VNS e VND (Variable Neighborhood Search e Descent)

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

1. VNS e VND
   - Introdução
   - VND
2. VNS
   - VNS
   - Considerações

Introdução

Visão geral:

- O VNS (*Variable Neighborhood Search*) ou busca em vizinhanças variadas, como o nome diz, tenta explorar sucessivamente um conjunto pré-definido de vizinhanças para encontrar uma solução melhor.
- A exploração usando o conjunto pode ser aleatória ou sistemática para obter diferentes ótimos locais e escapar de ótimos locais.
- O VNS explora o fato de que *usando várias vizinhanças na busca local, pode-se gerar diferentes ótimos locais.*
- E principalmente que, um **ótimo global** *corresponde a um ótimo local para todas as estruturas de vizinhança.*

Introdução

Visão geral:

- O VNS foi baseado no VND (*Variable Neighborhood Descent*), que é uma versão determinística do VNS.
- O VND utiliza vizinhanças sucessivas em descida até um ótimo local.
- Para entendermos, primeiro define-se um conjunto de vizinhanças: $N_l(l = 1, 2, \ldots, l_{max})$.
- Seja N_1 a primeira vizinhança e s uma solução inicial.
- Se nenhuma melhoria acontece na vizinhança $N_l(s)$, então a estrutura de vizinhança muda de N_l para N_{l+1}, ou seja, para a próxima vizinhança no conjunto.
- Se uma melhoria acontece, o método retorna à vizinhança N_1 com a nova solução corrente.

Introdução

Pseudo-código:

- O VND básico supondo um problema de **Minimização**:
- *VMax* define o número de estruturas de vizinhança existentes.

Algorithm 1: Variable Neighborhood Descent

```
1  s ← s_0 (Solução inicial)
2  k ← 1
3  while k ≤ VMax do
4  |   Encontre o melhor vizinho s' ∈ N_k(s)
5  |   if f(s') < f(s) then
6  |   |   s ← s'
7  |   |   k ← 1
8  |   end
9  |   else
10 |   |   k ← k + 1
11 |   end
12 end
13 return s
```

- A linha 4 atualmente com *Best Improvement*, pode ser substituída para *First Improvement*: $s' \leftarrow \text{PrimeiraMelhora}(s, k)$.

Introdução

Pseudo-código:

- Em uma versão mais genérica, o mesmo algoritmo pode ser reescrito assim:

Algorithm 2: Variable Neighborhood Descent (Genérico)

```
1 s ← s_0 (Solução inicial)
2 k ← 1
3 while k ≤ VMax do
4 |   s' ← MelhorVizinho(s, k)
5 |   AlteraVizinhança(s, s', k)
6 end
7 return s
```

- A função *AlteraVizinhança* seria exatamente o *if else* que compara as soluções e altera o valor de k de acordo com a comparação.

Introdução

Pseudo-código:

- A versão mais recente divulga pelos autores do VND é assim:

Algorithm 3: Variable Neighborhood Descent (Genérico mais recente)

```
1  s ← s_0 (Solução inicial)
2  repeat
3  |   k ← 1
4  |   PARA ← false
5  |   s_best ← s (Copia Solução Atual, melhor até então)
6  |   while k ≤ VMax do
7  |   |   s' ← MelhorVizinho(s, k)
8  |   |   AlteraVizinhança(s, s', k)
9  |   end
10 |   if f(s) ≥ f(s_best)  (True se não encontrou solução melhor) then
11 |   |   PARA ← true
12 |   end
13 until PARA = true;
14 return s_best
```

- A função *AlteraVizinhança* nesse caso pode assumir uma das três opções: Sequencial, Pipe ou Cíclica.

AlteraVizinhança

AlteraVizinhança(s, s', k) (Sequencial):

- Se houver melhora na solução atual, **retorna-se à primeira vizinhança**; caso contrário, passa-se para a vizinhança seguinte.

Algorithm 4: AlteraVizinhança(s, s', k) (Sequencial)

```
1 if f(s') < f(s) then
2 |   s ← s'
3 |   k ← 1
4 end
5 else
6 |   k ← k + 1
7 end
8 return s, k
```

AlteraVizinhança

AlteraVizinhança(s, s', k) (Pipe):

- Se houver melhora em uma vizinhança, **permanece-se nela**; caso contrário, passa-se para a vizinhança seguinte.

Algorithm 5: AlteraVizinhança(s, s', k) (Pipe)

```
1 if f(s') < f(s) then
2 |   s ← s'
3 end
4 else
5 |   k ← k + 1
6 end
7 return s, k
```

AlteraVizinhança

AlteraVizinhança(s, s', k) (Cíclica):

- Passa-se para a próxima vizinhança, **independentemente** de melhora na solução atual.

Algorithm 6: AlteraVizinhança(s, s', k) (Cíclica)

```
1 k ← k + 1
2 if f(s') < f(s) then
3 |   s ← s'
4 end
5 return s, k
```

Variantes do VND

Variantes do VND

- B-VND (VND Básico): Que usa o procedimento *Sequencial*.
- P-VND (VND com Viz. Pipe): Que usa o procedimento *Pipe*.
- C-VND (VND com Viz. Cíclica): Que usa o procedimento *Cíclica*.

R-VND

- R-VND (VND with Random neighborhood): a cada chamada do método VND, as vizinhanças são ordenadas aleatoriamente.
- Suponha que as vizinhanças de um dado problema são $N = \{N_1, N_2, N_3, N_4\}$.
- Ao aplicar o RVND, essas vizinhanças são colocadas em uma ordem aleatória, por exemplo: N_3, N_1, N_4, N_2.
- Neste caso, a primeira vizinhança a ser explorada será N_3; a próxima N_1; e assim por diante.
- *Uma vantagem do RVND, é que não é necessário calibrar a ordem das vizinhanças, como requerido nas outras variantes.*

Variantes do VND

Variantes do VND

- B-VND (VND Básico): Que usa o procedimento *Sequencial*.
- P-VND (VND com Viz. Pipe): Que usa o procedimento *Pipe*.
- C-VND (VND com Viz. Cíclica): Que usa o procedimento *Cíclica*.

R-VND

- R-VND (VND with Random neighborhood): a cada chamada do método VND, as vizinhanças são ordenadas aleatoriamente.
- Suponha que as vizinhanças de um dado problema são $N = \{N_1, N_2, N_3, N_4\}$.
- Ao aplicar o RVND, essas vizinhanças são colocadas em uma ordem aleatória, por exemplo: N_3, N_1, N_4, N_2.
- Neste caso, a primeira vizinhança a ser explorada será N_3; a próxima N_1; e assim por diante.
- *Uma vantagem do RVND, é que não é necessário calibrar a ordem das vizinhanças, como requerido nas outras variantes.*

Agenda

1. VNS e VND
   - Introdução
   - VND
2. VNS
   - VNS
   - Considerações

Introdução

Visão geral:

- O VNS é uma metaheurística de busca local que explora o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança.
- Na sua ideia, ele explora vizinhanças gradativamente mais "distantes".
- Possui quatro componentes principais:
  - Gerador de solução inicial.
  - Procedimento de perturbação (*shaking*): gera uma perturbação na k-ésima vizinhança da solução atual.
  - Procedimento de busca local, que pode ser também o VND: refina a solução atual.
  - Procedimento de troca da vizinhança: define qual a próxima vizinhança a ser explorada.

Introdução

Pseudo-código:

- A versão mais genérica do VNS é descrita assim:

Algorithm 7: Variable Neighborhood Search (Genérico)

```
1  s ← s_0 (Solução inicial)
2  while condição parada não satisfeita do
3  |   k ← 1
4  |   while k ≤ VMax do
5  |   |   s' ← Perturbação(s, k)
6  |   |   s'' ← BuscaLocal(s')
7  |   |   AlteraVizinhança(s, s'', k)
8  |   end
9  end
10 return s
```

- Na função *Perturbação*, o objetivo seria pegar uma solução aleatória da k-ésima vizinhança $N_k(s)$. O próximo slide mostra o pseudo-código dessa função.
- A função AlteraVizinhança pode variar entre as descritas anteriormente, mas a Sequencial é a básica.

Perturbação

Perturbação(s, k):

- Seja m_k um movimento na vizinhança N_k.
- Seja x uma representação aleatória de uma distribuição uniforme conforme o problema abordado com n elementos.

Algorithm 8: Perturbação(s, k)

1 $x \leftarrow$ valor aleatório $\in U[1, n]$
2 $s' \leftarrow s \oplus m_k(x)$
3 **return** s'

- Da forma que está descrito pelo livro, a perturbação pode receber soluções piores, mas o objetivo aqui é fugir de ótimos locais.
- O próximo passo é aplicar a busca local.

Variantes do VNS

Variantes do VNS

- Algumas variantes do VNS também são descritas:
  - VNS (Basic VNS): Busca local do VNS feita por um método de busca local convencional.
  - GVNS (General VNS): Busca local do VNS feita por um VND.
  - RVNS (Reduced VNS): Não tem busca local.
  - SVNS (Skewed VNS): Aceita soluções de piora que distam da solução atual por um determinado valor.
  - Smart VNS: Segue as mesmas ideias do *Smart ILS* que vimos.

Variantes do VNS

GVNS (General VNS):

- Busca local convencional é substituída por um VND.

Algorithm 9: GVNS

```
1  s ← s_0 (Solução inicial)
2  while condição parada não satisfeita do
3  |   k ← 1
4  |   while k ≤ VMax do
5  |   |   s' ← Perturbação(s, k)
6  |   |   s'' ← VND(s')
7  |   |   AlteraVizinhança(s, s'', k)
8  |   end
9  end
10 return s
```

- Condição de parada: Número máximo de iterações sem melhora, tempo de processamento, etc.

Variantes do VNS

RVNS (Reduced VNS):

- Não possui a etapa de busca local.
- Seleciona um ponto aleatório na k-ésima vizinhança, começando pela primeira. Se houver melhora, move-se para esse ponto e retorna-se à primeira vizinhança; caso contrário, avança-se para a próxima vizinhança.
- Termina ao alcançar a última vizinhança e satisfazer o critério de parada.

Algorithm 10: RVNS

```
1  s ← s_0 (Solução inicial)
2  while condição parada não satisfeita do
3  |   k ← 1
4  |   while k ≤ VMax do
5  |   |   s' ← Perturbação(s, k)
6  |   |   ~~s'' ← BuscaLocal(s')~~
7  |   |   AlteraVizinhança(s, s'', k)
8  |   end
9  end
10 return s
```

- Condição de parada: Número máximo de iterações sem melhora, tempo de processamento, etc.

Variantes do VNS

Smart VNS:

- Segue as mesmas ideias do *Smart ILS* que vimos.
- Número de tentativas (t) por pertubação, até um máximo ($TMax$).

Algorithm 11: Smart VNS

```
1  s ← s_0 (Solução inicial)
2  while condição parada não satisfeita do
3  |   k ← 1; t ← 1
4  |   while k ≤ VMax do
5  |   |   s' ← Perturbação(s, k, t)
6  |   |   s'' ← BuscaLocal(s')
7  |   |   AlteraVizinhança(s, s'', k, t, TMax)
8  |   end
9  end
10 return s
```

Variantes do VNS

Smart VNS:

- Pseudo-código do procedimento *AlteraVizinhança*:

Algorithm 12: AlteraVizinhança($s, s'', k, t, TMax$)

```
1  if f(s'') < f(s) then
2  |   s ← s'; k ← 1; t ← 1
3  end
4  else
5  |   if t ≥ TMax then
6  |   |   k ← k + 1; t ← 1
7  |   end
8  |   else
9  |   |   t ← t + 1
10 |   end
11 end
12 return s, k, t
```

Considerações do VNS e VND

Considerações:

- Sobre a ordem das vizinhanças:
  - Não é obrigatório mas o VNS usualmente aplica vizinhanças aninhadas, de forma que: cada vizinhança $N_k(s)$ contém a anterior $N_{k-1}(s)$

    $N_1(s) \subset N_2(s) \subset \cdots \subset N_k(s), \forall s \in S$

  - No VND, o uso mais popular é deixar as vizinhanças na ordem de complexidade.
  - Ex. Roteamento de Veículos (mais caminhões): shift, swap na rota, swap entre diferentes rotas, etc.

Considerações do VNS e VND

Considerações:

- Sobre Intensificação × Diversificação:
  - Como o livro cita, mais esforço empregado na etapa de *busca local* significa mais **Intensificação**.
  - E mais esforço empregado na etapa de *perturbação* significa mais **Diversificação**.

Exercício

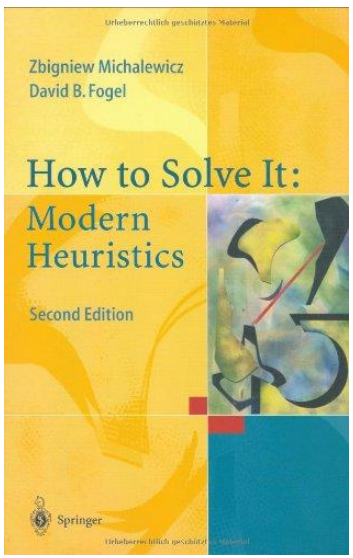
1. Faça um *VNS* para o TSP baseado no Algoritmo 7 da página 14:
   (a) Como critério de parada use um número de iterações (*VNSMax*) sem melhora.
   (b) Pode-se usar uma busca local comum.
   (c) Para a função *AlteraVizinhança*, pode-se usar a alteração **Sequencial**.

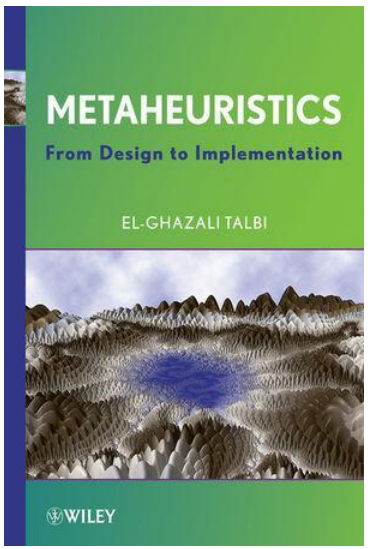
   Obs: Podemos usar a mesmas instâncias utilizadas naqueles exercícios anteriores.

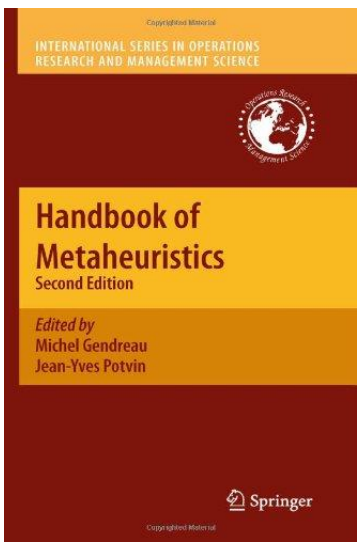
Bibliografias

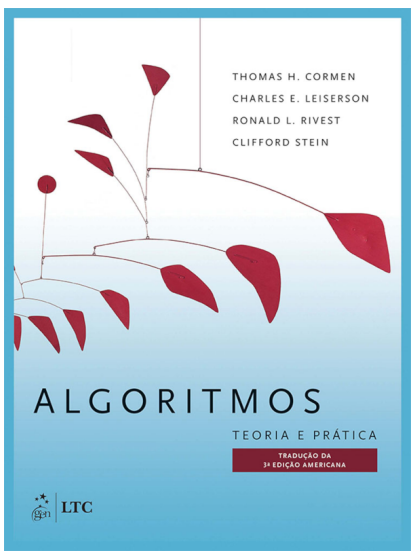
Bibliografia Básica

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.