT:

- Como enumerar todos os 2^n strings binários?
  - $< 0 \ldots 000 >$ até $< 1 \ldots 111 >$
- Algumas soluções:
  - Contar de 0 a $2^N - 1$ em decimal, transformando em binário
  - *Backtracking* (busca em profundidade numa árvore binária)
- Alguma leva vantagem?
  - Dependendo da abordagem, o *Backtracking* pode parar uma busca no meio do caminhamento (se testarmos que vai chegar em uma solução **false**).

- Suponha que a fórmula do SAT tenha o trecho: $\cdots \wedge (x_1 \vee \bar{x}_3 \vee x_4) \wedge \ldots$
- As atribuições: $x_1 = 0; x_2 = 1; x_3 = 1; x_4 = 0$ ou
  $x_1 = 0; x_2 = 0; x_3 = 1; x_4 = 0$
- Levam a uma resposta final FALSE independente do valor de outras variáveis, então o *Backtracking* pode buscar outro caminho.

Exemplos

TSP:

- Como enumerar todas as $(n-1)!$ permutações? Possíveis ideias?
  - Mapeamento de 0 a $(n-1)!$ em cada uma delas
  - *Backtracking* – procedimento recursivo que escolhe uma cidade por vez
  - Gerar aleatoriamente e marcar as que já foram usadas
  - Transformar uma permutação em outra (sem repetir)
  - Trocar elementos sucessivamente
- Alguma leva vantagem?
  - No TSP, o *Backtraking* pode parar a busca no meio do caminhamento, por exemplo se uma cidade não está ligada em outra por exemplo.

Agenda

1. Busca Exaustiva
   - Introdução
   - Exemplos
2. Busca Local
   - Introdução
   - Métodos de Busca Local

Introdução

Visão geral:

- A **busca local** significa concentrar a busca na vizinhança de alguma solução em particular, em vez de buscar exaustivamente no espaço de soluções inteiro.
- O passo a passo básico é:
  1. Escolher e avaliar uma solução do espaço de busca (sol. atual)
  2. Aplicar uma transformação na solução atual e avaliá-la (sol. nova)
  3. Se sol. nova é melhor que sol. atual, torna-se sol. atual (senão descarta)
  4. Repetir 2 e 3 até nenhuma transformação melhorar sol. atual

Introdução

Alguns cuidados na Busca Local:

- Se a transformação pode retornar qualquer solução do espaço de busca.
  - A solução atual não tem nenhum efeito na nova.
  - A busca se torna enumerativa (exaustiva).
  - Ou até pior que a exaustiva! Pois pode **repetir soluções**.
- Se a transformação retorna a própria solução
  - Não vai a lugar nenhum
  - A heurística pode entrar em *loop*!

Importante:

- É necessário encontrar um meio termo entre estes extremos
  - A transformação deve retornar **uma solução vizinha** (parecida/próxima com a atual).

Introdução

Alguns cuidados na Busca Local:

- Se a transformação pode retornar qualquer solução do espaço de busca.
  - A solução atual não tem nenhum efeito na nova.
  - A busca se torna enumerativa (exaustiva).
  - Ou até pior que a exaustiva! Pois pode **repetir soluções**.
- Se a transformação retorna a própria solução
  - Não vai a lugar nenhum
  - A heurística pode entrar em *loop*!

Importante:

- É necessário encontrar um meio termo entre estes extremos
  - A transformação deve retornar **uma solução vizinha** (parecida/próxima com a atual).

Introdução

Alguns cuidados na Busca Local:

- Em relação ao tamanho da vizinhança:
  - Se for pequena, podemos visitar todas soluções rapidamente, mas com o risco de ficar preso em um **ótimo local**.
  - Se for grande, é menor a chance de ficar preso mas a eficiência da busca é prejudicada.
- O tipo de transformação que vai determinar o tamanho da vizinhança
- Logo, a escolha do transformação deve ser feita de forma inteligente, levando em consideração o que sabemos da função de avaliação e da representação.

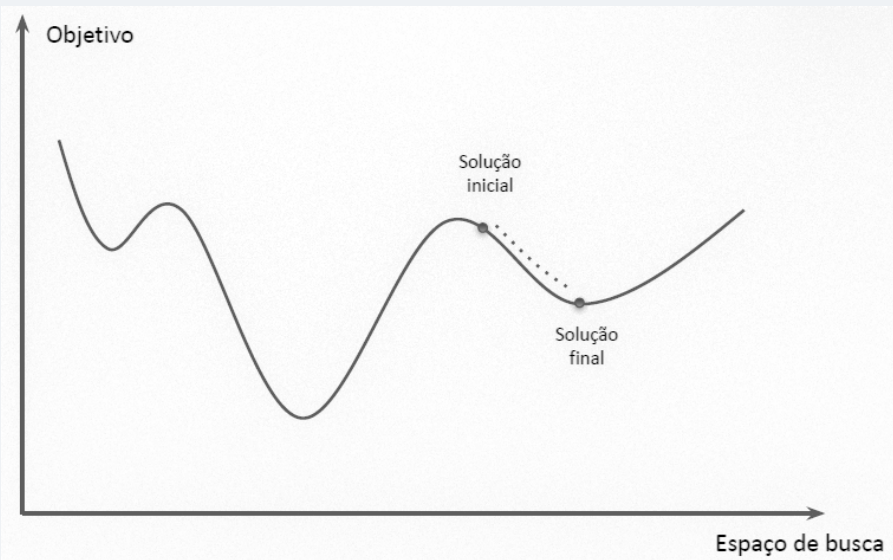
Introdução

Definições:

- A **busca local** também pode ser referida como:
  - *Hill-Climbing* (Subida da enconsta)
  - *Steepest descent* ou *descent* (Descida mais íngreme)
  - *Iterative Improvement* (Melhora iterativa)
- Considerada a mais antiga e a mais simples metaheurística.

Algorithm 1 Busca Local

$s_0 \leftarrow$ solução inicial
$s \leftarrow s_0$ (melhor solução atual)
while *Critério de parada não atingido* **do**
  Gerar $N(s)$ (vizinhança de s)
  if *Tem vizinho* s' *melhor que* s **then**
    $s \leftarrow s'$
  end
end
return s (melhor solução)



Métodos de Busca Local

Definição para o método (com diferentes nomes):

- O método *Hill-Climbing (Steepest descent, Iterative Improvement)* é uma busca local que melhora a solução iterativamente:
  - A estratégia é aplicada a um ponto do espaço (solução atual).
  - Seleciona um novo ponto na vizinhança.
  - Se for melhor que o atual, o novo ponto se torna o atual.
  - Senão, outro ponto na vizinhança é selecionado.
  - Termina quando não consegue mais melhorar.

Métodos de Busca Local

Definição para o método (com diferentes nomes):

- Claramente pode chegar apenas num **ótimo local**
- Não garante o ótimo global
  - E não há uma medida de erro (distância do ótimo global)
- Geralmente não se pode prever o tempo de execução
- O resultado **depende fortemente do ponto inicial**
- Uma opção é executar várias vezes, de várias pontos iniciais
  - Escolhidos aleatoriamente
  - Ou por uma distribuição no espaço de busca
  - Ou por algum outro método ou conhecimento prévio do problema

Métodos de Busca Local

Definição para o método (com diferentes nomes):

- Mas é relativamente fácil de aplicar!
- Tudo que precisa é:
  - Representação da solução
  - Função de avaliação
  - Medida (operador) que define a vizinhança

Algumas variações:

- Diferem-se basicamente na forma como a nova solução é selecionada:
  - Verificar a vizinhança inteira e retornar a melhor (se houver).
  - Verificar aleatoriamente uma certa quantidade de vizinhos.
  - Verificar sistematicamente e escolher a primeira que melhora.

Métodos de Busca Local

Considerações importantes:

- Técnicas eficientes de busca devem realizar um balanceamento entre a quantidade de esforço para tarefas de intensificação (***exploiting***) e exploração (***exploring***):
  - ***Exploiting*** consiste na intensificação das melhores soluções encontradas até o momento.
  - ***Exploring*** consiste na exploração mais global do espaço de busca.
- Técnicas de **busca local** são mais voltadas para o processo de ***exploiting***.
- Enquanto que uma busca puramente aleatória explora o espaço de buscas de maneira mais ampla (***exploring***) (assumindo que pontos são sorteados de maneira uniforme), porém não intensifica a busca em nenhuma região, ainda que promissora.

Pergunta aberta pelo livro:

- Como podemos desenvolver um algoritmo de busca que tem chance de fugir de ótimos locais, balancear ***Exploiting*** e ***Exploring*** e fazer a busca independente da solução inicial? (vamos ver mais pra frente).

Métodos de Busca Local

Considerações importantes:

- Técnicas eficientes de busca devem realizar um balanceamento entre a quantidade de esforço para tarefas de intensificação (***exploiting***) e exploração (***exploring***):
  - ***Exploiting*** consiste na intensificação das melhores soluções encontradas até o momento.
  - ***Exploring*** consiste na exploração mais global do espaço de busca.
- Técnicas de **busca local** são mais voltadas para o processo de ***exploiting***.
- Enquanto que uma busca puramente aleatória explora o espaço de buscas de maneira mais ampla (***exploring***) (assumindo que pontos são sorteados de maneira uniforme), porém não intensifica a busca em nenhuma região, ainda que promissora.

Pergunta aberta pelo livro:

- Como podemos desenvolver um algoritmo de busca que tem chance de fugir de ótimos locais, balancear ***Exploiting*** e ***Exploring*** e fazer a busca independente da solução inicial? (vamos ver mais pra frente).

Métodos de Busca Local

Voltando ao pseudocódigo..

Algorithm 2 Busca Local

$s_0 \leftarrow$ solução inicial
$s \leftarrow s_0$ (melhor solução atual)
while *Critério de parada não atingido* **do**
    Gerar $N(s)$ (vizinhança de s)
    if *Tem vizinho* s' *melhor que* s **then**
        $s \leftarrow s'$
    end
end
return s (melhor solução)

- Definimos a solução inicial s_0 e o operador de vizinhança $N(.)$.
- Algumas variantes incluem detalhar:
  - Ordem de geração das soluções da vizinhança.
  - **Estratégia de seleção da solução vizinha**.

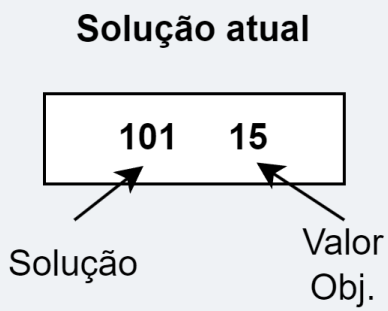
Métodos de Busca Local

Estratégia de seleção da solução vizinha

- Existem 3 estratégias principais de exploração da vizinhança para selecionar uma solução:
  - *Best improvement (steepest descent/ascent)* (**Melhor aprimoramento - Melhor melhora**)
  - *First improvement* (**Primeiro aprimoramento - Primeira melhora**)
  - *Random selection* (**Seleção aleatória**)

Métodos de Busca Local

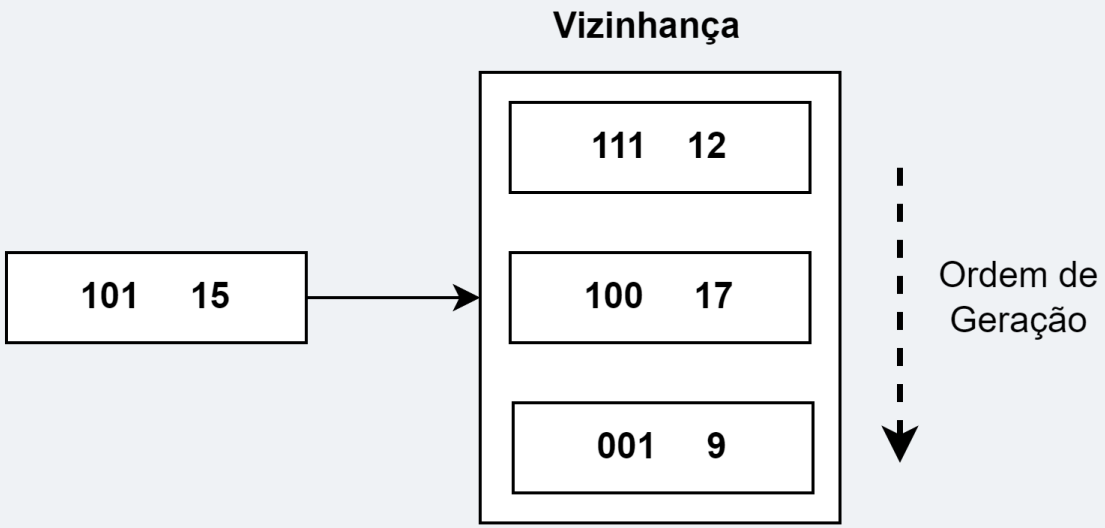
Estratégia de seleção da solução vizinha



Fonte: Talbi – Metaheuristics, Wiley 2009

Métodos de Busca Local

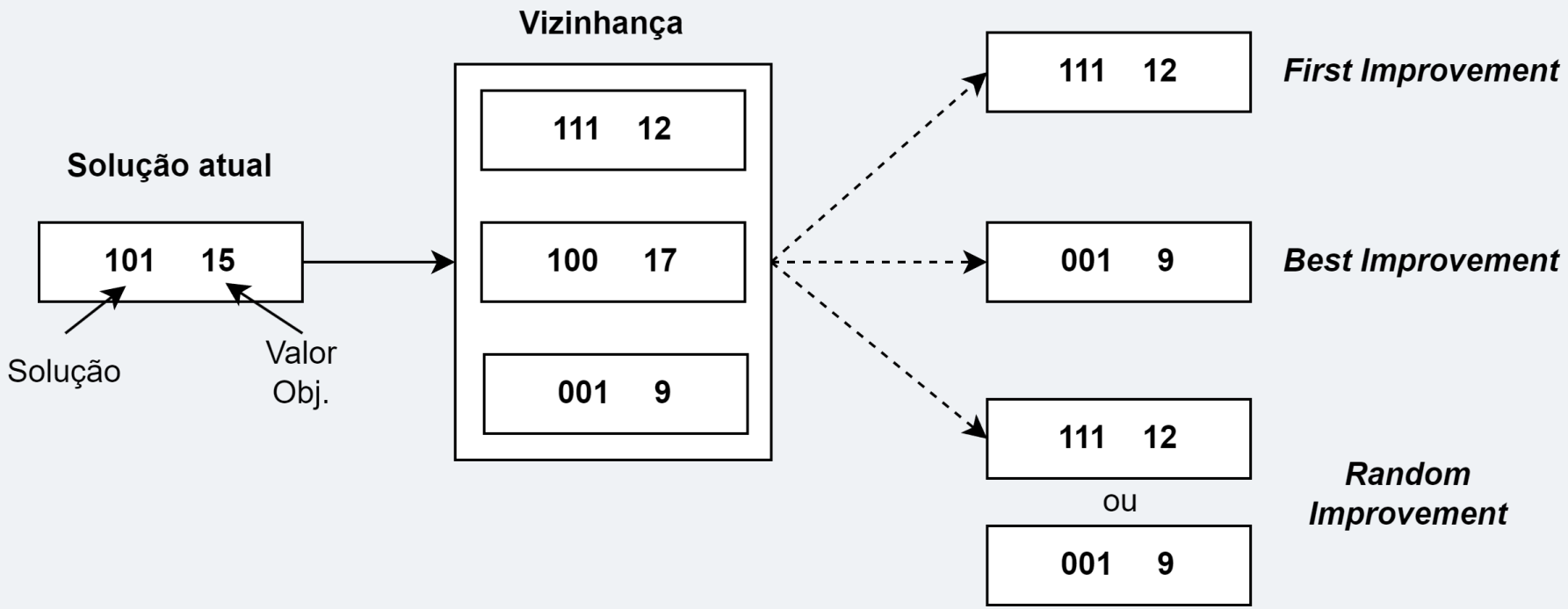
Estratégia de seleção da solução vizinha



Fonte: Talbi – Metaheuristics, Wiley 2009

Métodos de Busca Local

Estratégia de seleção da solução vizinha



Fonte: Talbi – Metaheuristics, Wiley 2009

Métodos de Busca Local

Estratégia de seleção da solução vizinha

- Qual é melhor em termos de **qualidade**?
- Qual é melhor em termos de **tempo de execução**?
- Fatores a considerar:
  - Qualidade
  - Velocidade
  - Tempo de busca
  - Convergência prematura
- Um bom equilíbrio entre qualidade da solução e tempo de busca:
  - Solução inicial qualquer (geração aleatória): ***first improvement***
  - Solução inicial boa (geração por heurística): ***best improvement***

Métodos de Busca Local

Estratégia de seleção da solução vizinha

- Qual é melhor em termos de **qualidade**?
- Qual é melhor em termos de **tempo de execução**?
- Fatores a considerar:
  - Qualidade
  - Velocidade
  - Tempo de busca
  - Convergência prematura
- Um bom equilíbrio entre qualidade da solução e tempo de busca:
  - Solução inicial qualquer (geração aleatória): ***first improvement***
  - Solução inicial boa (geração por heurística): ***best improvement***

Métodos de Busca Local

Estratégia de seleção da solução vizinha

- Qual é melhor em termos de **qualidade**?
- Qual é melhor em termos de **tempo de execução**?
- Fatores a considerar:
  - Qualidade
  - Velocidade
  - Tempo de busca
  - Convergência prematura
- Um bom equilíbrio entre qualidade da solução e tempo de busca:
  - Solução inicial qualquer (geração aleatória): ***first improvement***
  - Solução inicial boa (geração por heurística): ***best improvement***

Métodos de Busca Local

Estratégia de seleção da solução vizinha

- Vantagens:
  - Relativamente fácil de projetar
  - Relativamente fácil de implementar
  - Encontra soluções razoavelmente boas muito rapidamente
- Desvantagens:
  - Muito sensível à solução inicial
  - Não há uma estimativa de erro
  - Não há estimativa do número de iterações
  - Converge a um **ótimo local**

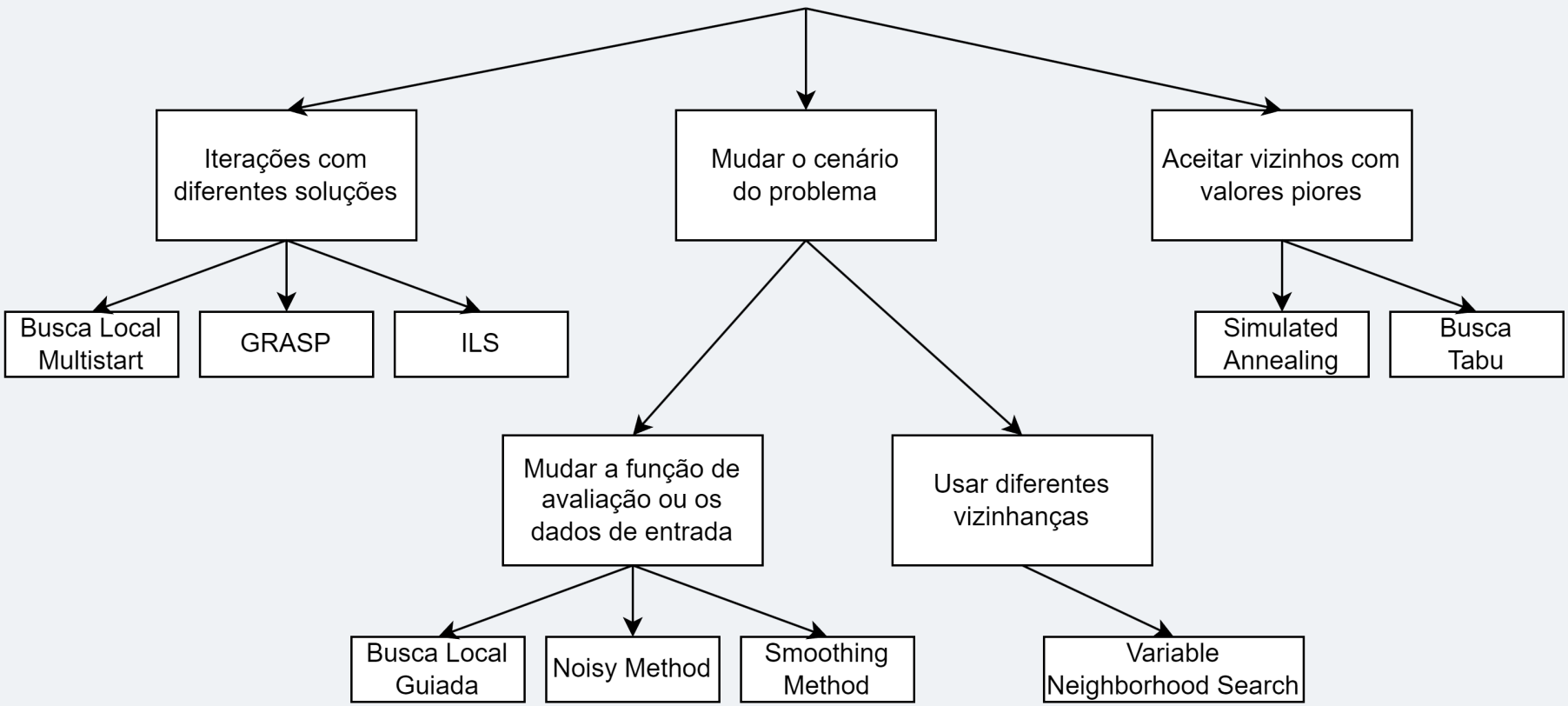
Métodos de Busca Local

Estratégia de seleção da solução vizinha

- Vantagens:
  - Relativamente fácil de projetar
  - Relativamente fácil de implementar
  - Encontra soluções razoavelmente boas muito rapidamente
- Desvantagens:
  - Muito sensível à solução inicial
  - Não há uma estimativa de erro
  - Não há estimativa do número de iterações
  - Converge a um **ótimo local**

Métodos de Busca Local

Estratégias para escapar de ótimo local



Métodos de Busca Local (Exemplos)

Procedimento *GSAT* para o SAT:

Algorithm 3 Greedy SAT

```
forall the i ∈ MAX_TENTATIVAS do
    T ← string_aleatoria_de_valores_booleanos
    forall the j ∈ MAX_TROCAS do
        if f(T) = TRUE then
            return T
        end
        else
            p ← varivel_que_trocada_maximiza_o_numero_de_clausulas_satisfeitas_de_f(T)
            T ← flip(T, p)
        end
    end
end
return Solucao_Nao_Encontrada
```

Métodos de Busca Local (Exemplos)

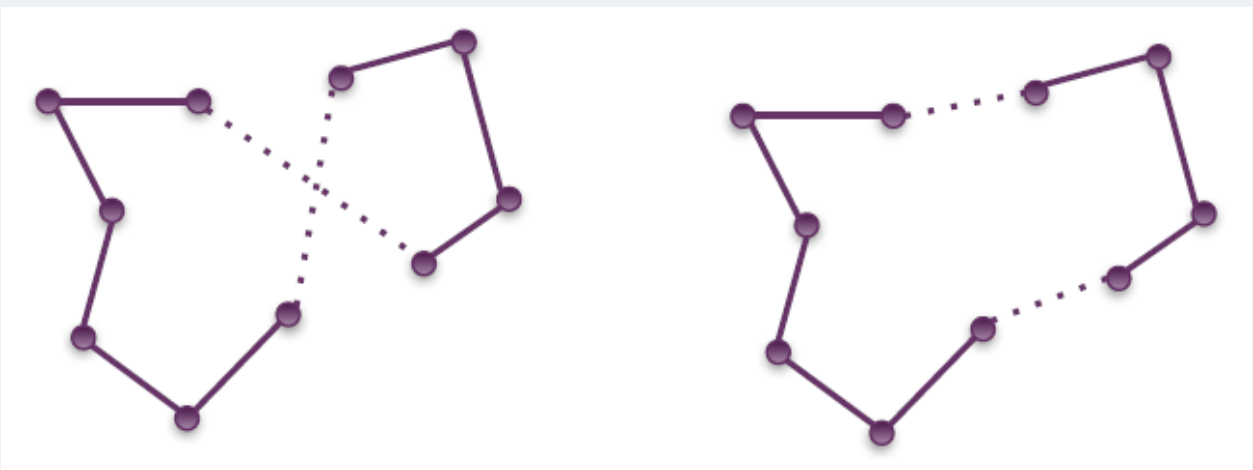
Procedimento *GSAT* para o SAT:

- Método de busca local guloso proposto por Selman, Levesque e Mitchel (1992).
- Inicia com uma cadeia aleatória de valores booleanos.
- Prossegue então por trocar (flip) o valor booleano da variável tal que produza o maior aumento no número total de cláusulas satisfeitas.
- As trocas são feitas até que se encontre uma atribuição verdadeira para a função, ou que alcance o número máximo de trocas possíveis.
- Note que para cada tentativa, uma nova solução aleatória é gerada. E tenta explorar a vizinhança desta solução.

Métodos de Busca Local (Exemplos)

Procedimento 2-opt para o TSP

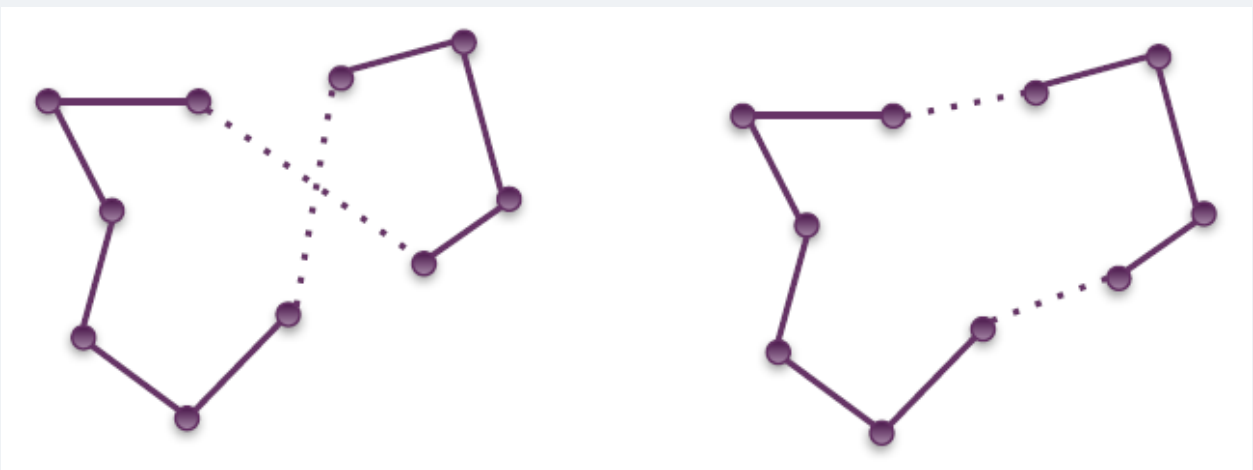
- Começar com uma rota T qualquer
- A vizinhança de T contém todos as rotas que podem ser obtidas trocando-se 2 arcos de T (movimento *2-exchange (2-swap)*).
- Uma nova rota T' substitui T se for melhor
- Se nenhuma rota vizinha de T for melhor, o algoritmo termina com T (chamada rota *2-optimal*).



Métodos de Busca Local (Exemplos)

Procedimento 2-opt para o TSP

- Começar com uma rota T qualquer
- A vizinhança de T contém todos as rotas que podem ser obtidas trocando-se 2 arcos de T (movimento *2-exchange (2-swap)*).
- Uma nova rota T' substitui T se for melhor
- Se nenhuma rota vizinha de T for melhor, o algoritmo termina com T (chamada rota *2-optimal*).



Métodos de Busca Local (Exemplos)

Procedimento 2-opt para o TSP

- Pode ser generalizado para k-opt.
- Nesse caso k arcos (ou até k arcos) devem ser removidos e substituídos
- k pequeno:
  - permite enumerar a vizinhança inteira
  - mas aumenta a chance de ótimo local.
- k grande:
  - vizinhança muito grande (de fato exponencial com k)
- Raramente se usa $k > 3$.

Exercícios

1. (**Busca Exaustiva**) Dado um valor inteiro n, criar um programa que gere/imprime todas as strings binárias de tamanho n. Como se fosse uma busca exaustiva para testarmos em um **SAT** ou **Mochila 0/1**.
2. (**Busca Exaustiva**) **a)** Dado um valor inteiro n, criar um programa que gere/imprime todas as permutações que contém os valores de 1 a n. Como se fosse uma busca exaustiva para testarmos em um **TSP**, onde o n representa o número de cidades.

   b) Seguindo a ideia das permutações na busca exaustiva, agora vamos aplicar para um grafo de cidades, usando os códigos de grafos que fizemos. Nesse caso, crie um algoritmo de backtracking que para a construção da permutação caso não exista uma aresta entre duas cidades.

- **OBS:** Para ambos, inclua um clock do sistema para vermos o tempo de processamento à medida que os valores/dados se alteram. Veja um exemplo de uso do clock em C++ no SIGAA.

Exercícios

3. (**Busca Local**) (Problema da Mochila 01) Crie um algoritmo de busca local para um problema da mochila 01. Ele vai utilizar a representação de **string binária** para uma solução e o conceito do ***flip*** como operador de vizinhança, conforme visto na aula (Significa mudar o valor de um dos itens da mochila de 0 pra 1 ou vice-versa).

   Lembre-se que um algoritmo de busca local precisa de:

   - uma solução inicial
   - uma função de avaliação para quantificar uma solução
   - o operador de vizinhança (ex: *flip*)
   - um critério de parada (ex: número de iterações)
   - e a política de melhoria (***First ou Best Improvement***).

   Entradas para o algoritmo estão nos arquivos complementares dessa aula, leia o ***Read-me***.

Exercícios

4. (**Busca Local**) (Problema do Caixeiro Viajante - *TSP*) Crie um algoritmo de busca local para o TSP. Ele vai utilizar a representação de vetor para uma solução e o conceito do ***2-opt*** como operador de vizinhança, conforme visto na aula (Significa trocar duas cidades de posição no vetor).

   Lembre-se que um algoritmo de busca local precisa de:

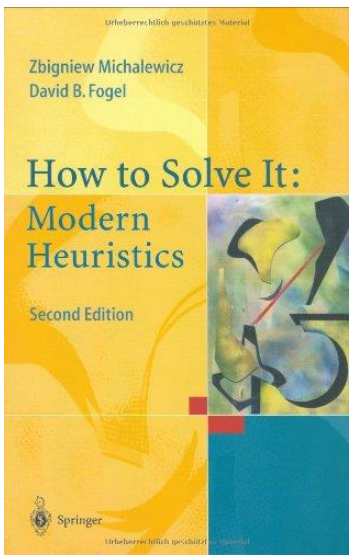
   - uma solução inicial
   - uma função de avaliação para quantificar uma solução
   - o operador de vizinhança (ex: *2-opt*)
   - um critério de parada (ex: número de iterações)
   - e a política de melhoria (***First ou Best Improvement***).

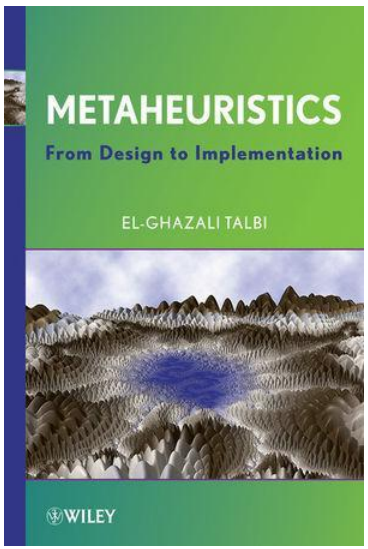
   Entradas para o algoritmo estão nos arquivos complementares dessa aula, leia o ***Read-me***.

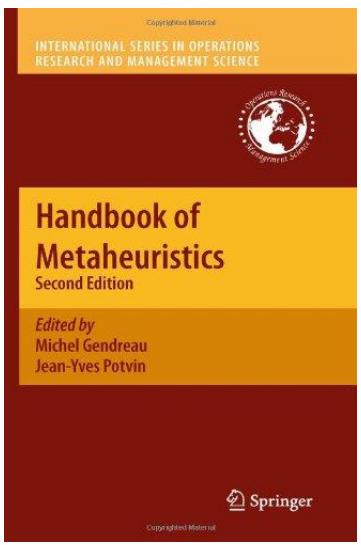
Bibliografias

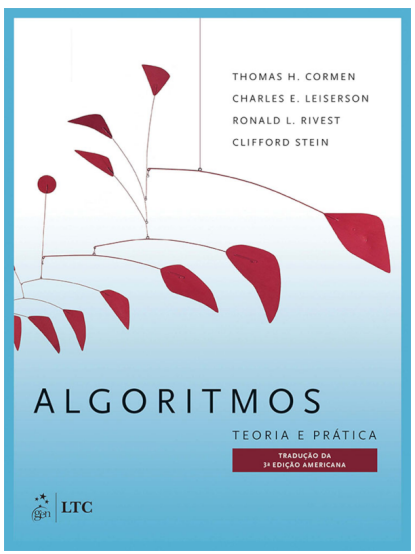
Bibliografia Básica

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.