BUSCA EM VIZINHANÇA VARIÁVEL

(VNS – VARIABLE NEIGHBORHOOD SEARCH)

Francisco A. M. Gomes

1° sem/2009

MT852 – Tópicos em pesquisa operacional

Variable neighborhood search (VNS)

- Método proposto por Mladenovic & Hansen em 1997, no qual
  - Efetua-se uma sequência de buscas locais.
  - Troca-se a estrutura de vizinhança durante a busca.
  - Explora-se vizinhanças gradativamente maiores.

Princípios básicos

- Um ótimo local com relação a uma vizinhança não necessariamente corresponde a um ótimo com relação a outra vizinhança.
- Um ótimo global corresponde a um ótimo local para todas as estruturas de vizinhança.
- Frequentemente, ótimos locais relativos a estruturas de vizinhança semelhantes estão relativamente próximos.

Variable neighborhood descent (VND)

- Sejam dados:
  - uma solução inicial x; e
  - um conjunto de vizinhanças factíveis $N_k(x)$, $k = 1, ..., k_{max}$.
- Vamos supor que, para cada vizinhança $N_k(x)$, disponhamos de um método de busca local que tenta encontrar uma solução melhor que x em $N_k(x)$.
- O VND varia a vizinhança de x na ordem estipulada: $N_1(x)$, $N_2(x)$, ...
- Em uma vizinhança $N_k(x)$,
  - se a busca local encontra x' melhor que x, substituímos x por x' e voltamos à vizinhança N_1.
  - Caso contrário, passamos à vizinhança $N_{k+1}(x)$.

VND - Algoritmo

1 Determinar uma solução inicial x;
2 $k \leftarrow 1$;
3 Enquanto ($k \leq k_{max}$),
3.1 Encontrar o melhor vizinho $x' \in N_k(x)$;
3.2 Se ($f(x') < f(x)$),
3.2.1 então $x \leftarrow x'$; $k \leftarrow 1$;
3.2.2 senão $k \leftarrow k + 1$;
3.3 Fim-Se;
4 Fim-Enquanto;
5 Retornar x;

Características das vizinhanças

- As vizinhanças devem ser ordenadas de modo que primeiras sejam as menores, ou seja, que as primeiras envolvam movimentos mais fáceis que as últimas.
- Assim, geralmente, o gasto computacional cresce quando mudamos de vizinhança.
- Na prática, em cada vizinhança $N_k(x)$, podemos encontrar
  - o melhor ponto, x^*; (*best improvement*)
  - o primeiro ponto x' tal que $f(x') < f(x)$. (*first improvement*)

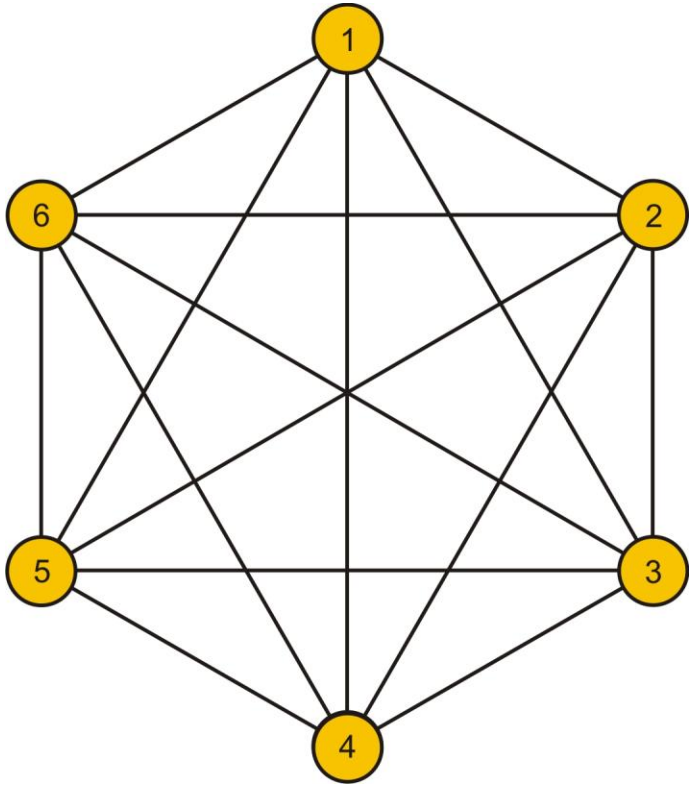
Problema do caixeiro viajante (TSP)

Dados:

- $V = \{v_1, ..., v_m\}$ conjunto de cidades (nós)
- $A = \{(r, s) \mid r, s \in V\}$ conjunto de arestas (caminhos entre cidades)
- c_{rs} = custo associado à aresta (r, s)

Propósito:

- O caixeiro deve visitar todas as cidades
- Passando uma só vez em cada cidade
- Com o menor custo possível



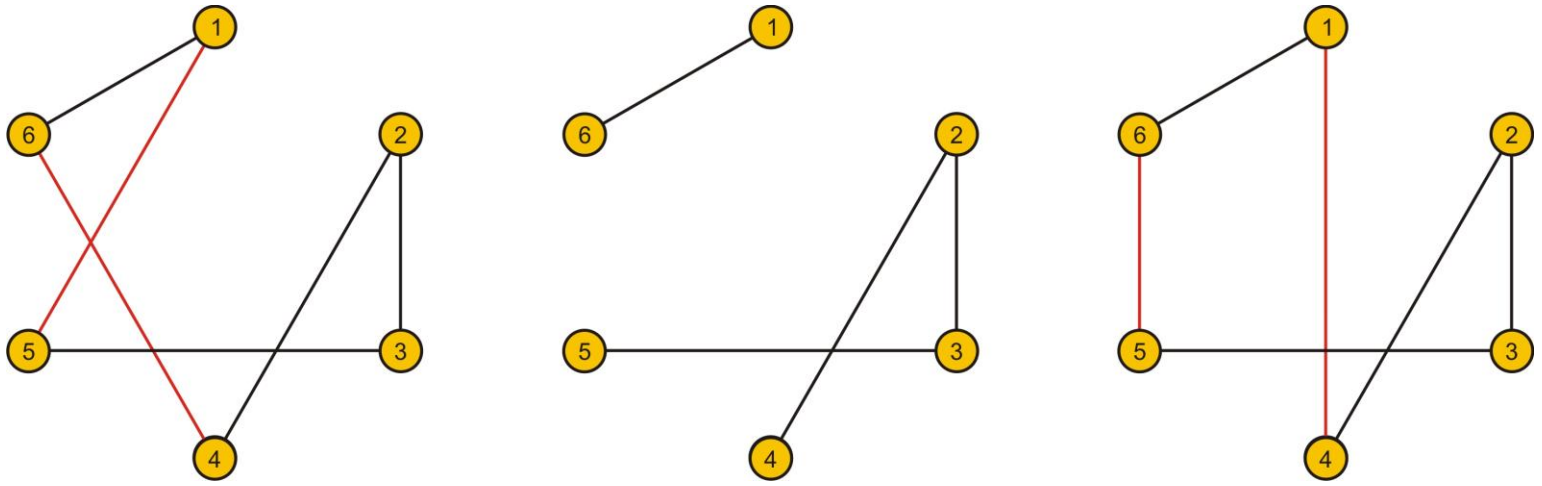
TSP simétrico

Vizinhança para o TSP

- Exemplos de vizinhança para o TSP:
  - $N_1(x)$ = Vizinhança baseada no 2-opt.
  - $N_2(x)$ = Vizinhança baseada no Or-opt.
  - $N_3(x)$ = Vizinhança baseada no 3-opt.
  - $N_4(x)$ = Vizinhança baseada no CROSS (4-opt)

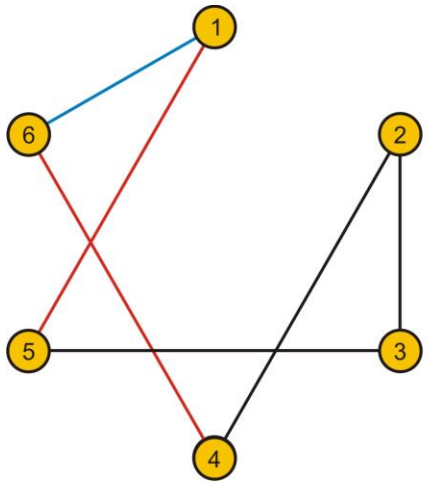
2-opt

- Retira 2 arcos do ciclo.
- Em um ciclo com n nós:
  - há C(n,2) modos de retirar 2 arcos;
  - há uma única maneira de reconectar o ciclo.

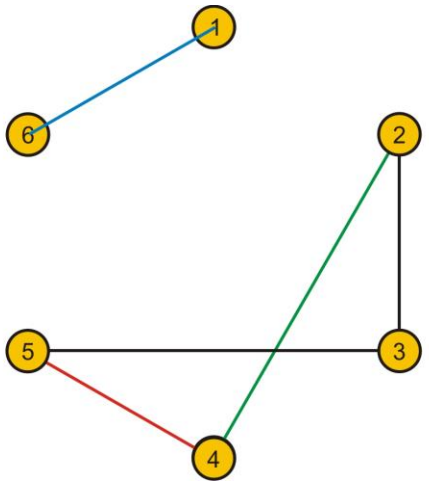


Or-opt

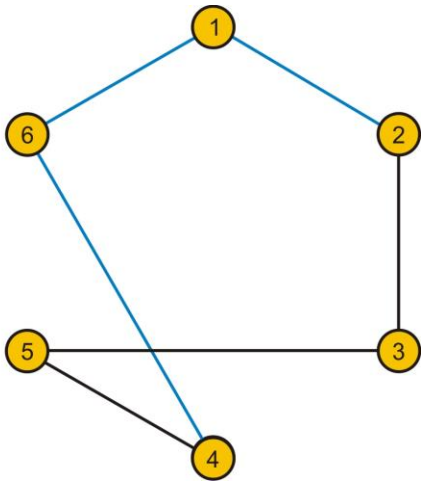
- Retira um trecho com t nós consecutivos.
- Recoloca esse trecho entre dois outros nós sucessivos do caminho que restou do ciclo.



Nós selecionados: 1 e 6.
Arcos por retirar:
(5,1) e (6,4)



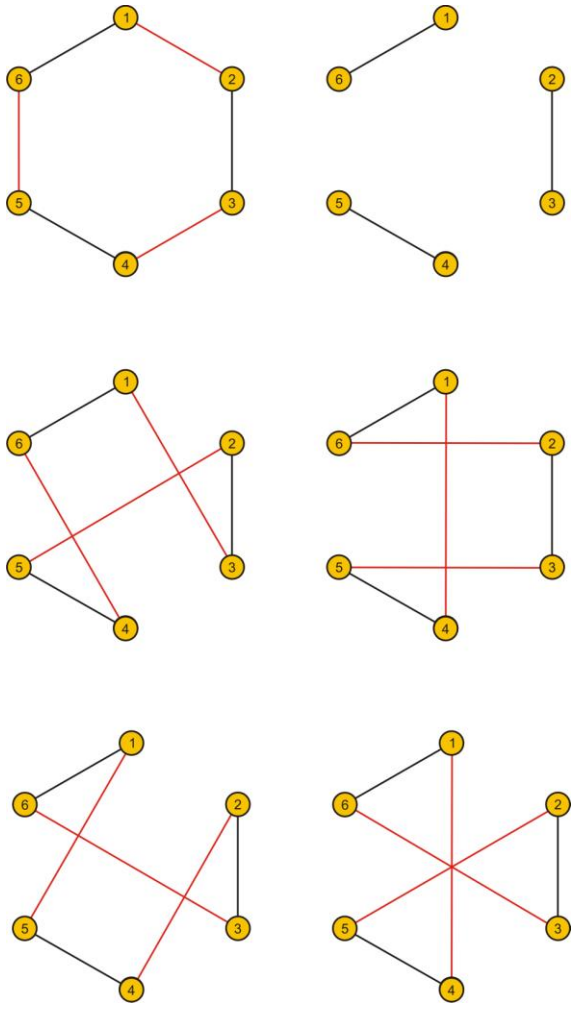
Arco introduzido: (5,4)
Ponto de inserção do arco
(1,6): entre os nós 2 e 4



Arcos introduzidos:
(2,1) e (4,6)

3-opt

- Retira 3 arcos do ciclo.
- Em um ciclo com n nós,
  - há C(n,3) modos de eliminar 3 arcos;
  - há 4 maneiras de reconectar os trechos que sobram (sem cair em um movimento 2-opt).



Cross exchange (4-opt)

- Retira 4 arcos do ciclo.
- Religa o ciclo formando uma cruz.

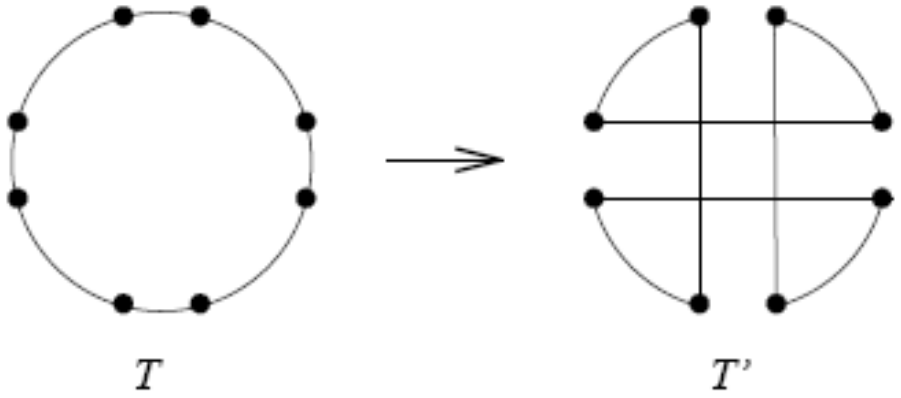


Figura extraída de Applegate et alii, Finding tours in the TSP.

Voltando ao VNS

- Suponhamos que um mínimo local tenha sido encontrado.
- Queremos deixar a vizinhança deste ponto e encontrar outro vale, preferencialmente um mais profundo.
- Para tanto, precisamos decidir:
  - Em que direção andar
  - O quanto andar nessa direção
  - Como modificar os movimentos se eles não são bem-sucedidos.

VNS reduzido

- Geralmente, usa vizinhanças “aninhadas”.
- Gera aleatoriamente um ponto na vizinhança de x.
- Se esse ponto não melhora f(x), passa-se à vizinhança seguinte.

VNS reduzido - Algoritmo

1 Definir um conjunto de k_{max} vizinhanças;
2 Determinar uma solução inicial x;
3 Enquanto não é satisfeito um critério de parada,
3.1 $k \leftarrow 1$;
3.2 Enquanto ($k \leq k_{max}$),
3.2.1 Gerar aleatoriamente $x' \in N_k(x)$;
3.2.2 Se ($f(x') < f(x)$),
3.2.2.1 Então $x \leftarrow x'$; $k \leftarrow 1$;
3.2.2.2 Senão $k \leftarrow k + 1$;
3.2.3 Fim-se;
3.3 Fim-enquanto;
4 Fim-enquanto;

Critérios de parada: número máximo de iterações ou tempo máximo.

VNS básico

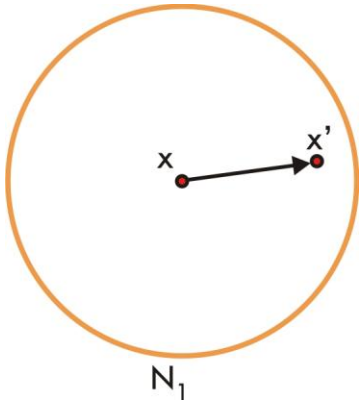
- Combina:
  - A seleção aleatória de um ponto de $N_k(x)$ (como VNSR)
  - A aplicação de um algoritmo de busca local

    ("f*irst improvement*" é o mais comum, mas "*best improvement*" também pode ser usado).

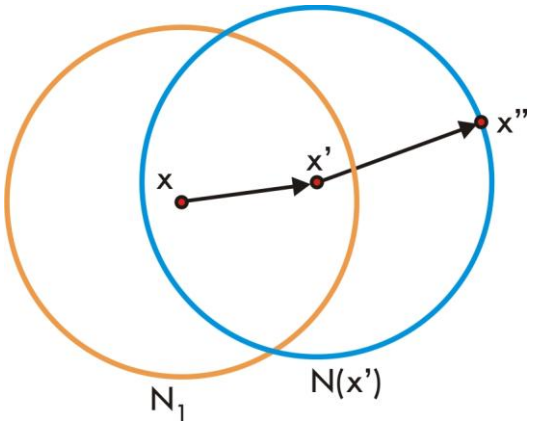
VNS básico - Algoritmo

1 Definir um conjunto de k_{max} vizinhanças.
2 Determinar uma solução inicial x;
3 Enquanto não é satisfeito um critério de parada,
3.1 $k \leftarrow 1$;
3.2 Enquanto ($k \leq k_{max}$),
3.2.1 Gerar, aleatoriamente, $x' \in N_k(x)$;
3.2.2 Usando um método de busca local, encontrar x'', mínimo local próximo de x'.
3.2.3 Se ($f(x'') < f(x)$),
3.2.3.1 Então $x \leftarrow x''$; $k \leftarrow 1$;
3.2.3.2 Senão $k \leftarrow k + 1$;
3.2.4 Fim-se;
3.3 Fim-enquanto;
4 Fim-enquanto;

VNS básico



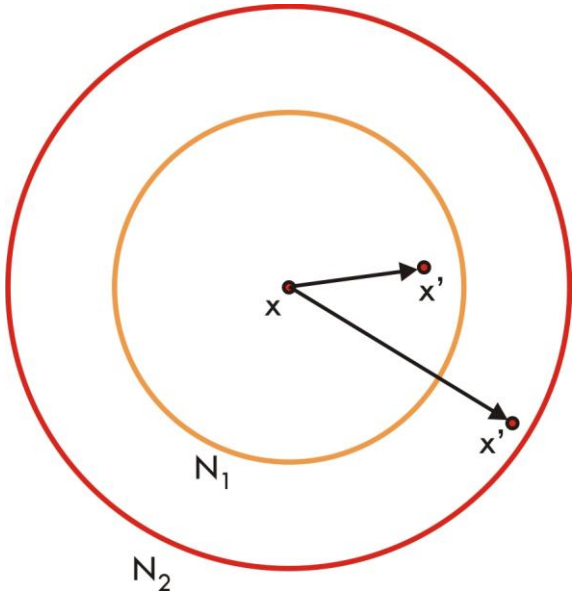
VNS básico



x'' é aceito quando f(x'') < f(x)

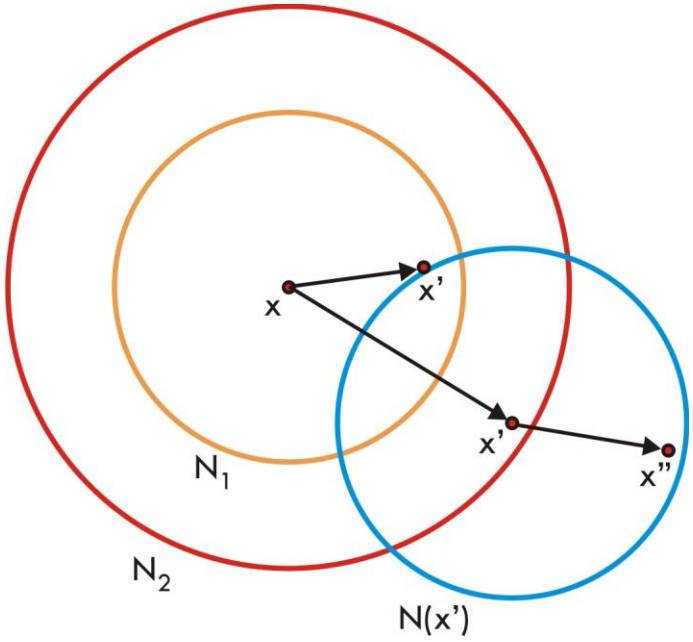
Figura extraída de Marcone Souza, Variable neighborhood search.

VNS básico



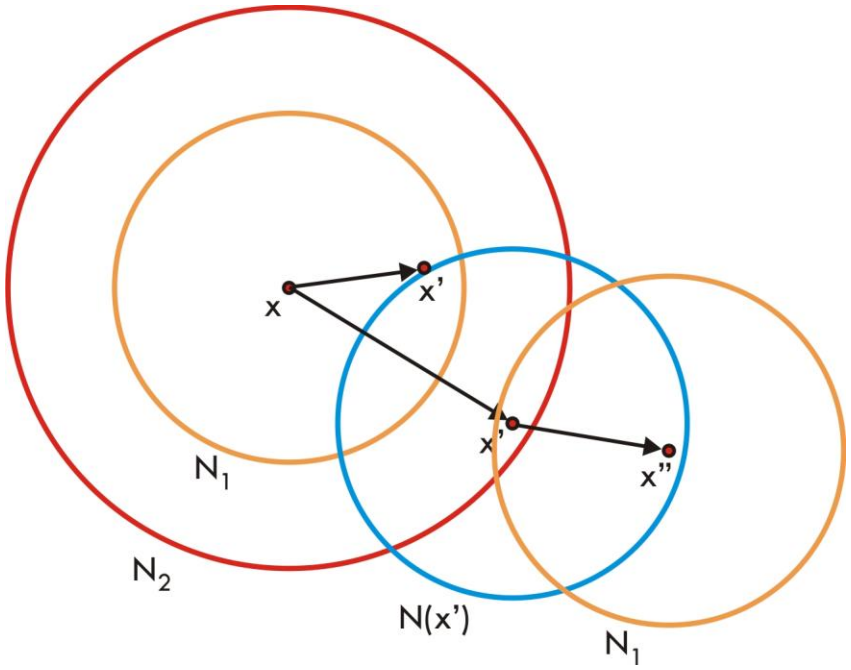
Se f não melhora na vizinhança de x' em N_1,
geramos novo x' em N_2.

VNS básico



Neste caso, nova busca local é feita na vizinhança de x' de N_2.

VNS básico



Se x" é aceito, a iteração externa é reiniciada,
e voltamos a usar a vizinhança N_1.

VNS básico

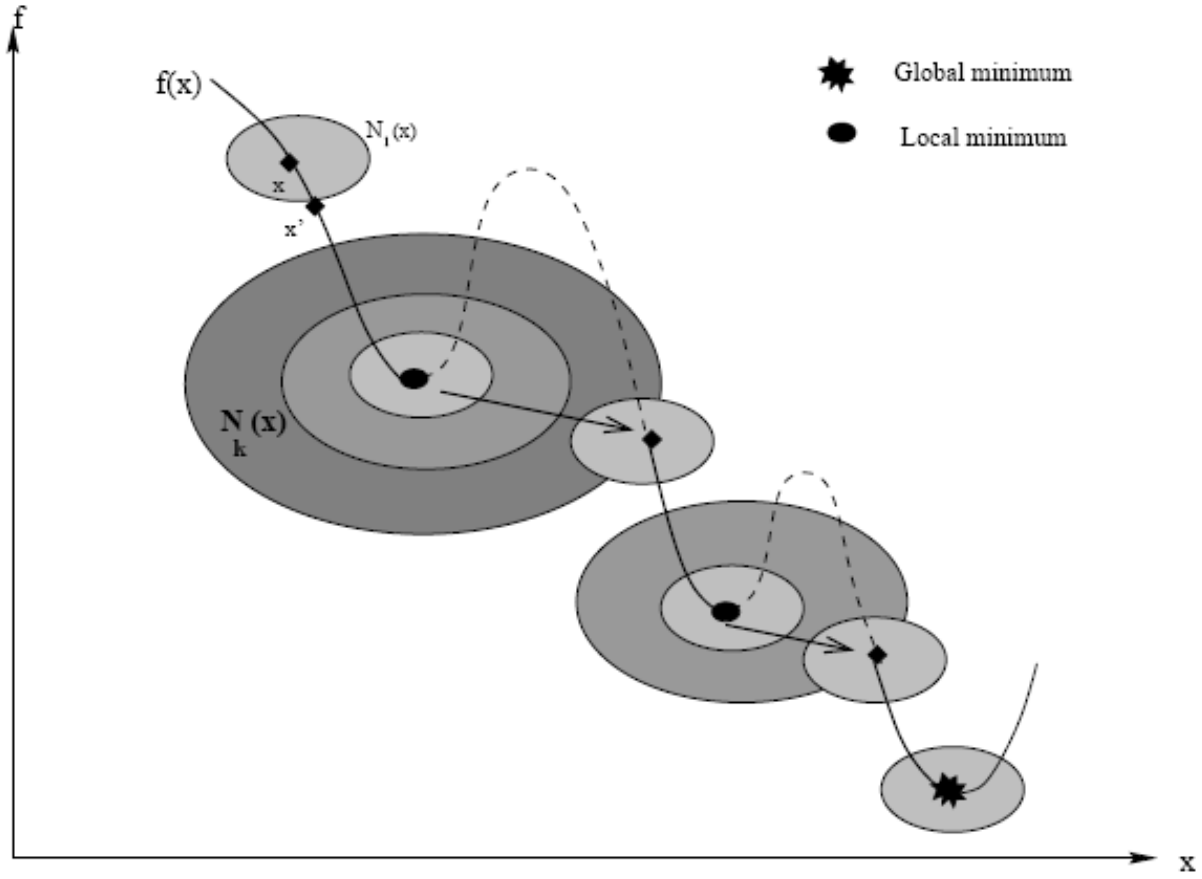


Figura extraída de Hansen e Mladenovic, variable neighborhood search methods, 2007.

VNS geral

- Combina
  - o VNS básico
  - com o emprego do VND como algoritmo de busca local.

VNS geral

1 Definir um conjunto de k_{max} vizinhanças;
2 Definir outro conjunto de m_{max} vizinhanças;
3 Determinar uma solução inicial x;
4 Enquanto não é satisfeito um critério de parada,
4.1 $k \leftarrow 1$;
4.2 Enquanto ($k \leq k_{max}$),
4.2.1 Gerar aleatoriamente $x' \in N_k(x)$;
4.2.2 $m \leftarrow 1$;
4.2.3 Enquanto ($m \leq m_{max}$),
4.2.3.1 Encontrar o melhor vizinho $x'' \in N'_m(x')$.
4.2.3.2 Se ($f(x'') < f(x')$),
4.2.3.2.1 Então $x' \leftarrow x''$; $m \leftarrow 1$;
4.2.3.2.2 Senão $m \leftarrow m + 1$;
4.2.3.3 Fim-se;
4.2.4 Fim-enquanto;
4.2.5 Se ($f(x') < f(x)$),
4.2.5.1 Então $x \leftarrow x'$; $k \leftarrow 1$;
4.2.5.2 Senão $k \leftarrow k + 1$;
4.2.6 Fim-se;
4.3 Fim-enquanto;
5 Fim-enquanto;

Extensões do VNS

- VNS enviesado (*skewed* VNS):
  - Muitas vezes, é necessário explorar vales distantes do ponto atual.
  - Mas gerar pontos aleatórios distantes é o mesmo que adotar recomeços (o que nem sempre é eficiente).
  - Neste caso, podemos adotar uma nova regra para aceitar pontos:

    $$x'' \text{ é aceito se } f(x'') < f(x) + \alpha\, \rho(x, x'')$$

    onde $\alpha > 0$ é uma constante e $\rho(x, x'')$ é uma função que mede a distância entre x e x".

- VNS não monótono:
  - Também podemos, com uma certa probabilidade, aceitar x" pior que x (ou pior que a melhor solução já encontrada).

Extensões do VNS

- VNS decomposto:
  - Depois de selecionarmos x' aleatoriamente,
  - podemos separar algumas características y que distinguem x' de x,
  - e fazer uma busca local levando em conta apenas no espaço y.
  - Geralmente, usa-se a própria VNS para fazer essa busca. Neste caso, temos a VNS em dois níveis (*bi-level VNS*).