Metaheurísticas

Thiago Noronha – tfn@dcc.ufmg.br

Algoritmos vs. Heurísticas

Algoritmo computa a resposta exata para um problema específico

Algoritmos vs. Heurísticas

Heurística computa uma resposta aproximada para um problema específico

Paradigmas de projeto de heurísticas

Metaheurísticas

NÃO resolvem um problema específico

Metaheurísticas

Solucionam problemas através de uma amostragem do conjunto de soluções

Metaheurísticas

Quanto mais eficaz e mais eficiente for esta amostragem melhor é a heurística resultante

Mecanismos fundamentais

- Intensificação
- Diversificação
- Memória

Características

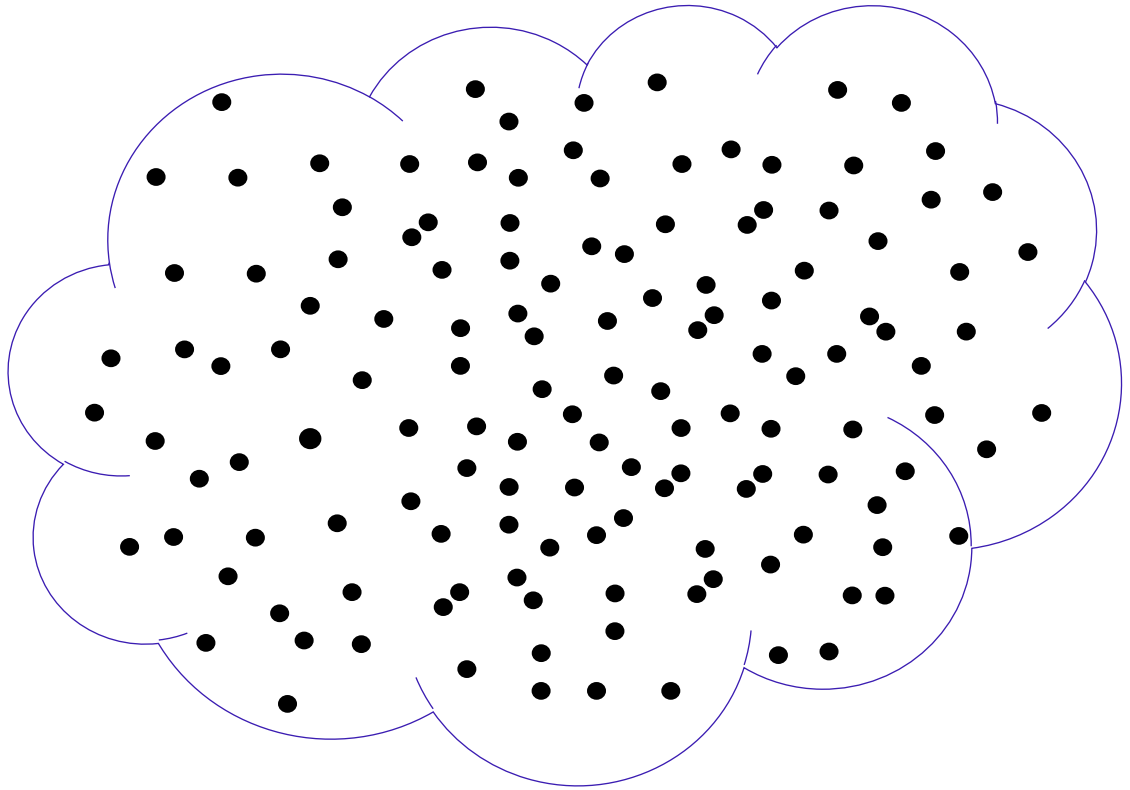
- Simplicidade
  - Baseadas em princípios simples e claros
- Generalidade
  - Capazes de solucionar diferentes problemas

Características

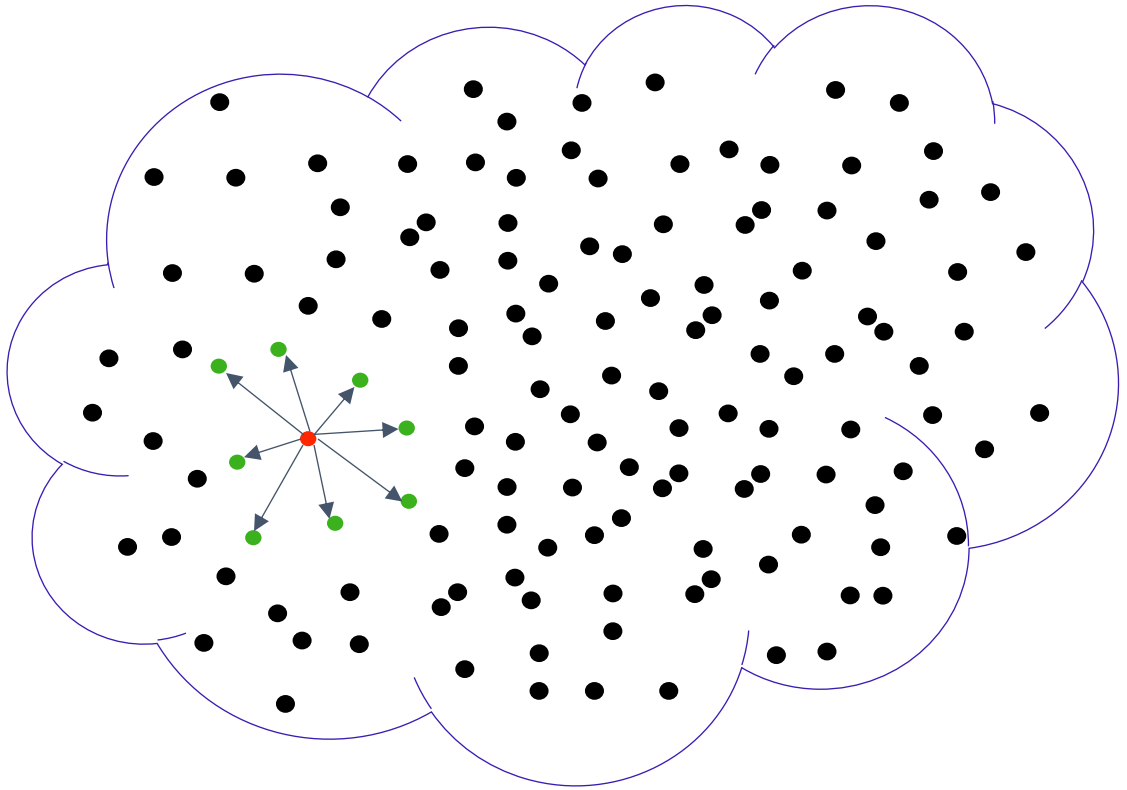
- Eficácia
  - Fornecem soluções quase ótimas
- Eficiência
  - Baixo custo computacional
- Robustez
  - Performance consistente sobre várias instâncias

Metaheurísticas de busca em vizinhança

Espaço de Busca



Função de vizinhança



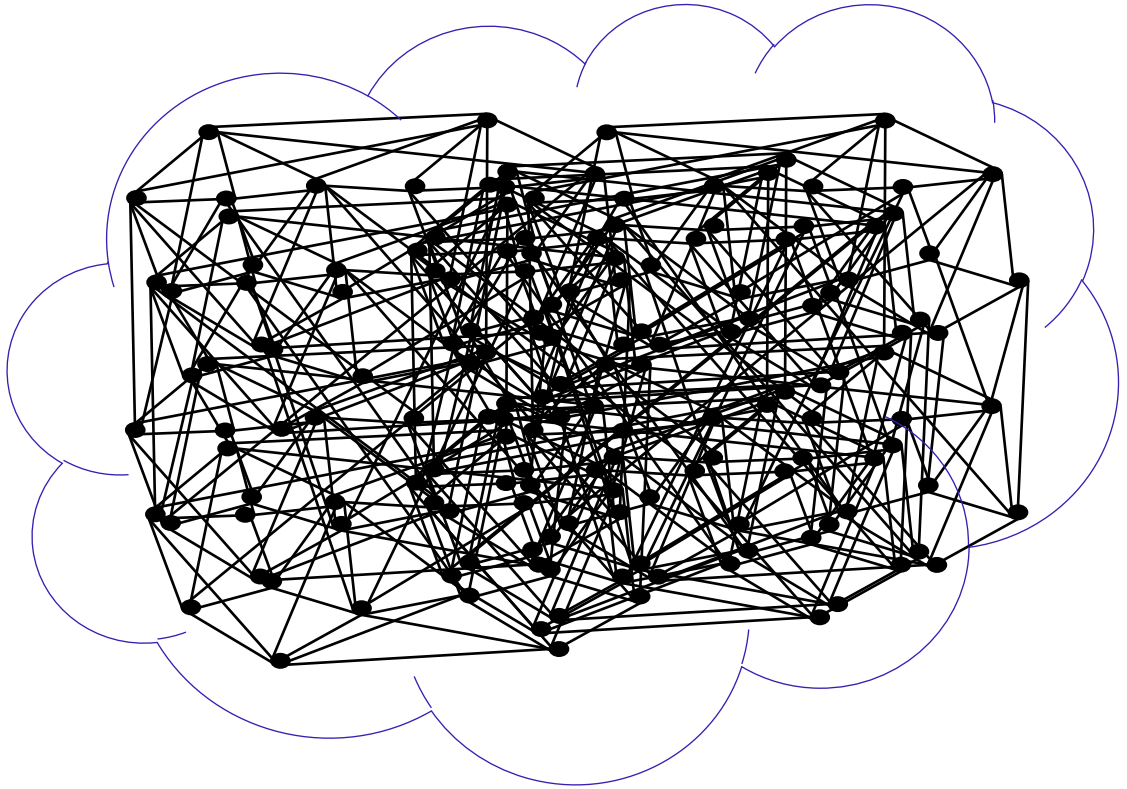
Função de vizinhança

- Função $N: \Gamma \longmapsto 2^{\Gamma}$
  - Mapeia uma solução $S \in \Gamma$
  - a um subconjunto $N(S) \subseteq \Gamma$,
  - onde Γ é o conjunto de soluções

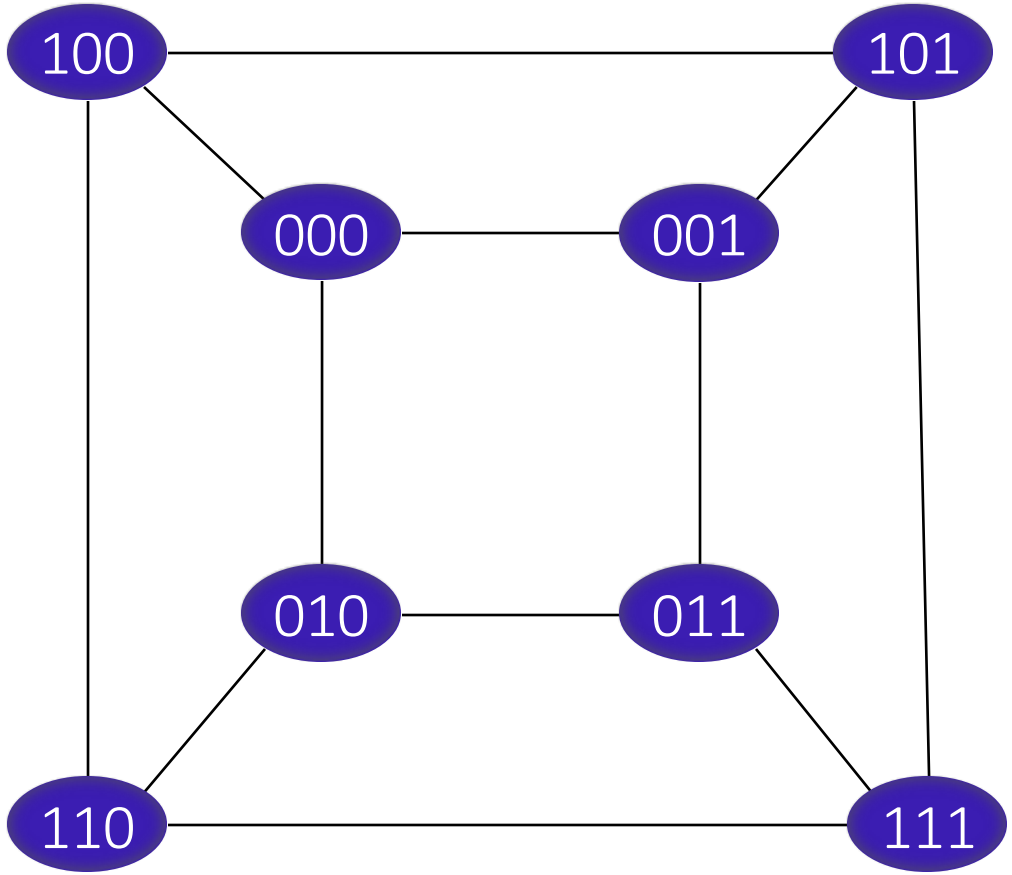
Grafo de vizinhança

- Duas soluções em Γ podem ser vizinhas ou não
  - Dependendo da função de vizinhança

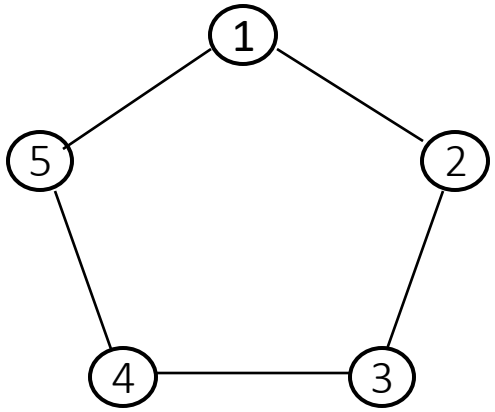
Grafo de Vizinhança



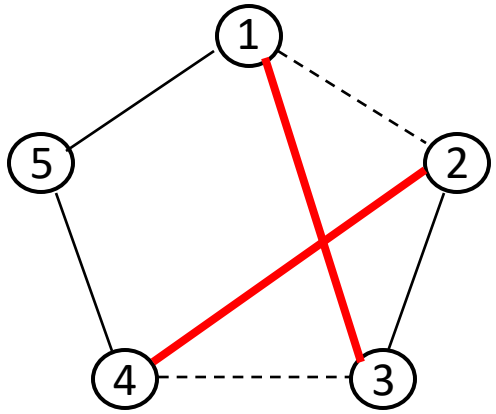
Vizinhança N^1

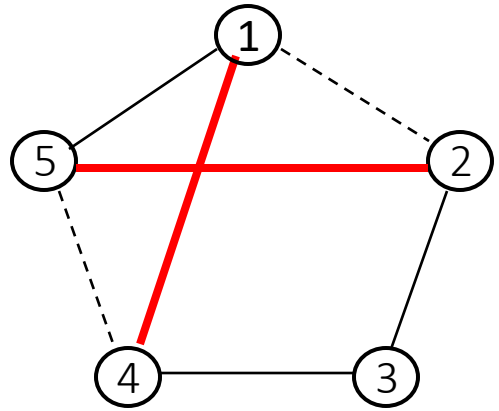


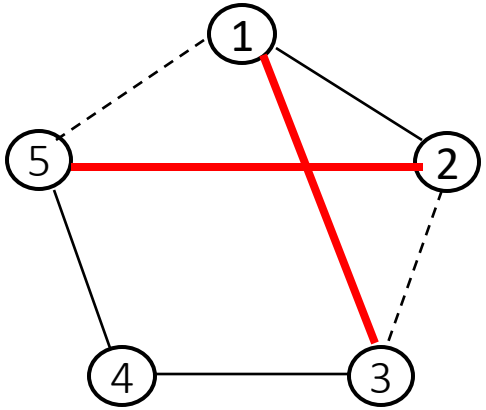
Vizinhança 2-opt

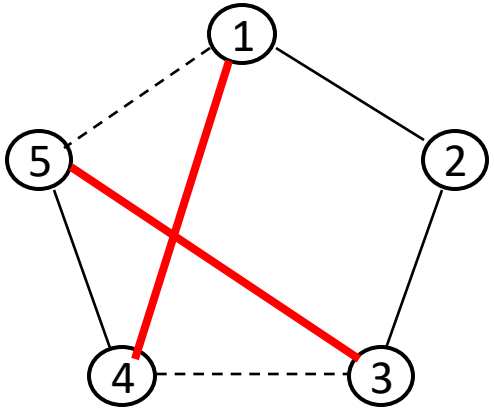


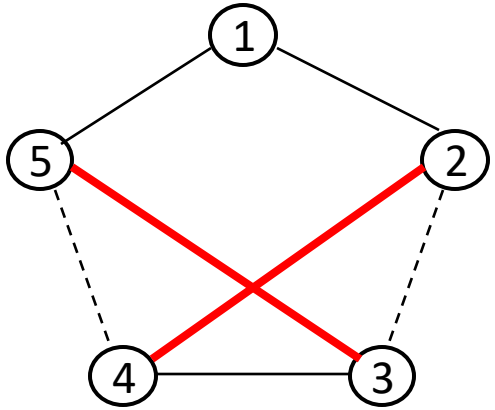
Vizinhança 2-opt



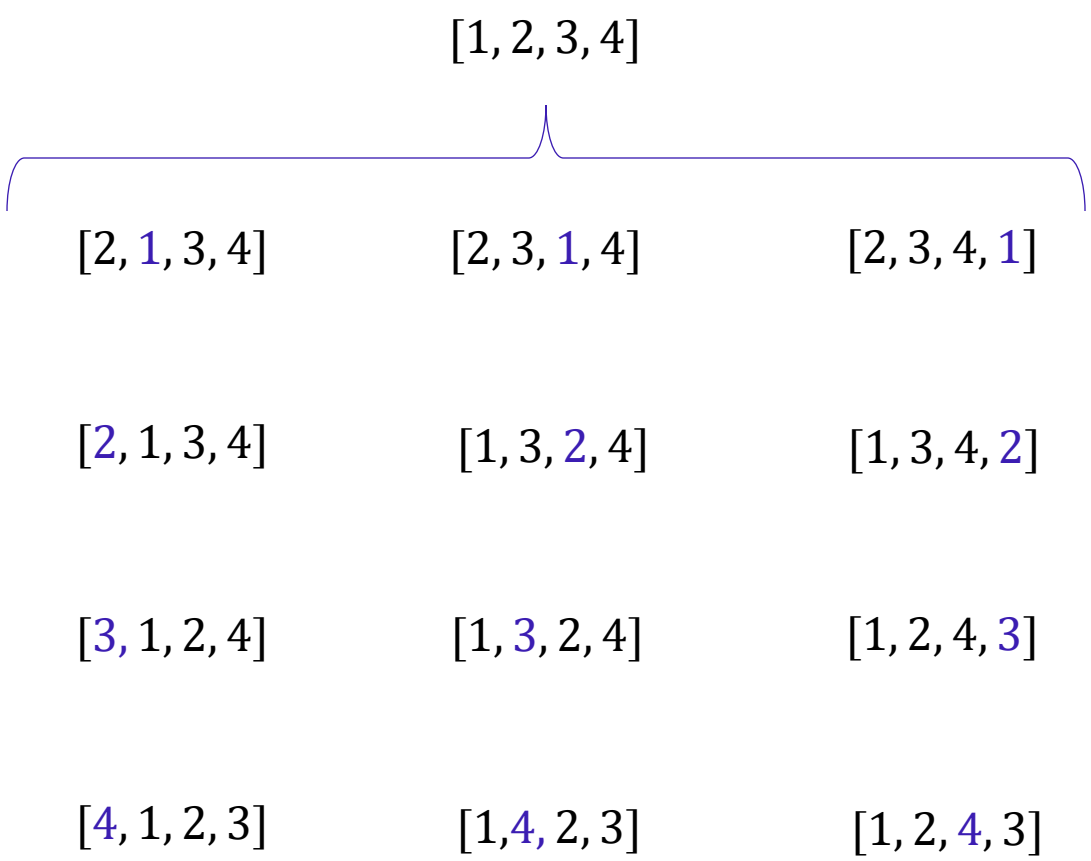




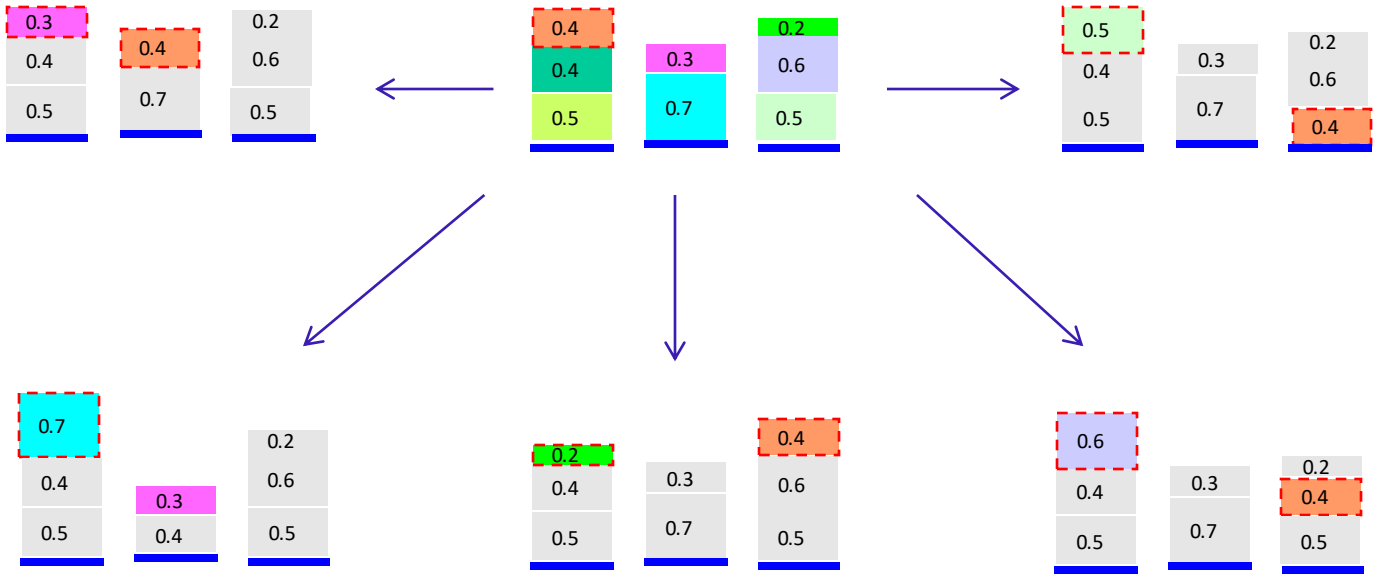




Vizinhança *Reinsert*



Swap


0.3
0.4
0.5
0.4
0.7
0.2
0.6
0.5
0.4
0.4
0.5
0.3
0.7
0.2
0.6
0.5
0.5
0.4
0.5
0.3
0.7
0.2
0.6
0.4
0.7
0.3
0.4
0.2
0.6
0.5
0.2
0.4
0.4
0.5
0.3
0.7
0.4
0.6
0.5
0.6
0.4
0.5
0.3
0.7
0.2
0.4
0.5

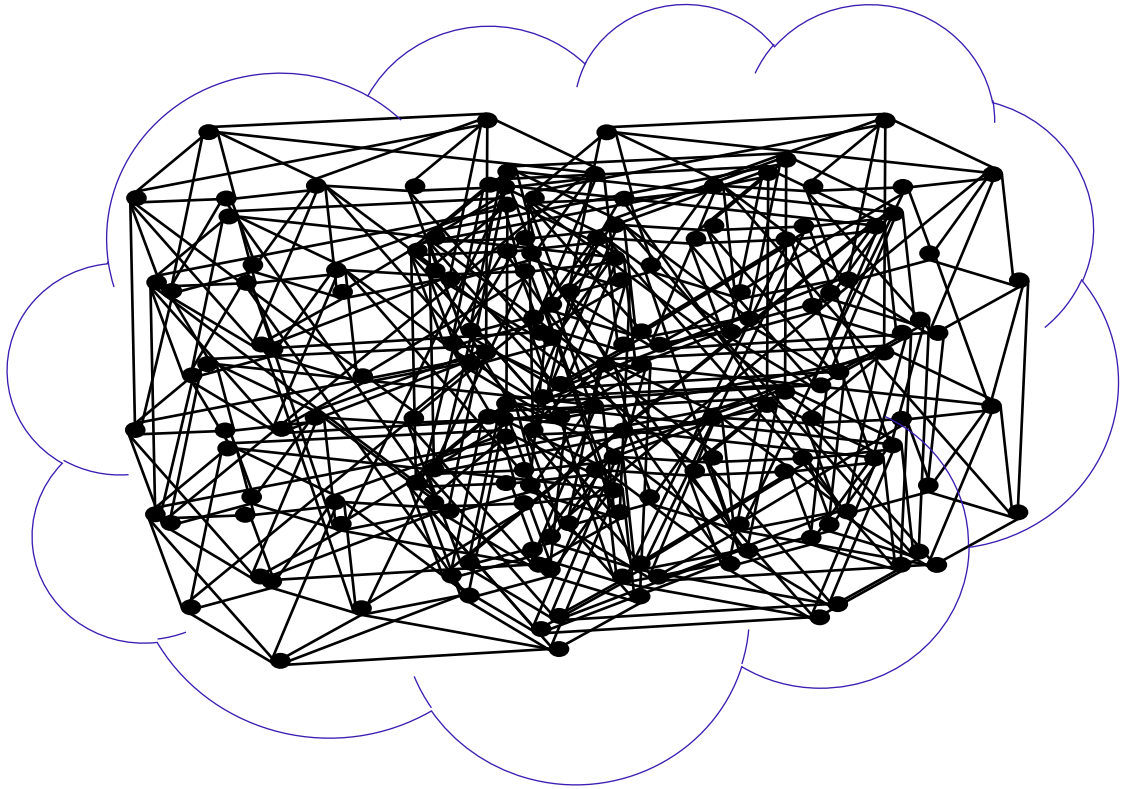
Heurísticas de busca em vizinhança

Exploram o espaço de soluções movendo-se entre soluções vizinhas

Heurística do passeio aleatório

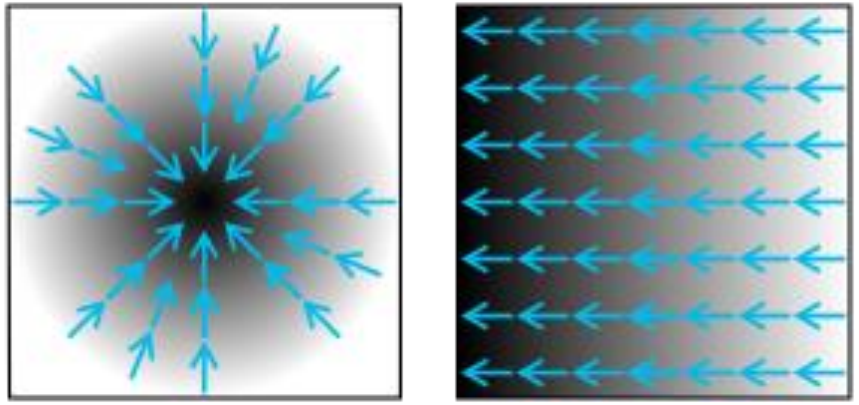
- Início
  - Solução viável
- Iteração
  - Mover-se para um vizinho aleatoriamente
- Parada
  - Qualquer critério de convergência

Grafo de Vizinhança



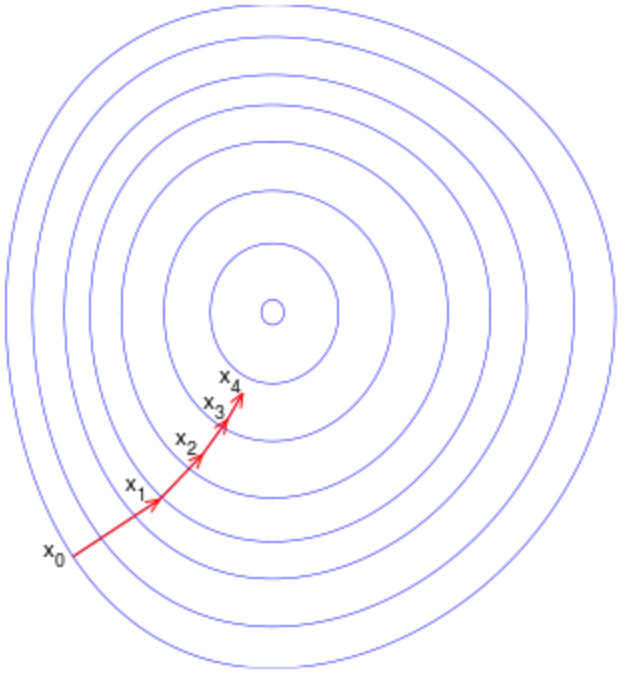
Heurísticas de busca local

Inspirados em algoritmos de gradiente



Fonte: https://en.wikipedia.org/wiki/Gradient

Inspirados em algoritmos de gradiente

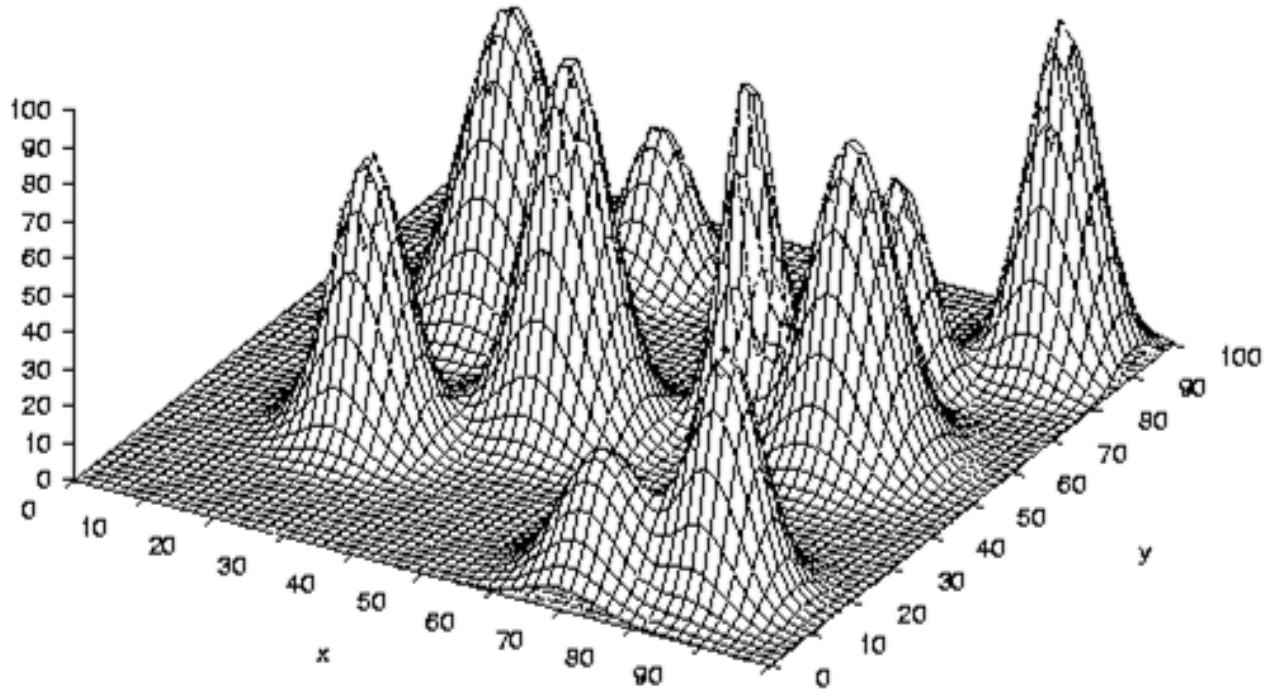


Fonte: https://en.wikipedia.org/wiki/Gradient

Princípio

Soluções semelhantes tendem a ter custo semelhante

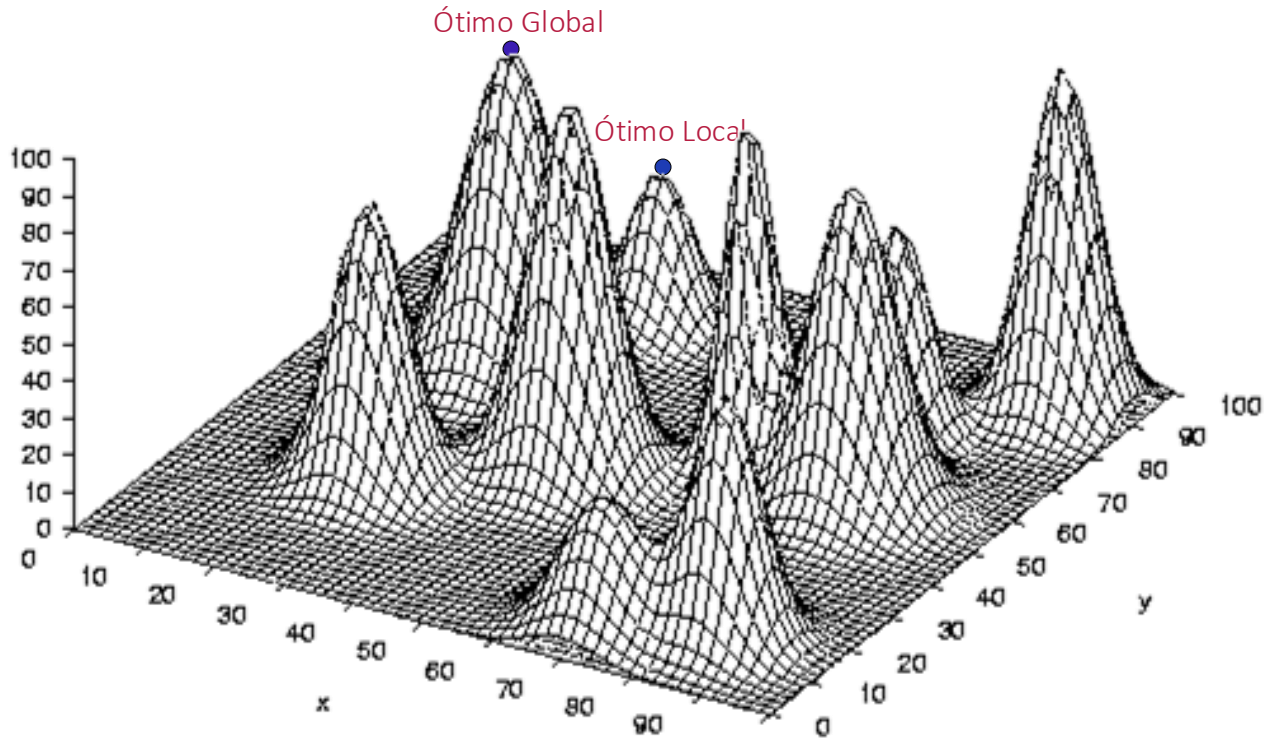
Superfície definida pelo custo das soluções



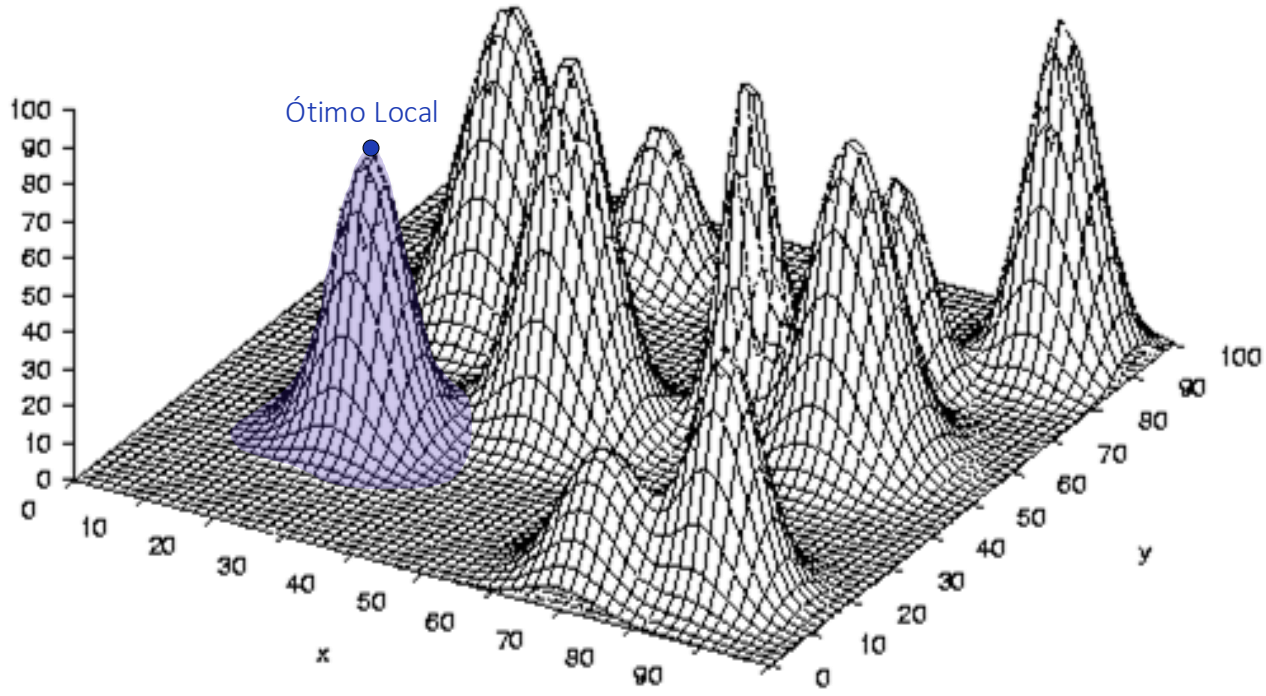
Heurística de busca local

- Início
  - Solução viável
- Iteração
  - Mover-se para o vizinho melhor
- Parada
  - Nenhum vizinho melhor encontrado

Ótimo Local vs. Ótimo global



Região de Atração



Ótimo Local vs. Ótimo global

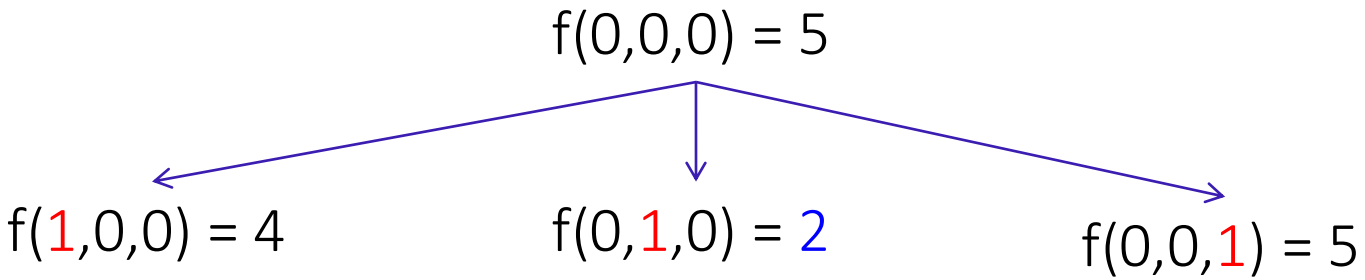
Um ótimo global é ótimo local

de qualquer vizinhança

Busca local melhor aprimorante

f(0,0,0) = 5

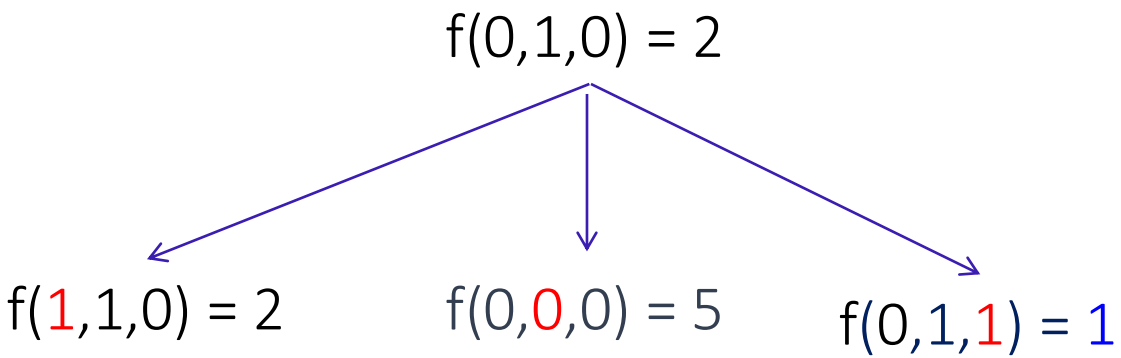
Busca local melhor aprimorante



Busca local melhor aprimorante

f(0,1,0) = 2

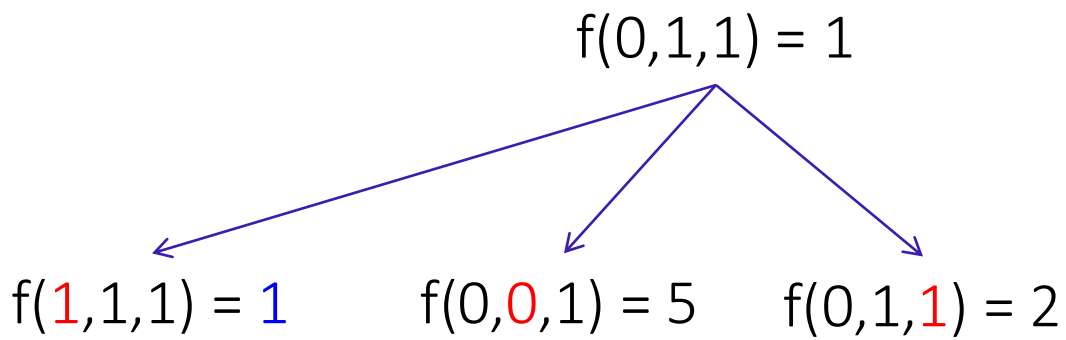
Busca local melhor aprimorante



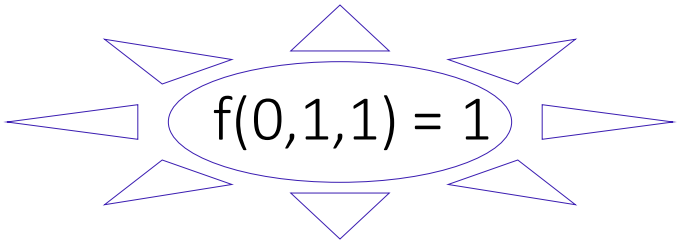
Busca local melhor aprimorante

f(0,1,1) = 1

Busca local melhor aprimorante



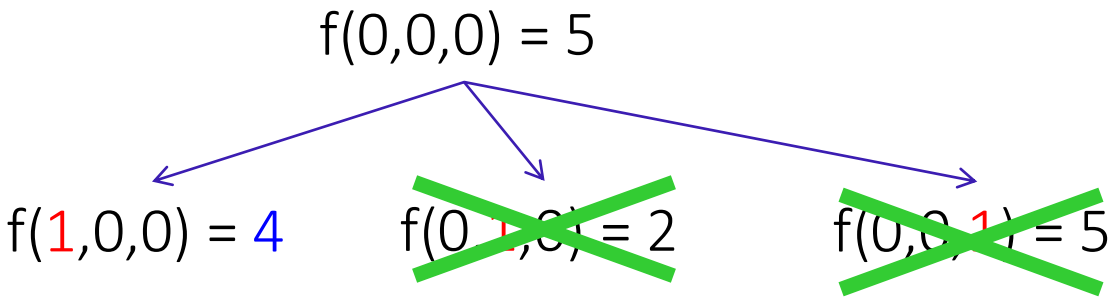
Busca local melhor aprimorante



Busca local primeiro aprimorante

f(0,0,0) = 5

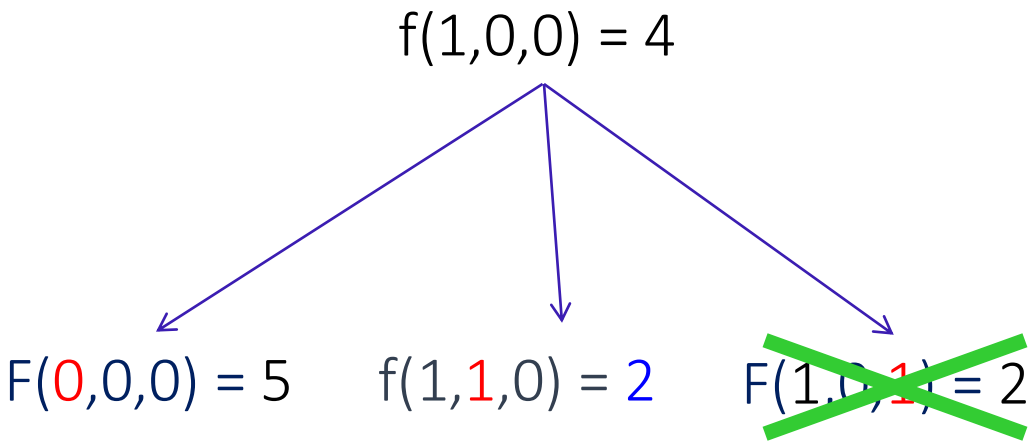
Busca local primeiro aprimorante



Busca local primeiro aprimorante

f(1,0,0) = 4

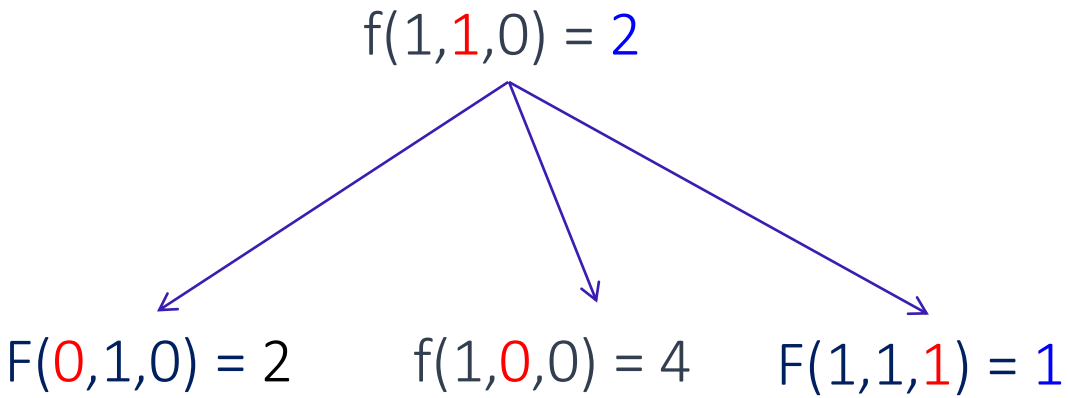
Busca local primeiro aprimorante



Busca local primeiro aprimorante

f(1,1,0) = 2

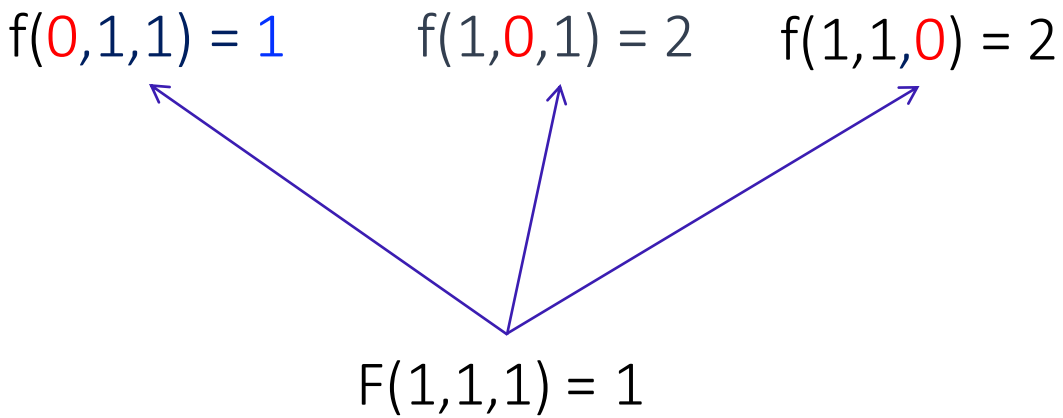
Busca local primeiro aprimorante



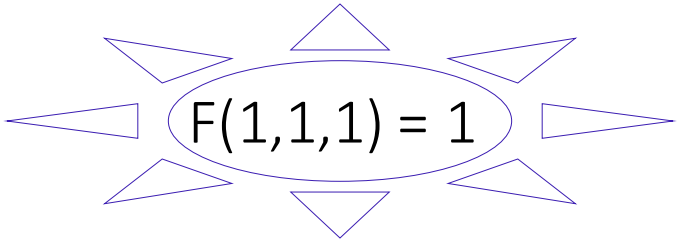
Busca local primeiro aprimorante

F(1,1,1) = 1

Busca local primeiro aprimorante



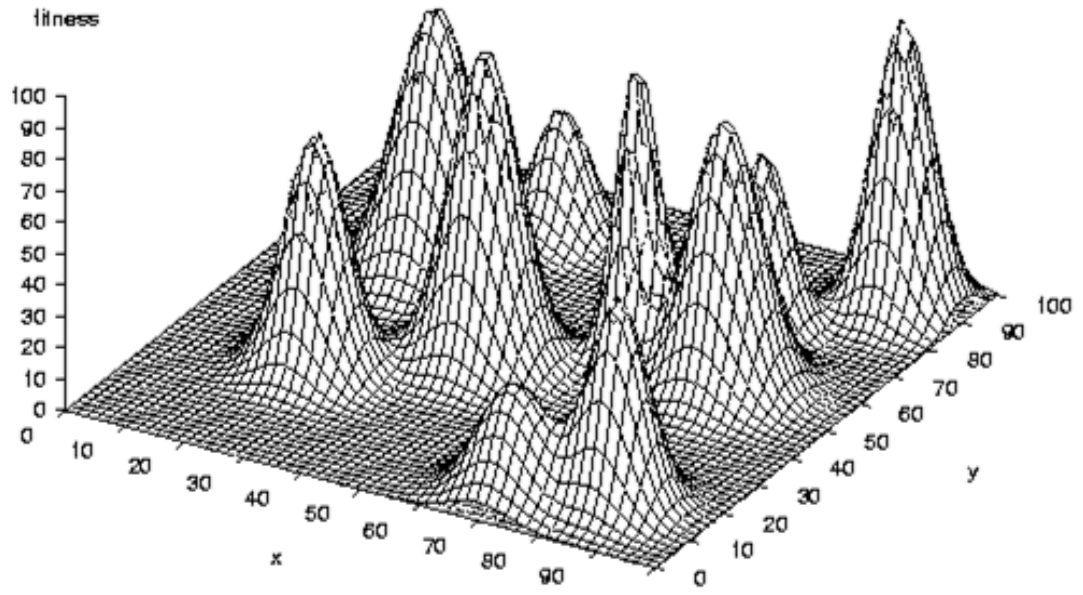
Busca local primeiro aprimorante



Estratégias de Busca

- Melhor aprimorante
  - Move-se para o melhor de todos os vizinhos
- Primeiro aprimorante
  - Mover-se para o primeiro vizinho aprimorante
  - A ordem de visitação é arbitrária

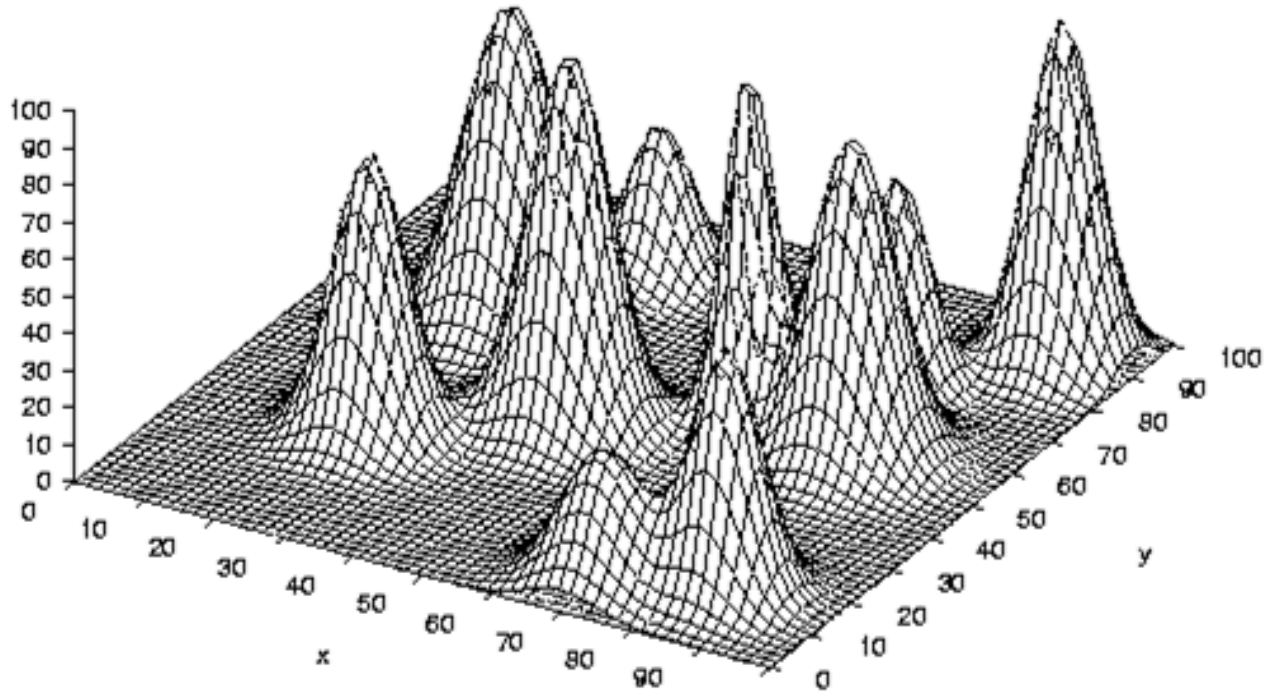
Estratégias de Busca



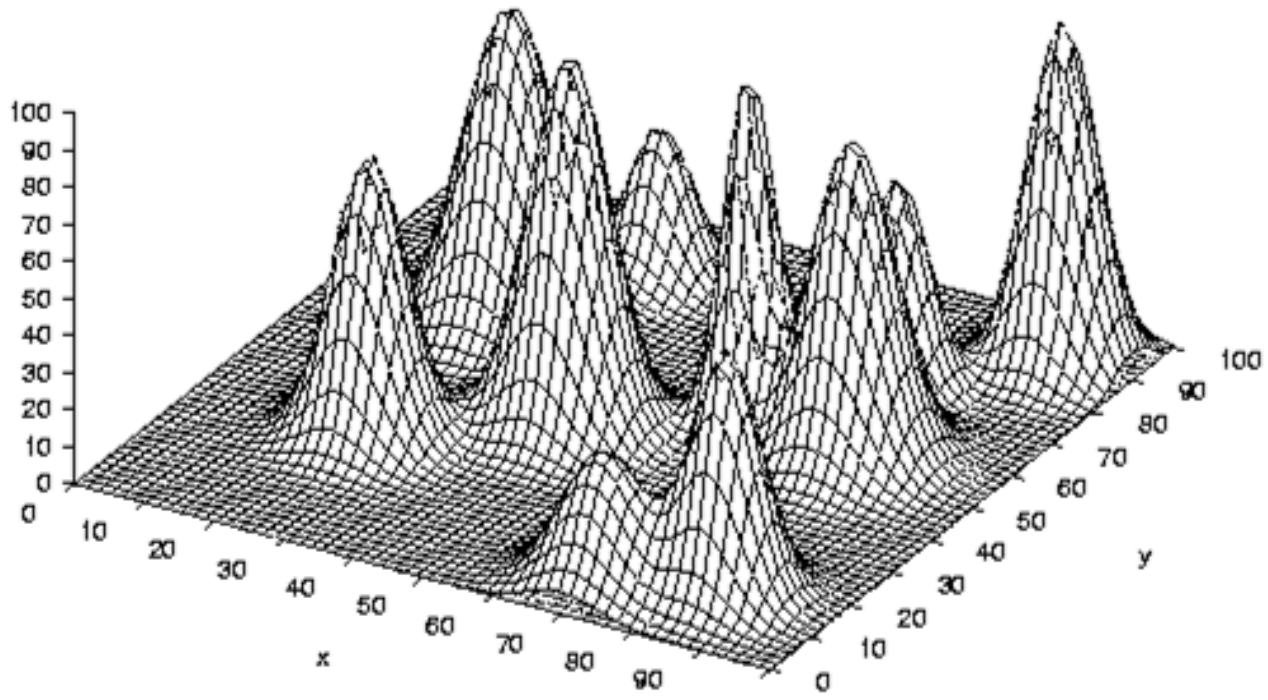
Questões fundamentais

- Densidade do grafo de busca
  - Quantidade de ótimos locais
  - Conexidade do espaço de soluções
- Custo de cada iteração
  - Número de vizinhos
  - Complexidade de avaliar cada vizinho

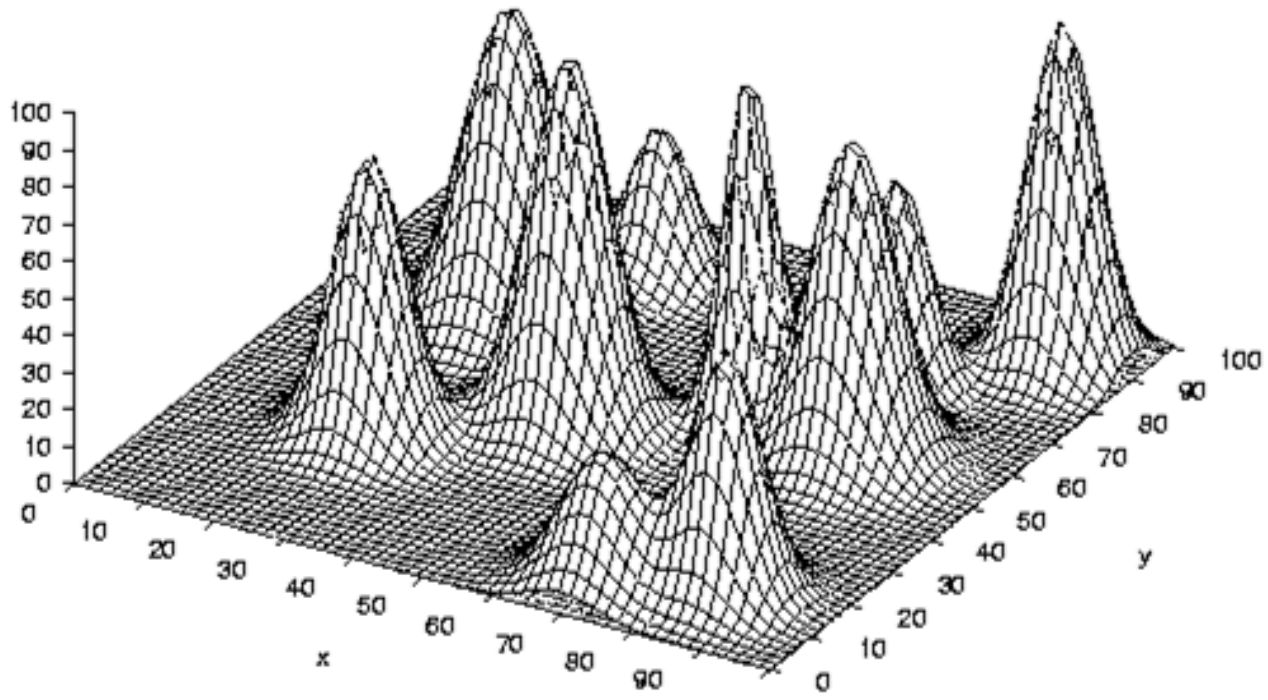
Limitações: Sensível à solução inicial



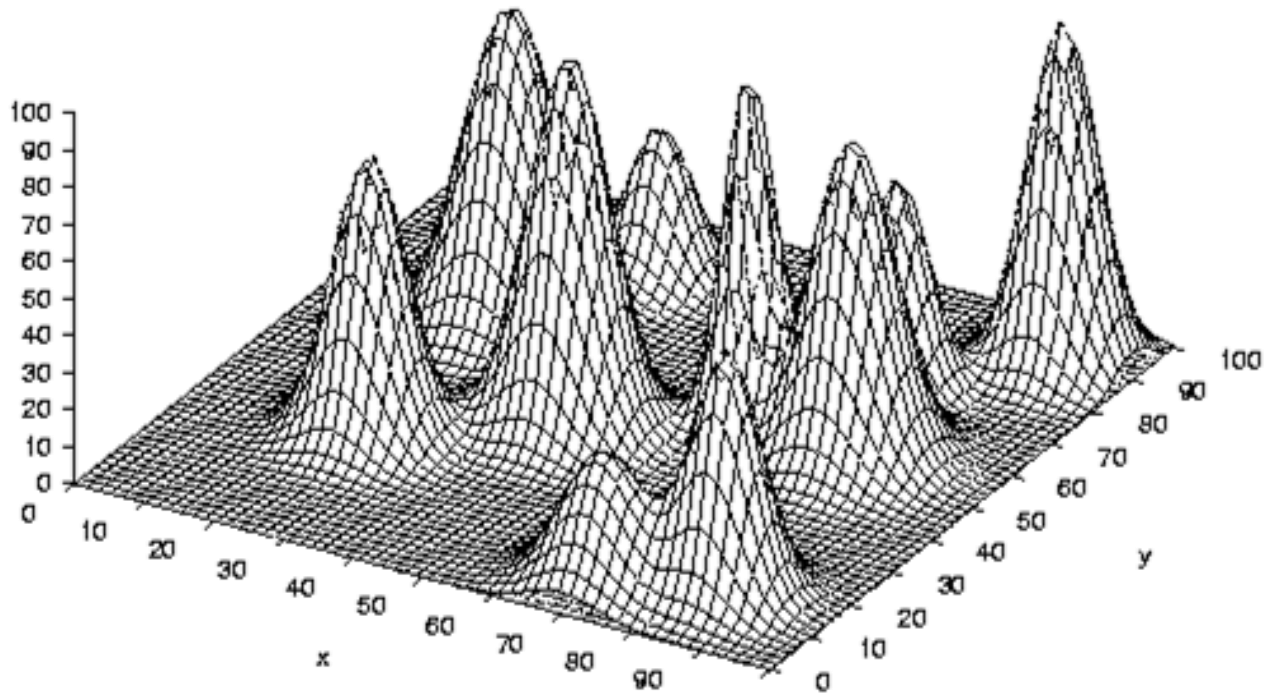
Limitações: Sensível à vizinhança utilizada



Limitações: Convergência prematura



Limitações: Exponencial no pior caso



Consideração Final

É um algoritmo de busca exaustiva na vizinhança

Very Large Neighborhood Search

Very Large Neighborhood Search

Dados uma instância e uma solução para o problema, encontrar um vizinho melhor

Very Large Neighborhood Search

- Vizinhanças muito vizinhos
  - $|N(S)| = (n^4)$
  - $|N(S)| = (n^{10})$
  - $|N(S)| = (2^n)$
- Não é computacionalmente viável resolver por inspeção

Very Large Neighborhood Search

Resolve um problema de otimização semelhante ao original

Melhor vizinho de $\bar{x}$ na vizinhança N^k

$$I = \{1, 2, \ldots, |I|\}$$

$$a_i \in N$$

$$c_i \in N$$

$$B \in N$$

$$\max \sum_{i \in I} c_i x_i \quad s.t.$$

$$\sum_{i \in I} a_i x_i \leq B$$

$$x_i \in \{0,1\}$$

Melhor vizinho de $\bar{x}$ na vizinhança N^k

$$I = \{1, 2, \ldots, |I|\}$$

$$a_i \in N$$

$$c_i \in N$$

$$B \in N$$

$$\max \sum_{i \in I} c_i x_i \quad s.t.$$

$$\sum_{i \in I} a_i x_i \leq B$$

$$x_i \in \{0,1\}$$

$$\sum_{i \in I} |\bar{x}_i - x_i| \leq k$$

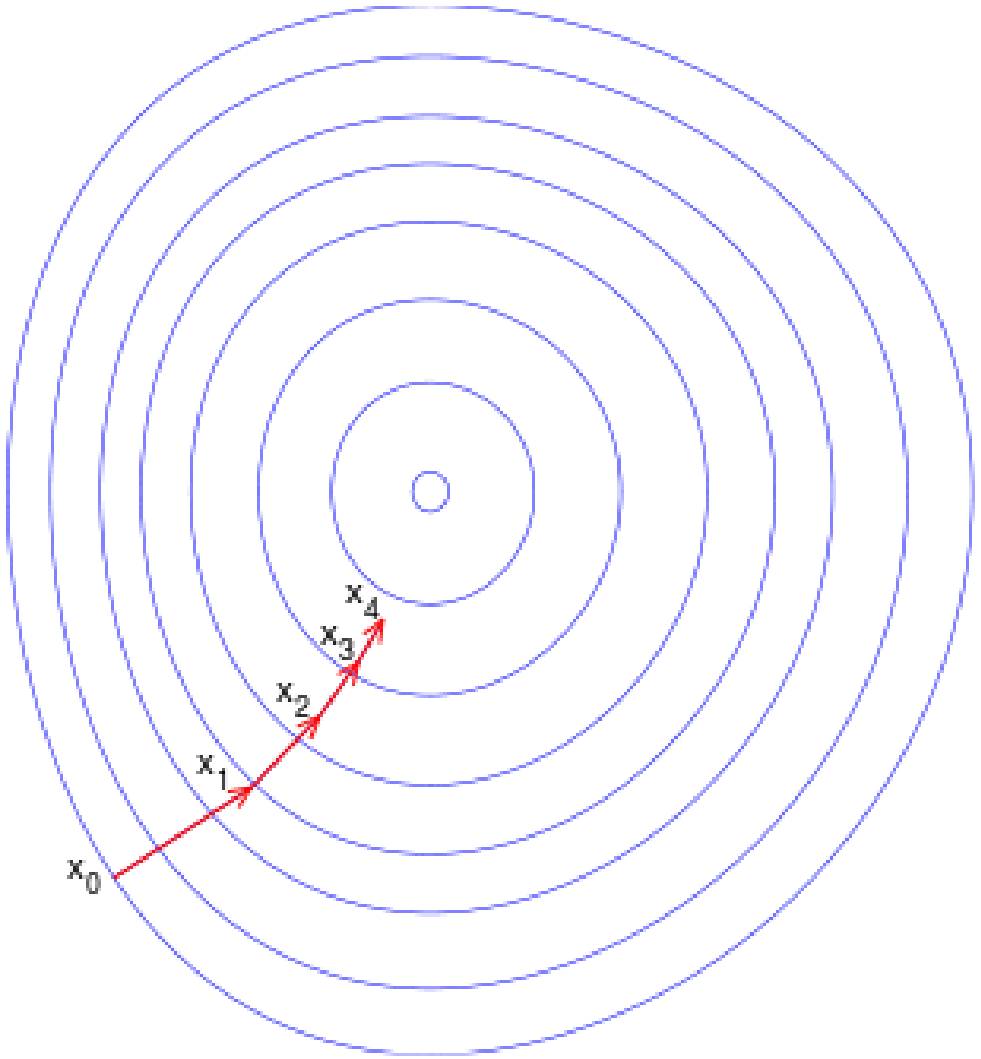
Considerações finais

- Balancear o tamanho da vizinhança com o incremento na qualidade da solução
- Este subproblema pode ser polinomial

Variable Neighborhood Descent

Busca local em múltiplas vizinhanças

Inspirados em algoritmos de gradiente



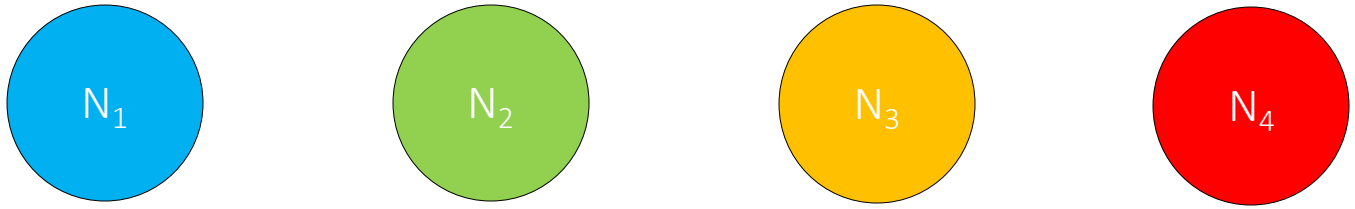
Fonte: https://en.wikipedia.org/wiki/Gradient

Princípio

- Aplicar busca local na união de múltiplas vizinhanças
  - $N(S) = N_1(S) \cup N_2(S) \cup \cdots \cup N_k(S)$
  - $\Psi(N) = \Psi(N_1) \cap \Psi(N_2) \cap \cdots \cap \Psi(N_k)$
    - Conjunto de ótimos locais

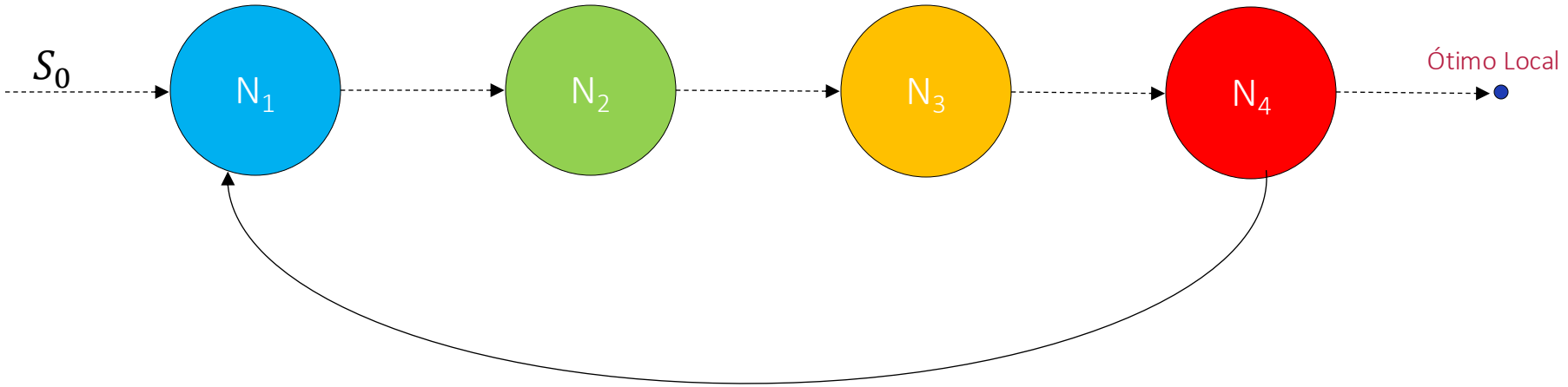
Variable Neighborhood Descent

- $N(S) = N_1(S) \cup N_2(S) \cup N_3 \cup (S) \cup N_4(S)$

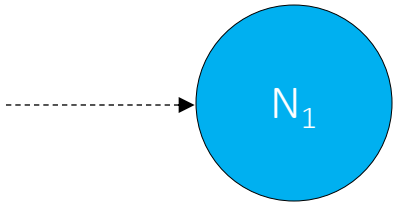


Variable Neighborhood Descent

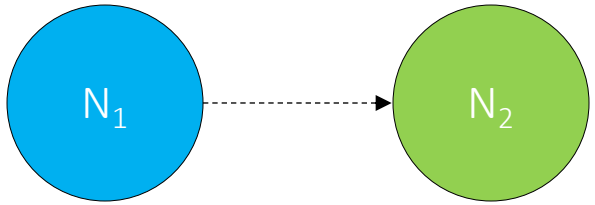
- $N(S) = N_1(S) \cup N_2(S) \cup N_3 \cup (S) \cup N_4(S)$



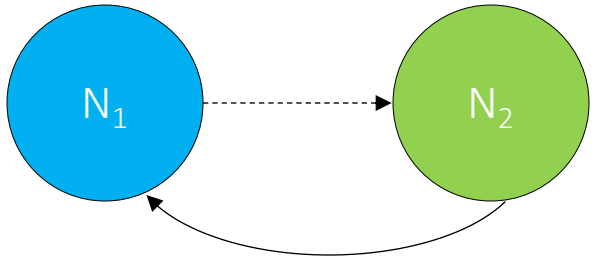
Variable Neighborhood Descent



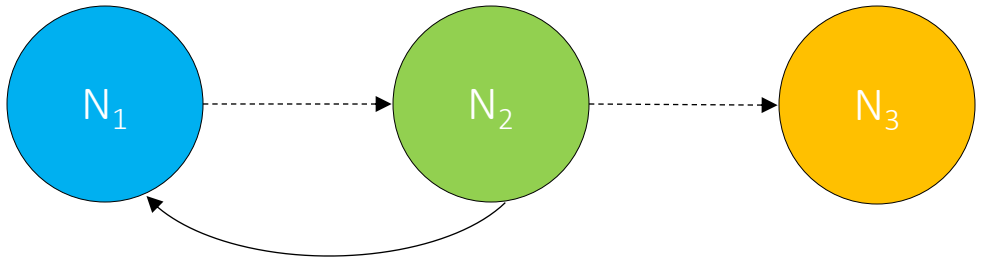
Variable Neighborhood Descent



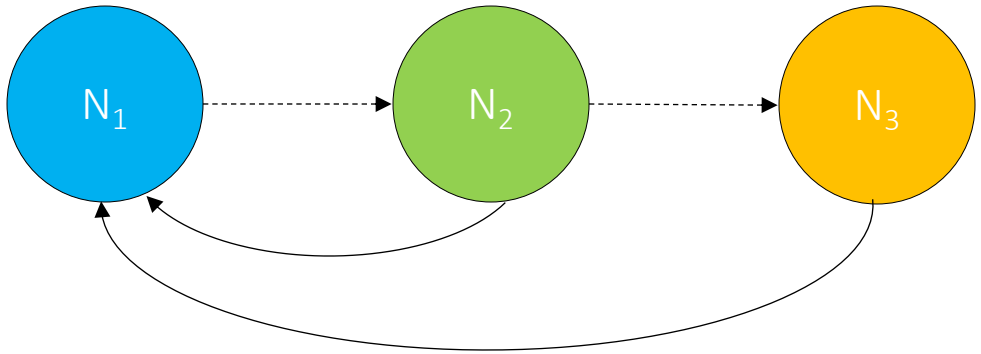
Variable Neighborhood Descent



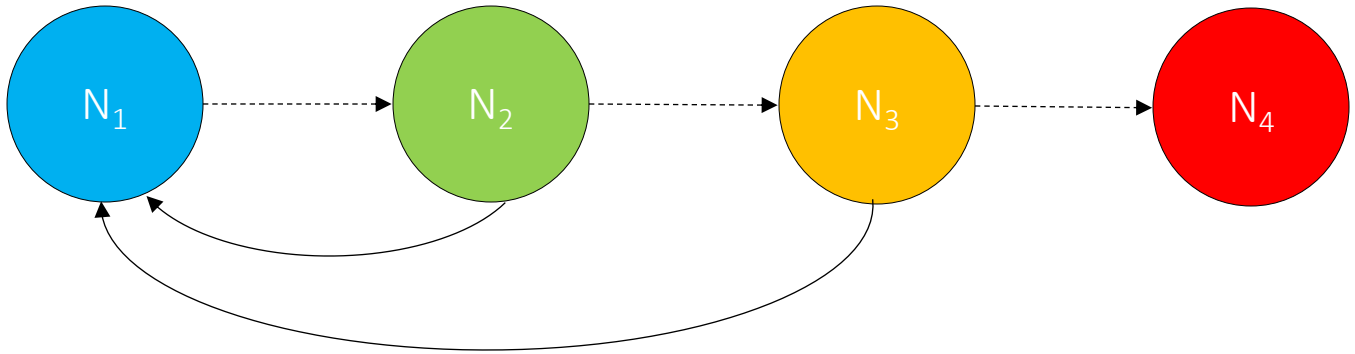
Variable Neighborhood Descent



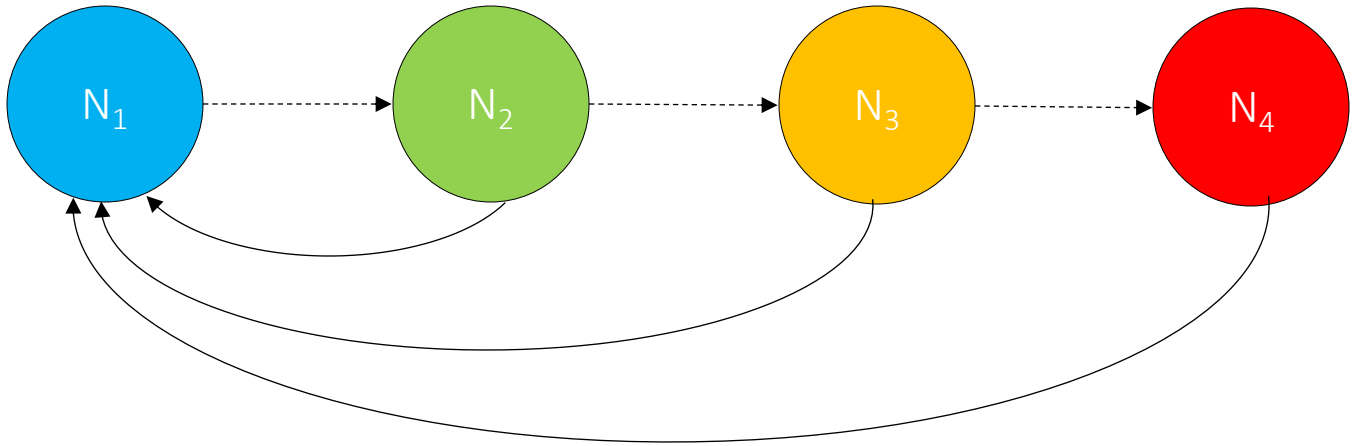
Variable Neighborhood Descent



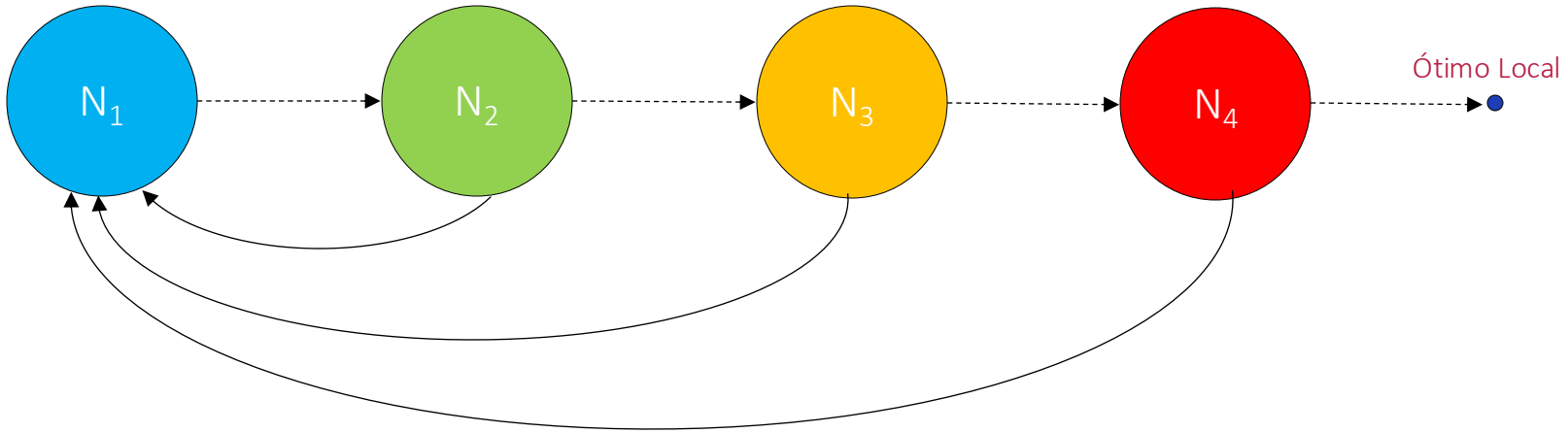
Variable Neighborhood Descent



Variable Neighborhood Descent



Variable Neighborhood Descent



Considerações Finais

- Quanto maior número de vizinhanças
  - Melhor a qualidade dos ótimos locais
  - Maior é o custo computacional
- Só é eficiente se as vizinhanças forem suficientemente distintas

Considerações Finais

- Princípio fundamental do VND
  - Quando chegar em um ótimo local, trocar a vizinhança da busca local

Busca Tabu

Princípio

O que fazer quando chegar em um

Ótimo Local?

Princípio

Continue fazendo busca local!

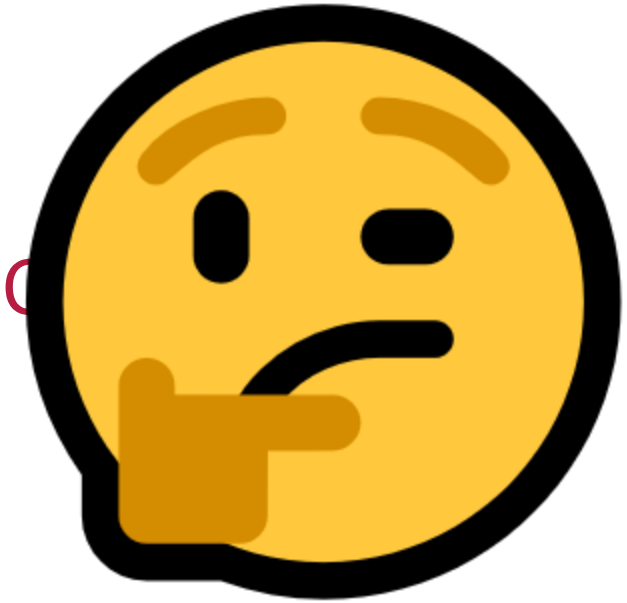
Problema de Ciclagem

Como evitar ciclos?

Lista Tabu

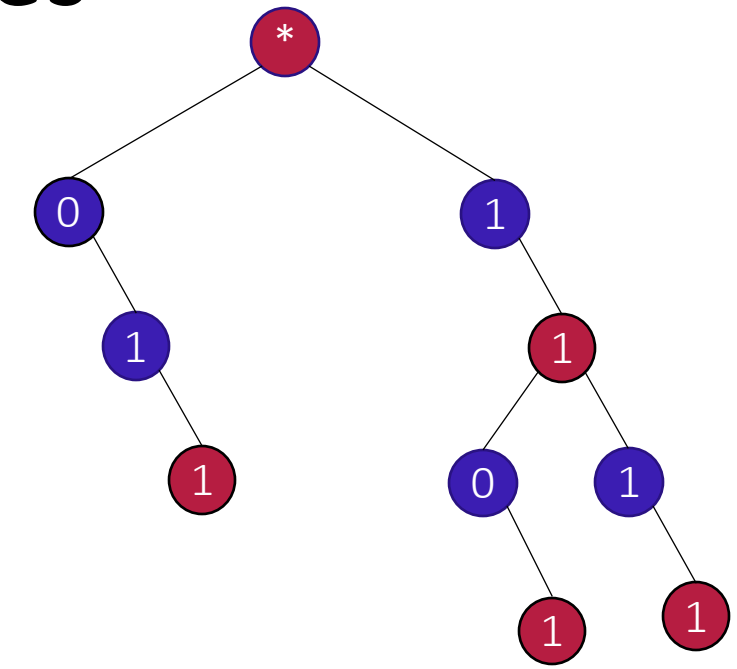
Armazena todas as soluções visitadas

Lista Tabu

Armazena to  ões visitadas

Implementação da Lista Tabu

- Prefix Trees
  - 1111
  - 0011
  - 1011
  - 0110
  - 0000



Implementação da Lista Tabu

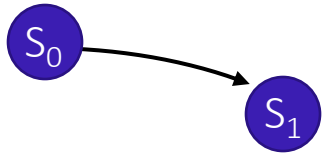
- Armazena as *k* últimas soluções
  - Reduz O(*n*) bits de memória
  - Evita apenas ciclos de tamanho $\leq k$

Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_0]

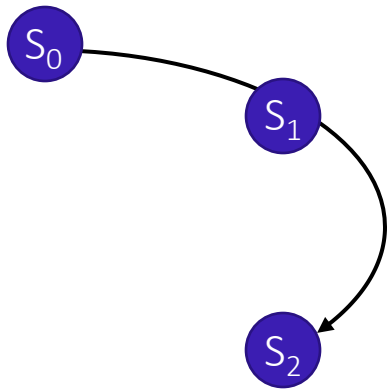
Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_0, S_1]



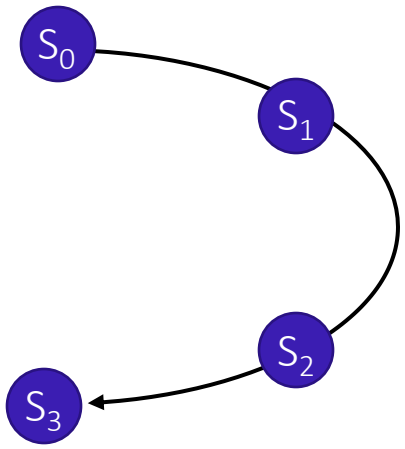
Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_0, S_1, S_2]



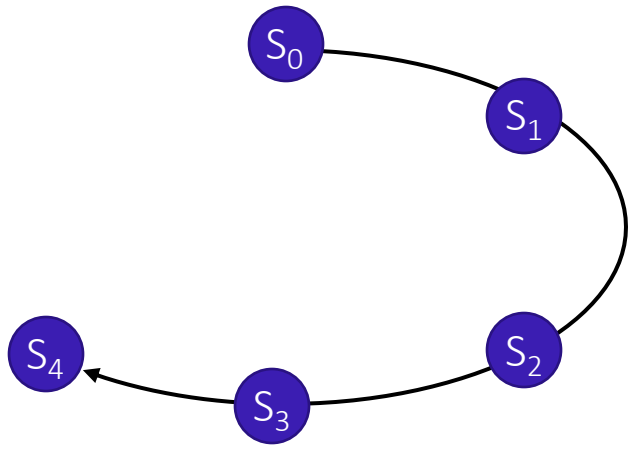
Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_0, S_1, S_2, S_3]



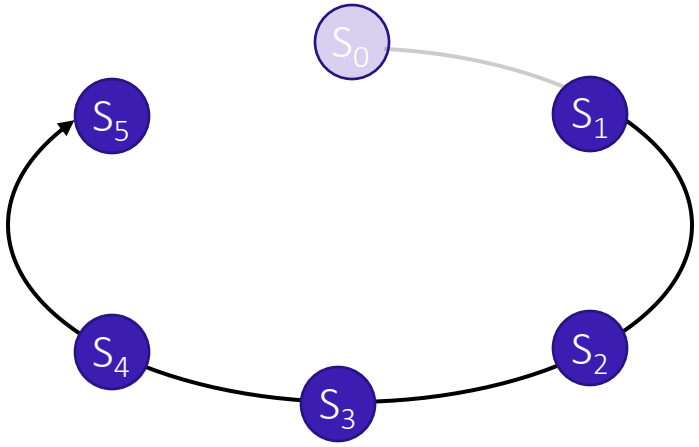
Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_0, S_1, S_2, S_3, S_4]



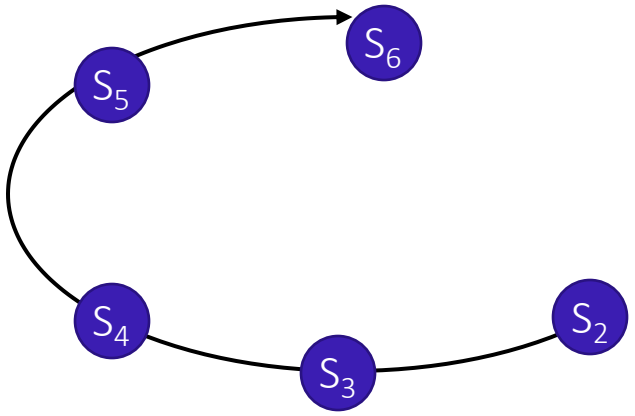
Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_1, S_2, S_3, S_4, S_5]



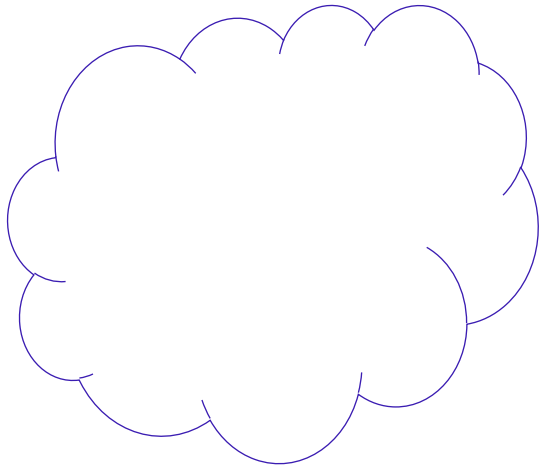
Implementação da Lista Tabu

- Armazena as *5* últimas soluções
  - L = [S_2, S_3, S_4, S_5 , S_6]



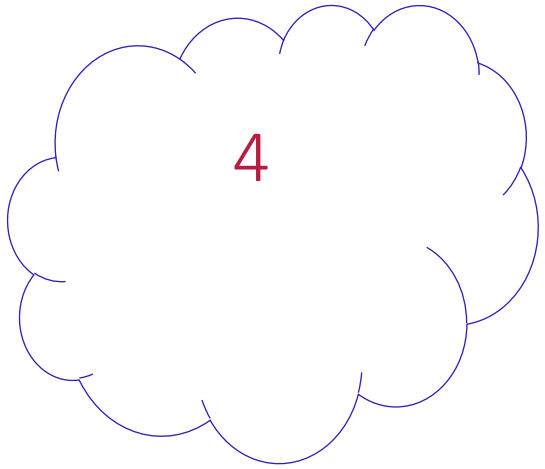
Implementação da Lista Tabu

- Tabelas de *Hash*



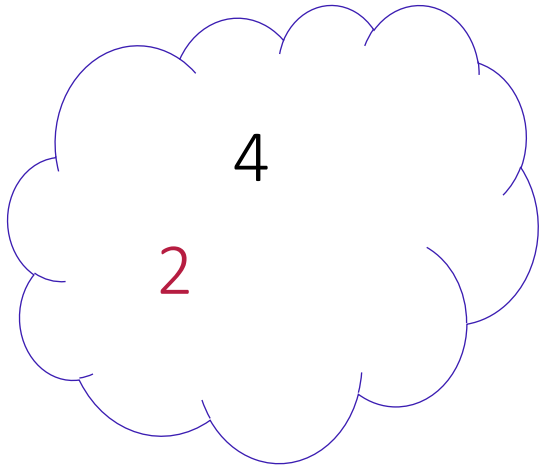
Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111



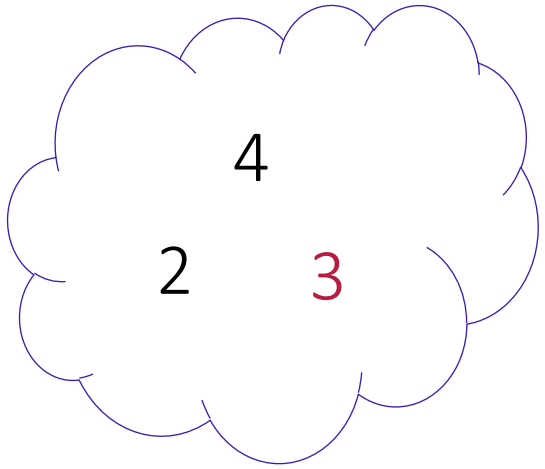
Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011



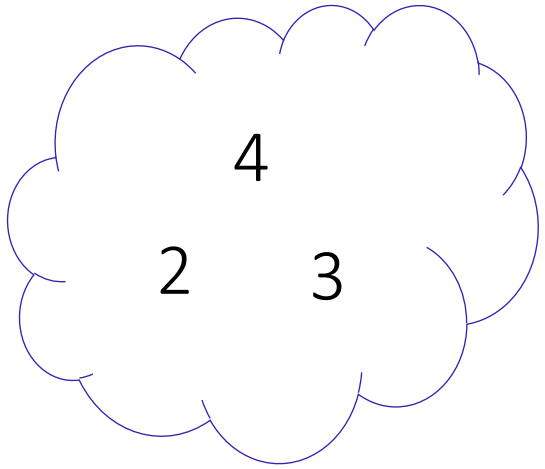
Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011
  - 1011



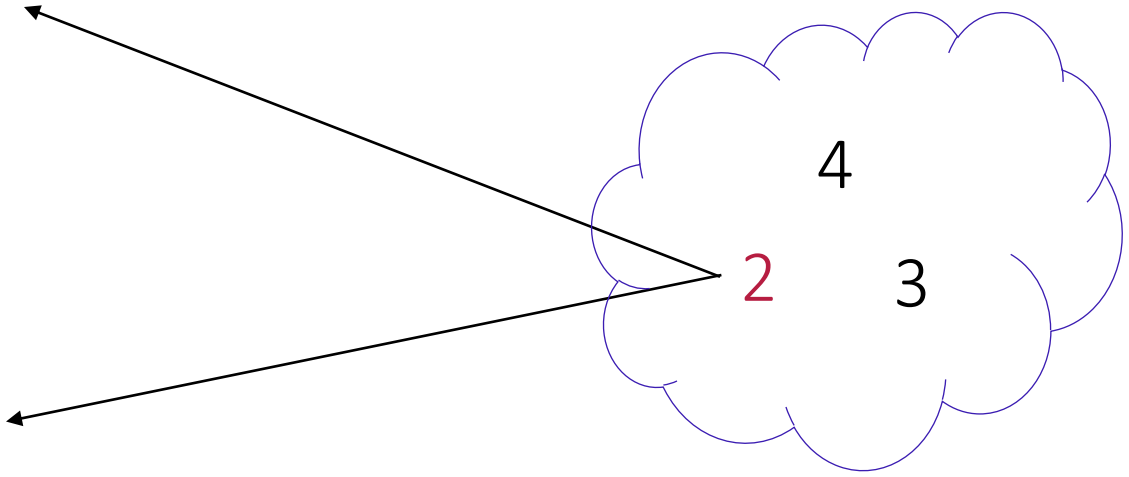
Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011
  - 1011
  - ~~1111~~



Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011
  - 1011
  - ~~1111~~
  - 0110

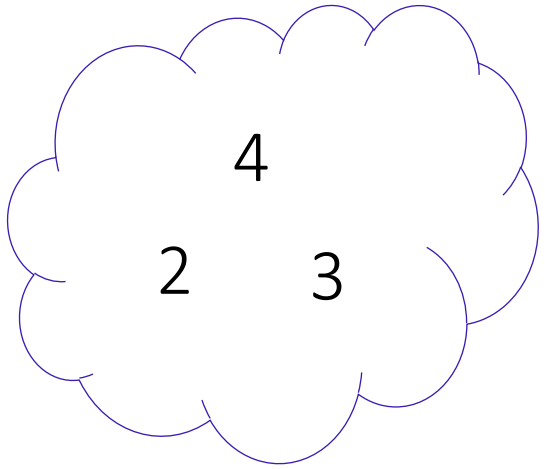


Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011
  - 1011
  - ~~1111~~
  - 0110

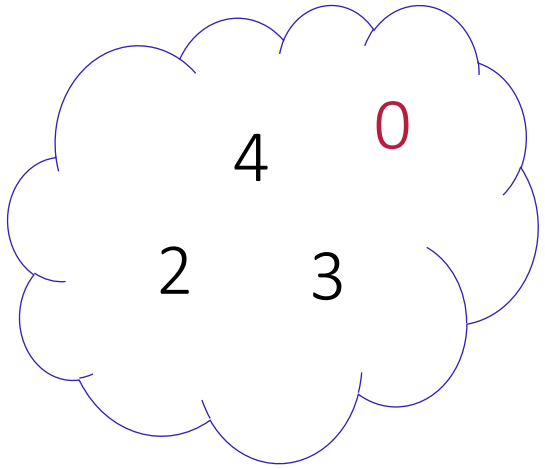
Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011
  - 1011
  - ~~1111~~
  - ~~0110~~



Implementação da Lista Tabu

- Tabelas de *Hash*
  - 1111
  - 0011
  - 1011
  - ~~1111~~
  - ~~0110~~
  - 0000



Implementação da Lista Tabu

- Características das soluções como Hash
  - S_0 = [C, A, B, C, B, A, D, C]

	1	2	3	4	5	6	7	8
A	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0

Implementação da Lista Tabu

- Características das soluções como Hash
  - S_0 = [C, A, B, C, B, A, D, C]
  - S_1 = [C, A, B, A, B, A, D, C]

	1	2	3	4	5	6	7	8
A	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0
C	0	0	0	5	0	0	0	0
D	0	0	0	0	0	0	0	0

Implementação da Lista Tabu

- Características das soluções como Hash
  - S_0 = [C, A, B, C, B, A, D, C]
  - S_1 = [C, A, B, A, B, A, D, C]
  - S_2 = [C, A, B, A, D, A, D, C]

	1	2	3	4	5	6	7	8
A	0	0	0	0	0	0	0	0
B	0	0	0	0	5	0	0	0
C	0	0	0	4	0	0	0	0
D	0	0	0	0	0	0	0	0

Implementação da Lista Tabu

- Características das soluções como Hash
  - S_0 = [C, A, B, C, B, A, D, C]
  - S_1 = [C, A, B, A, B, A, D, C]
  - S_2 = [C, A, B, A, D, A, D, C]
  - S_3 = [C, D, B, A, D, A, D, C]

	1	2	3	4	5	6	7	8
A	0	5	0	0	0	0	0	0
B	0	0	0	0	4	0	0	0
C	0	0	0	3	0	0	0	0
D	0	0	0	0	0	0	0	0

Implementação da Lista Tabu

- Características das soluções como Hash
  - S_1 = [C, A, B, A, B, A, D, C]
  - S_2 = [C, A, B, A, D, A, D, C]
  - S_3 = [C, D, B, A, D, A, D, C]
  - S_4 = [C, D, B, D, D, A, D, C]

	1	2	3	4	5	6	7	8
A	0	4	0	5	0	0	0	0
B	0	0	0	0	3	0	0	0
C	0	0	0	2	0	0	0	0
D	0	0	0	0	0	0	0	0

Implementação da Lista Tabu

- Características das soluções como Hash
  - S_2 = [C, A, B, A, D, A, D, C]
  - S_3 = [C, D, B, A, D, A, D, C]
  - S_4 = [C, D, B, D, D, A, D, C]
  - S_5 = [C, D, B, D, D, A, C, C]

	1	2	3	4	5	6	7	8
A	0	3	0	4	0	0	0	0
B	0	0	0	0	2	0	0	0
C	0	0	0	1	0	0	0	0
D	0	0	0	0	0	0	5	0

Implementação da Lista Tabu

- Características das soluções como Hash
  - S_3 = [C, D, B, A, D, A, D, C]
  - S_4 = [C, D, B, D, D, A, D, C]
  - S_5 = [C, D, B, D, D, A, C, C]
  - S_6 = [C, D, B, D, D, A, C, B]

	1	2	3	4	5	6	7	8
A	0	4	0	3	0	0	0	0
B	0	0	0	0	1	0	0	0
C	0	0	0	0	0	0	0	5
D	0	0	0	0	0	0	4	0

Critério de aspiração

- Move-se para o melhor vizinho mesmo que ele seja tabu
  - Caso ele seja a melhor solução conhecida
  - Quando todos os vizinhos são tabu

Considerações sobre a lista tabu

- Impacto do tamanho da lista
- Formas de computar o tamanho
  - Estático, Aleatório, ou Reativo
- Múltiplas listas tabu

Mecanismos de Memória

- Curto prazo
  - Lista tabu
- Médio prazo
  - Utilizada para fazer intensificação
- Longo prazo
  - Utilizada para fazer diversificação

Memória de Médio Prazo

- Utilizada para fazer intensificação
- Enviesa a busca na direção de um conjunto de soluções elite

Memória de Médio Prazo: Exemplo

- Seja $\mathcal{H} \subseteq \Gamma$ o conjunto de soluções elite
- r_i : frequência do elemento *i* nas soluções de $\mathcal{H}$

Memória de Médio Prazo: Exemplo

- Altera a função objetivo de acordo com r_i
  - $\min\ F(x) = \sum_{i \in I} c_i x_i$
  - $\min\ F(x) = \sum_{i \in I} (c_i - g(r_i)) x_i$

Memória de Longo Prazo

- Utilizada para fazer diversificação
- Enviesa a busca na direção oposta das soluções já amostradas

Memória de Longo Prazo: Exemplo

- Seja $\mathcal{B} \subseteq \Gamma$ o conjunto amostradas até então
- q_i : frequência do elemento *i* nas soluções de $\mathcal{B}$

Memória de Longo Prazo: Exemplo

- Altera a função objetivo de acordo com q_i
  - $\min\ F(x) = \sum_{i \in I} c_i x_i$
  - $\min\ F(x) = \sum_{i \in I} (c_i + f(q_i)) x_i$

Considerações Finais

- Intensificação
  - Memória de médio prazo
- Diversificação
  - Memórias de curto e longo prazo

Considerações Finais

- Os mecanismos de memória são dependentes do problema
- Só encontra soluções na mesma componente conexa da solução inicial

GRASP

Greedy Randomized Adaptative Search Procedure

Princípio

O que fazer quando chegar em um

Ótimo Local?

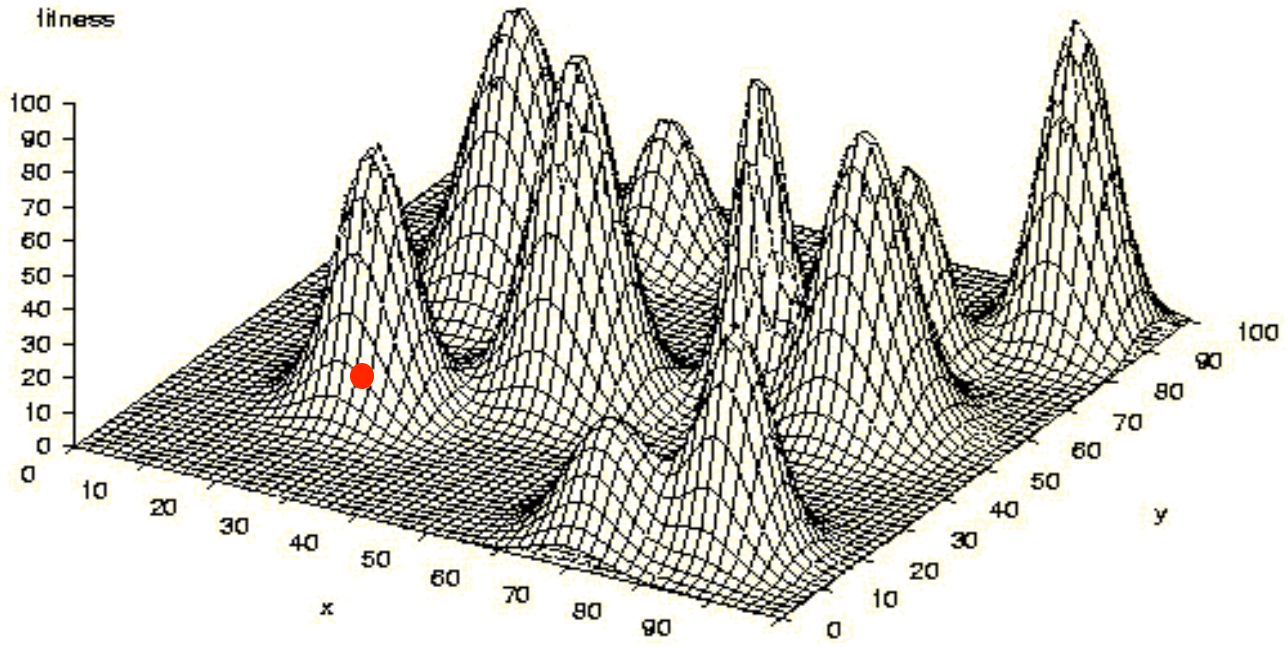
Princípio

Reinicie a partir de uma solução diferente

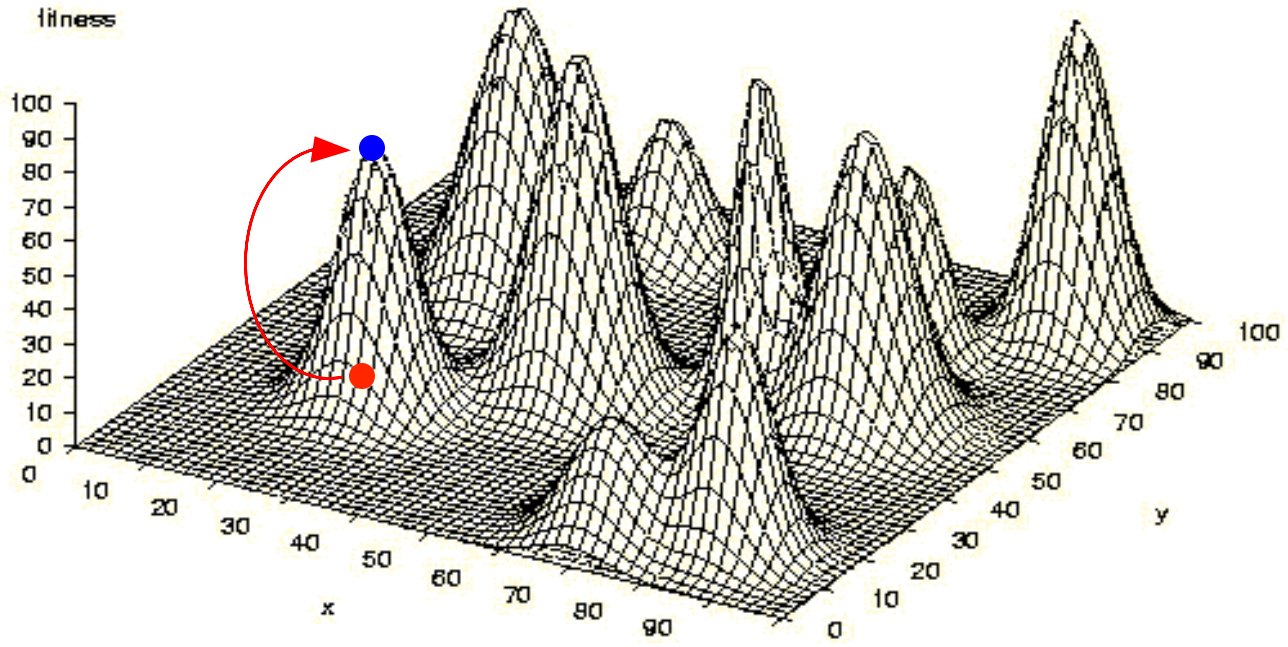
Heurísticas de Multipartida

- Algoritmos estocásticos
- A cada iteração gera uma solução potencialmente diferente
- Retorna a melhor solução amostrada

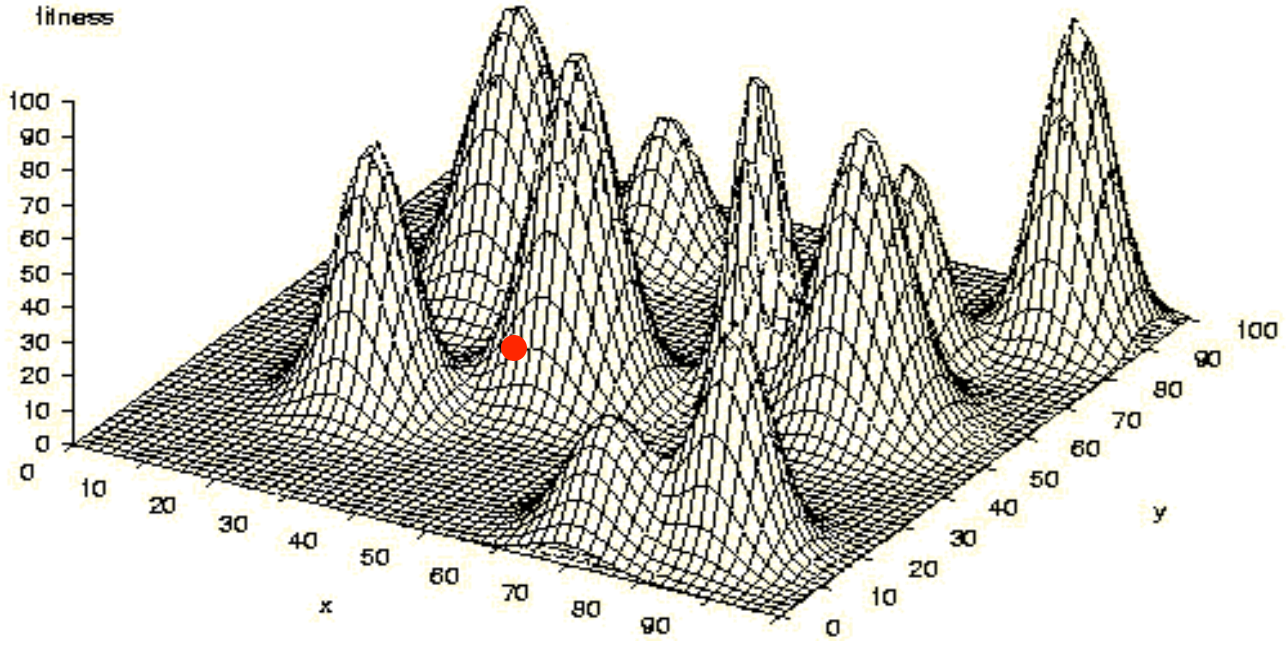
Heurísticas de Multipartida



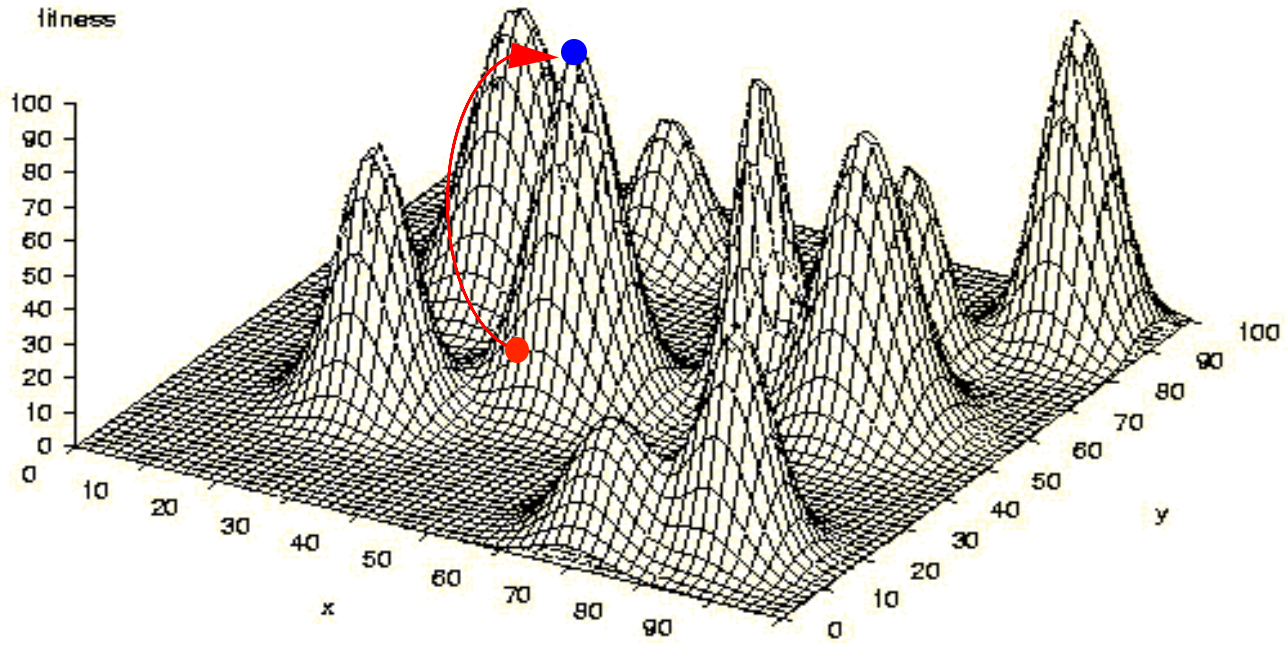
Heurísticas de Multipartida



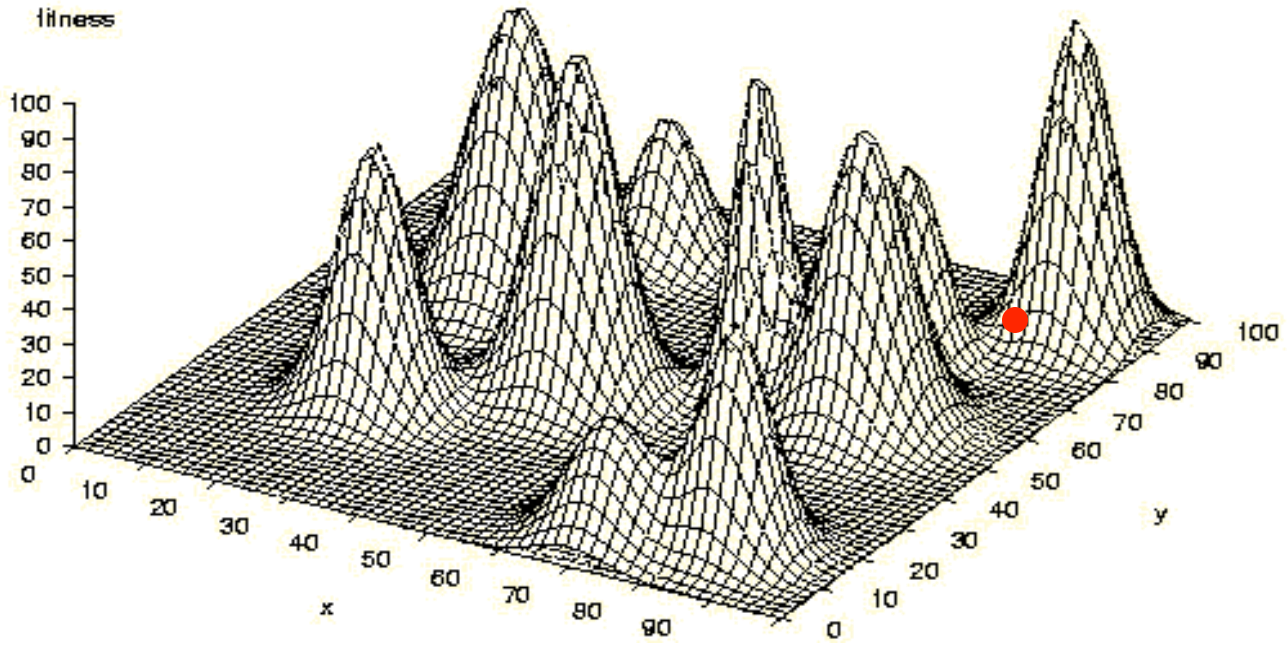
Heurísticas de Multipartida



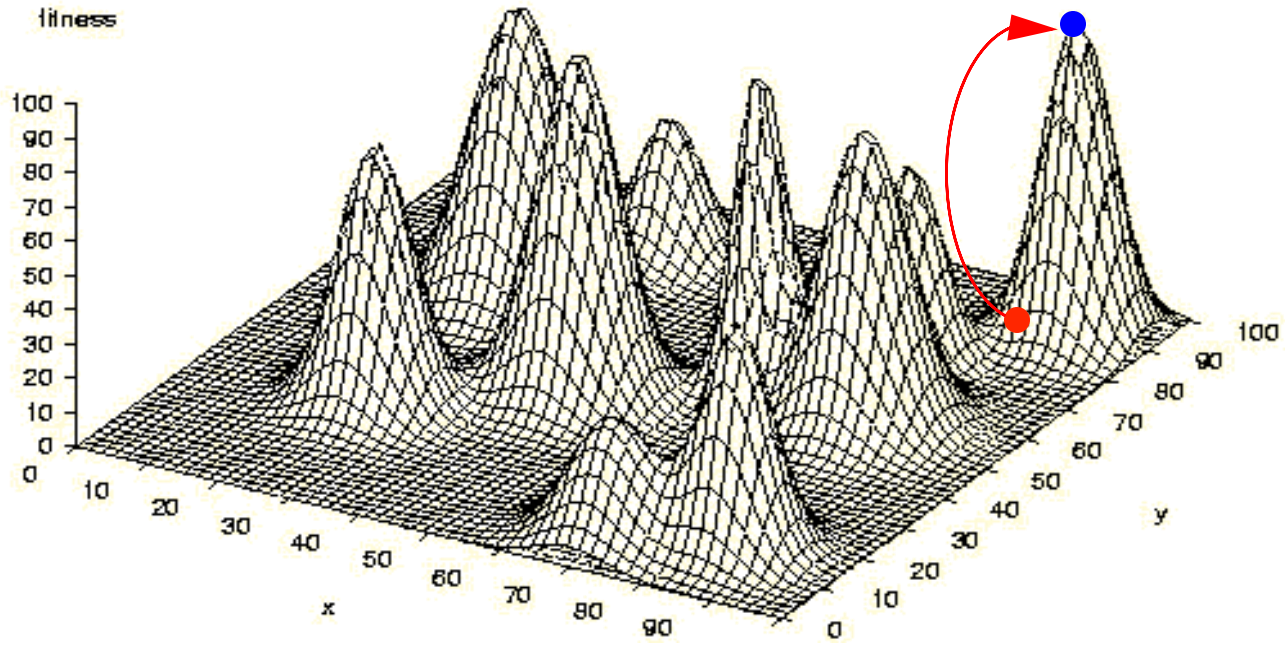
Heurísticas de Multipartida



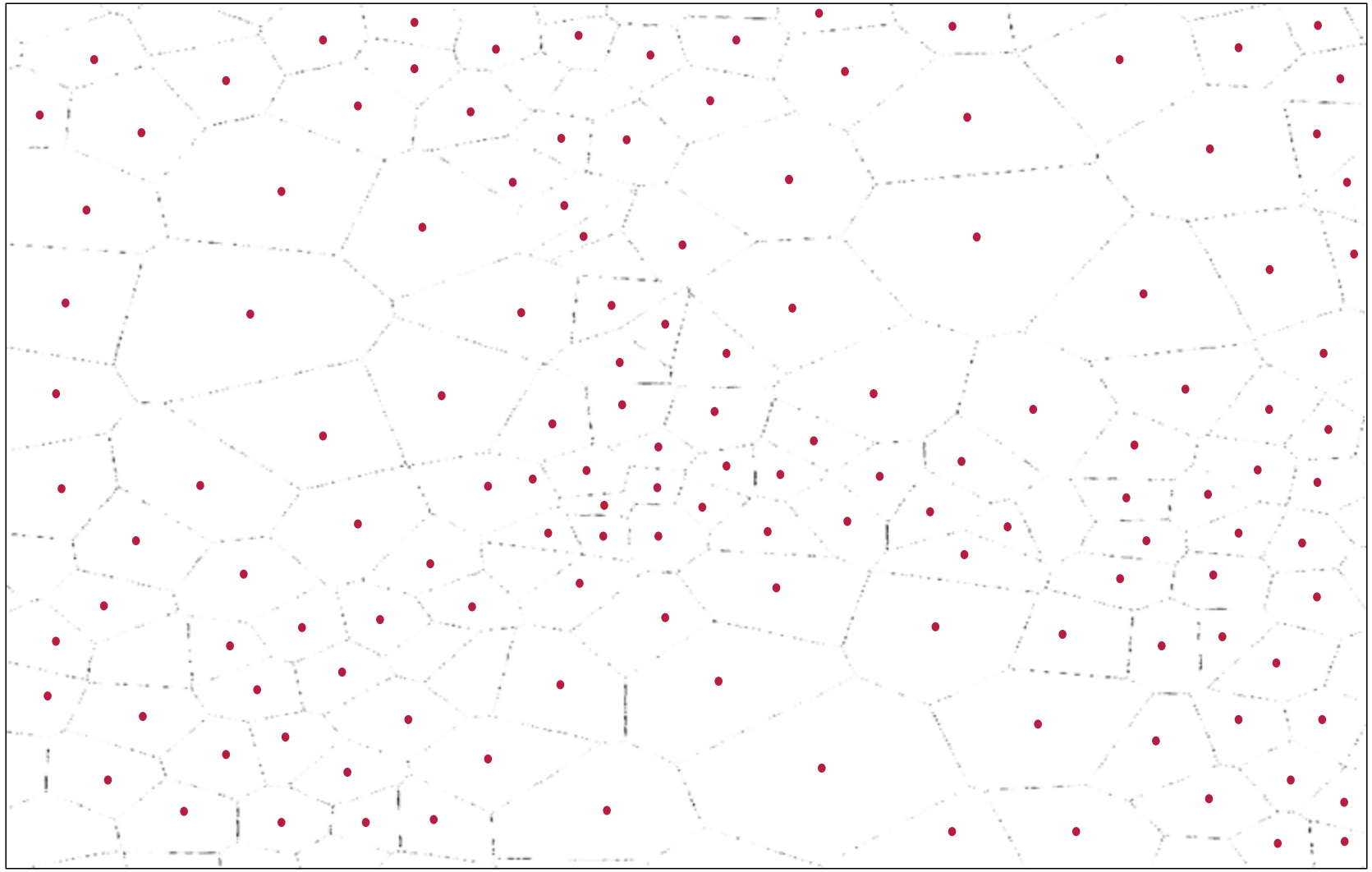
Heurísticas de Multipartida



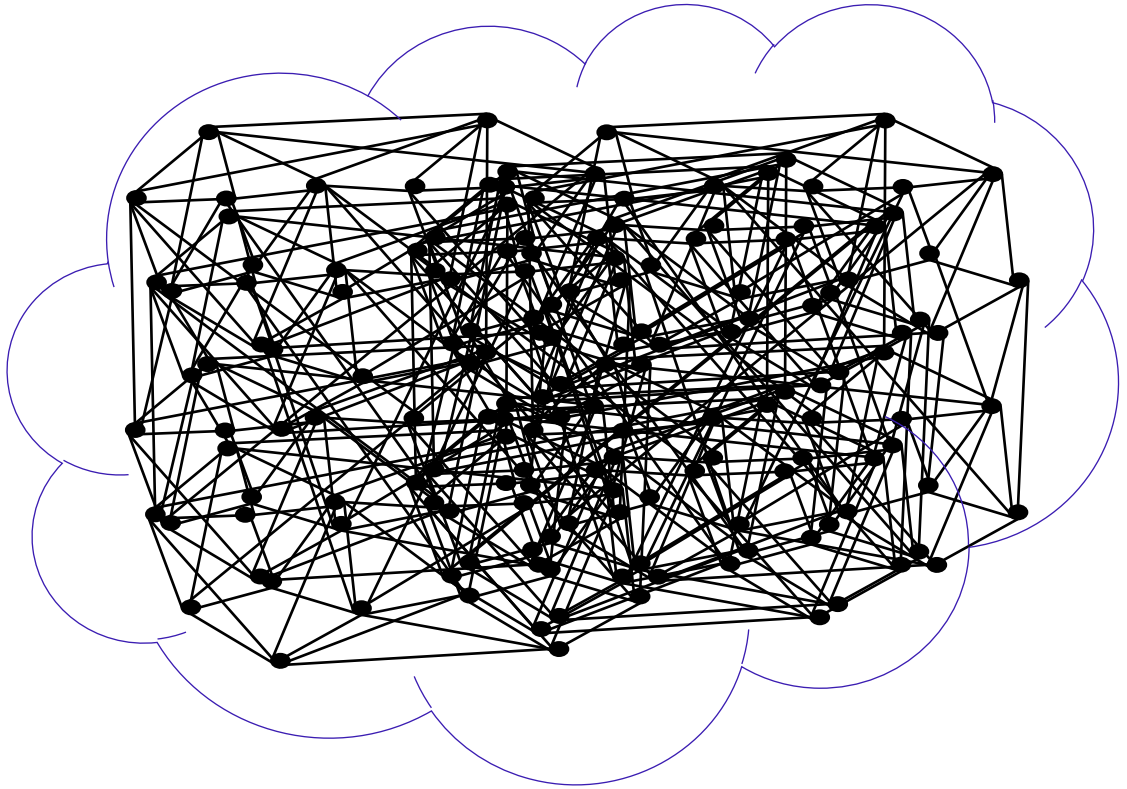
Heurísticas de Multipartida



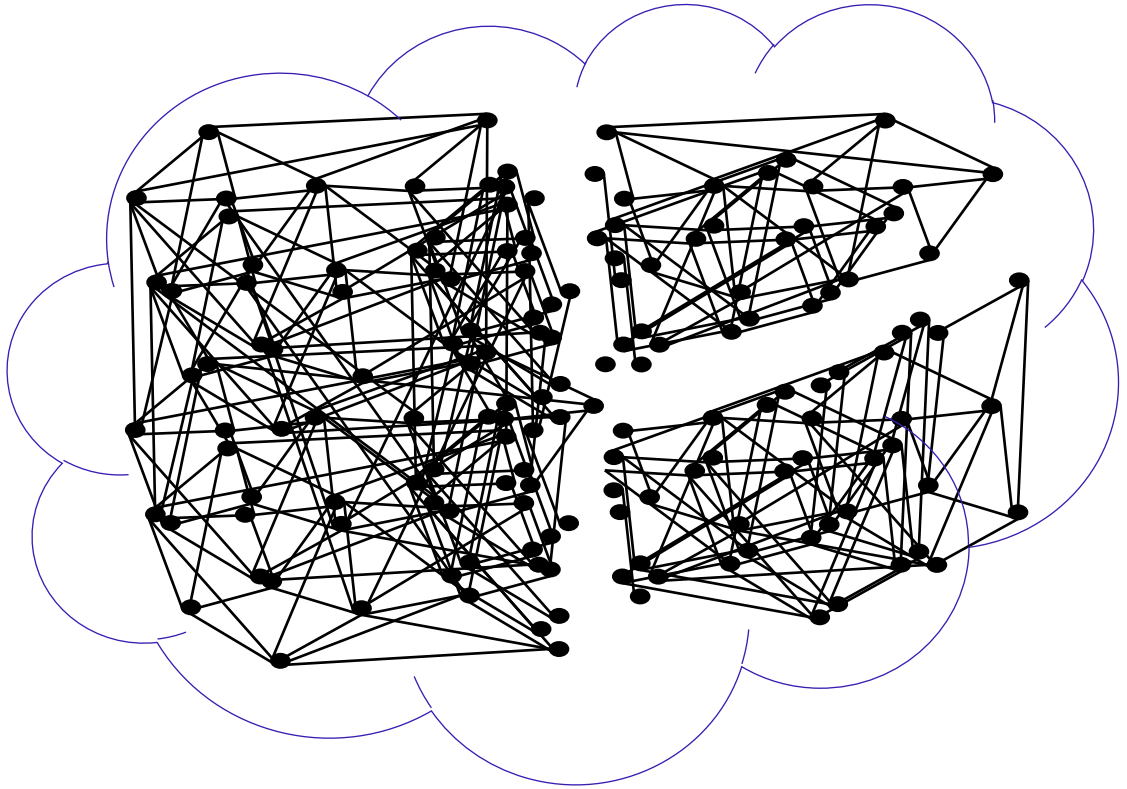
Regiões de atração



Grafo de Vizinhança



Grafo de Vizinhança



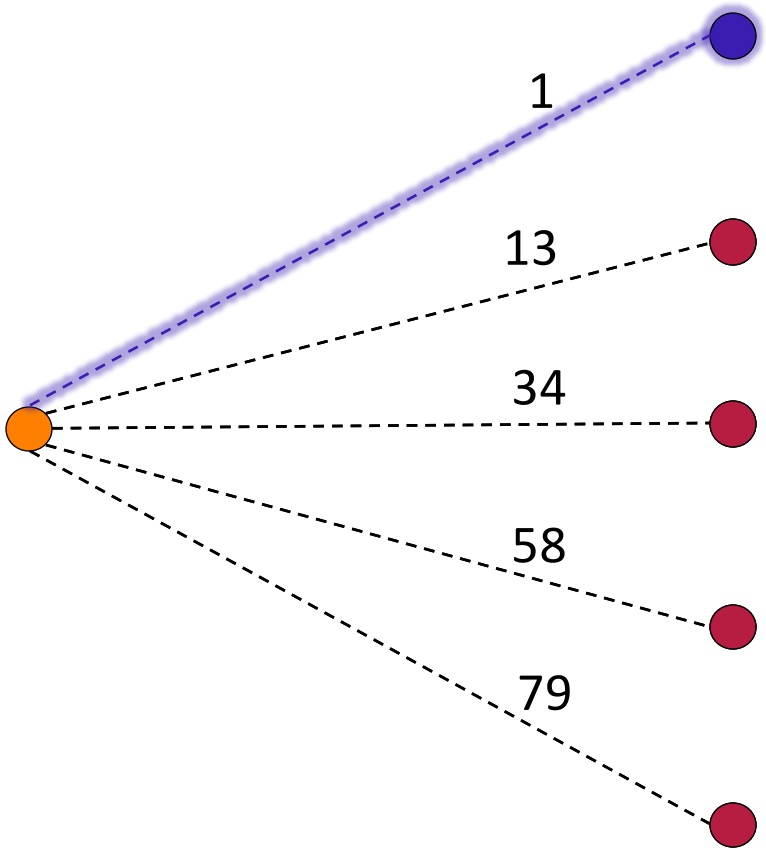
Heurísticas de Multipartida

- A cada iteração
  - Fase construtiva
    - Constrói uma solução inicial
  - Fase de busca local
    - Amostra o ótimo local que atrai esta solução

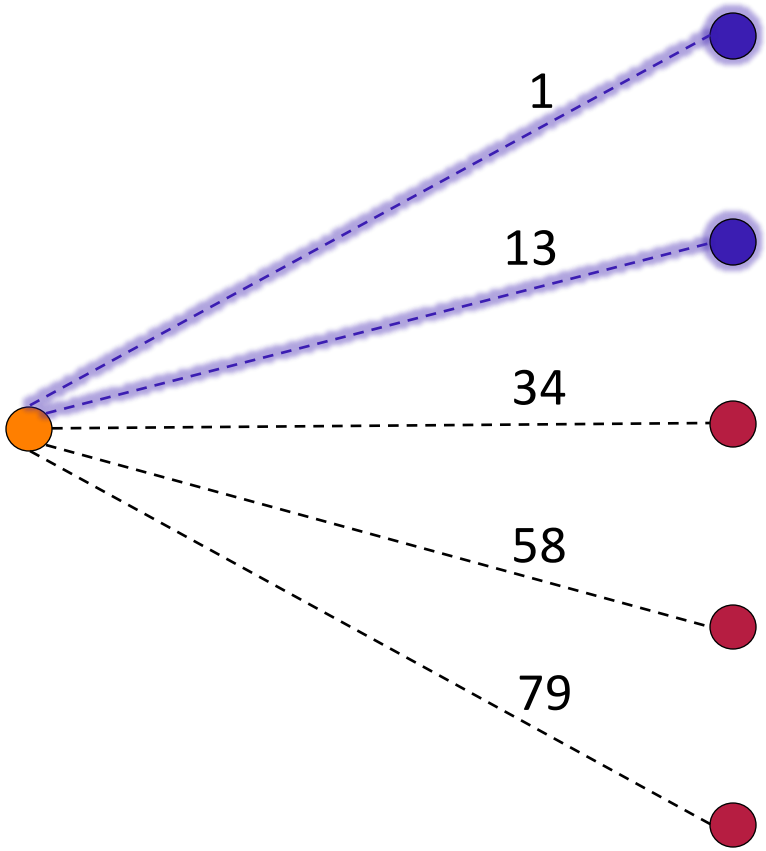
Greedy randomized adaptive search procedure

- A cada iteração
  - Fase construtiva
    - Constrói uma solução inicial com de uma heurística gulosa aleatorizada e adaptativa
  - Fase de busca local
    - Amostra o ótimo local que atrai esta solução

Heurísticas Gulosas Aleatorizadas



Heurísticas Gulosas Aleatorizadas



Heurísticas Gulosas Aleatorizadas

Utilizam um mecanismo de aleatorização na função gulosa

Heurísticas Gulosas Aleatorizadas

Decisões erradas não necessariamente,
são repetidas nas iterações subsequentes

Heurísticas Gulosas Aleatorizadas

- Cria-se uma lista de candidatos
  - melhores de acordo com a função gulosa
- Retorna um elemento desta lista
  - Escolhido aleatoriamente

Heurísticas Gulosas Aleatorizadas

A diversidade das soluções depende da cardinalidade da lista de candidatos

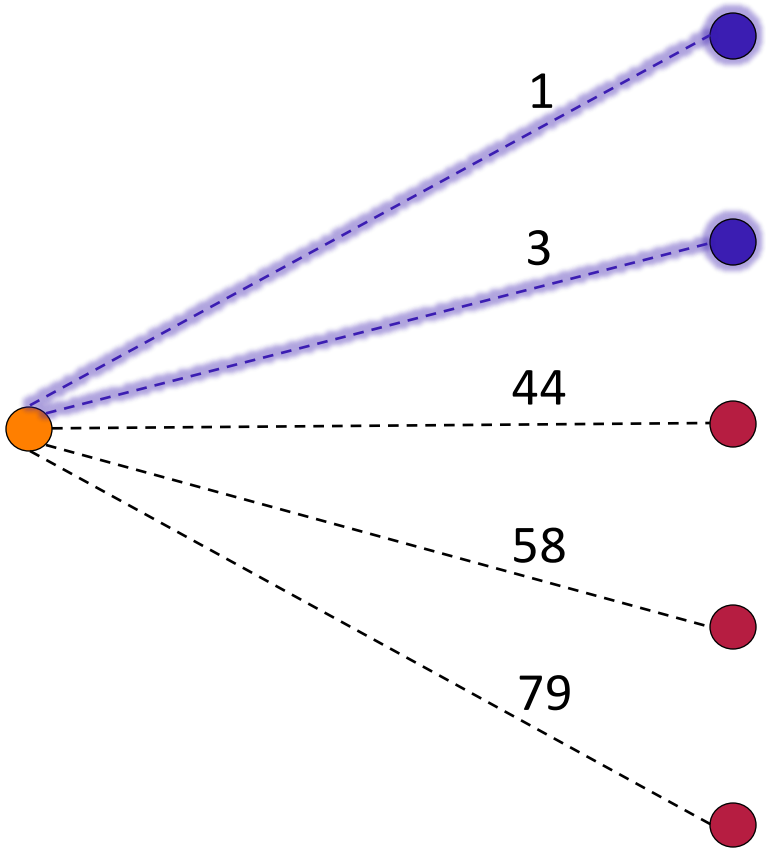
Heurísticas Gulosas Aleatorizadas

- Casos extremos
  - Algoritmo guloso puro ($|L| = 1$)
  - Solução aleatória ($|L| = |C|$)

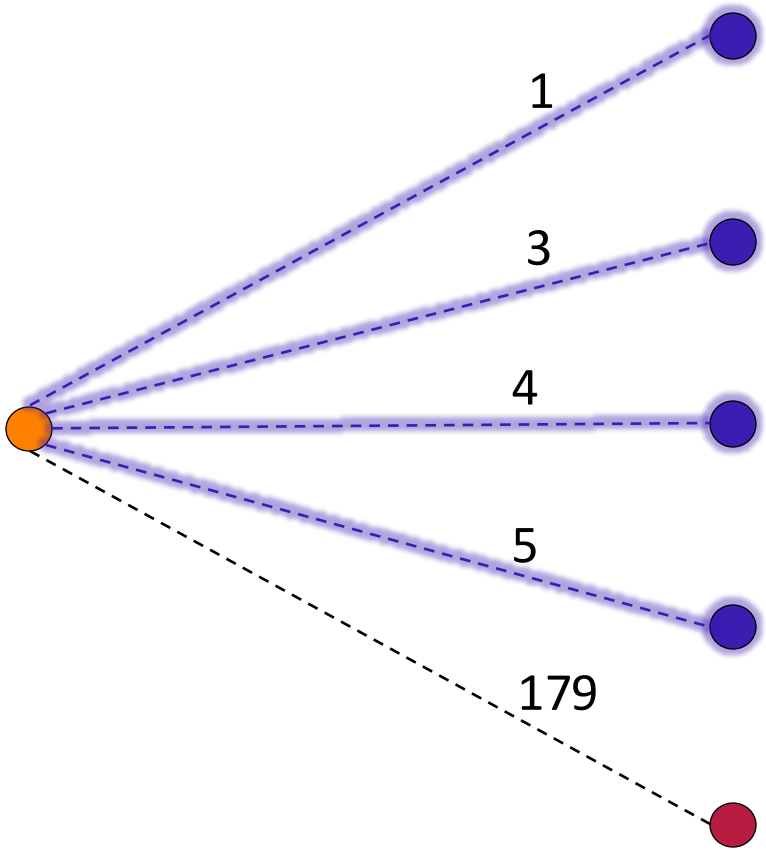
Heurísticas Gulosas Aleatorizadas Adaptativas

A cardinalidade da lista de candidatos varia a cada iteração da heurística

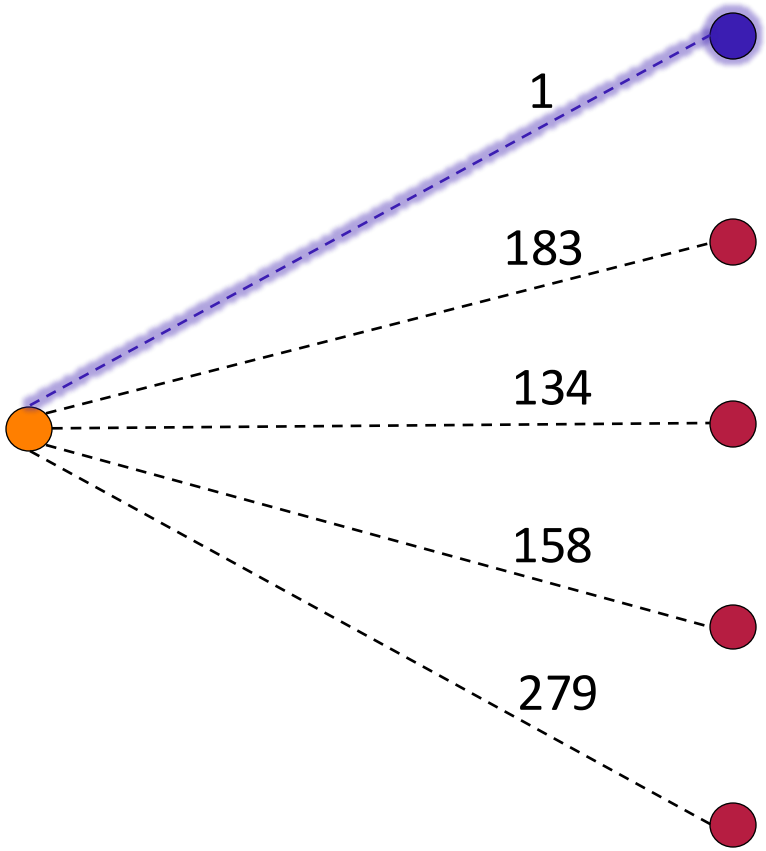
Heurísticas Gulosas Aleatorizadas



Heurísticas Gulosas Aleatorizadas



Heurísticas Gulosas Aleatorizadas



Greedy randomized adaptive search procedure

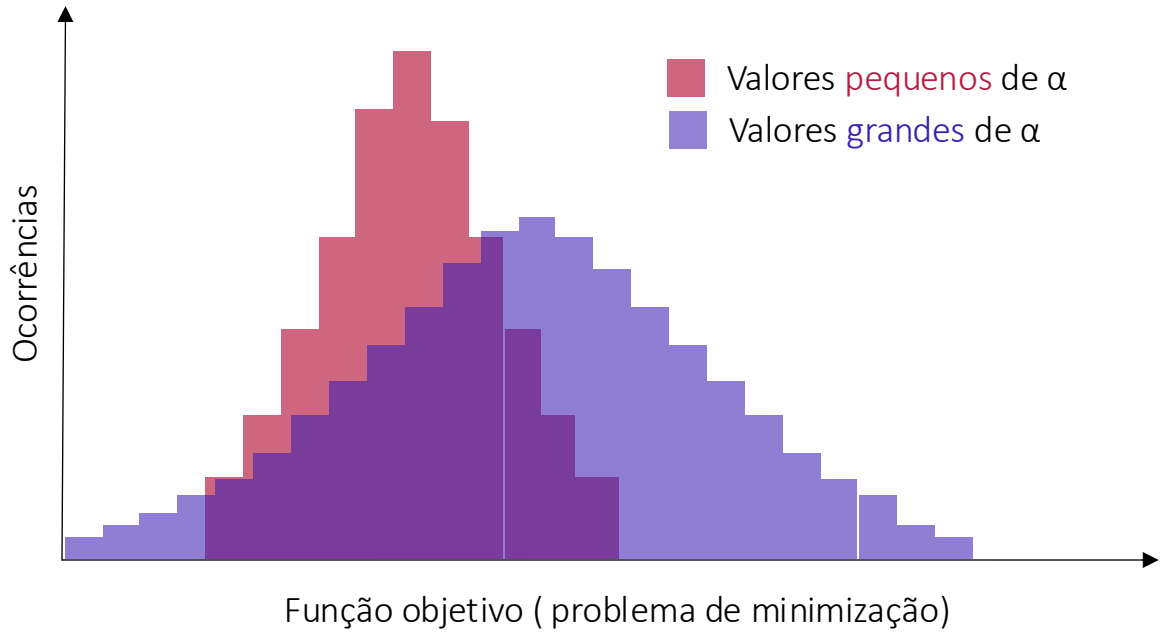
- A cada iteração
  - Fase construtiva
    - Constrói uma solução inicial com de uma heurística gulosa aleatorizada e adaptativa
  - Fase de busca local
    - Amostra o ótimo local que atrai esta solução

Cardinalidade Adaptativa

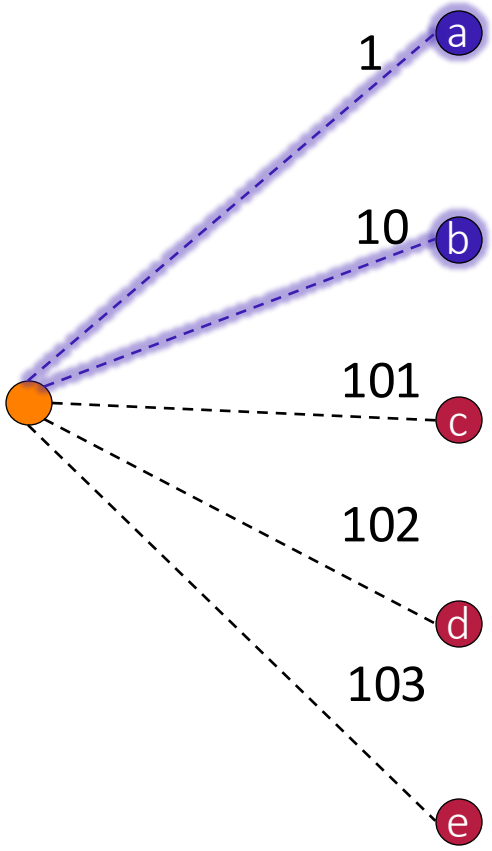
- Parâmetro $0 \leq \alpha \leq 1$
  - $c^{min} = \min_{e \in E} c(e)$
  - $c^{max} = \max_{e \in E} c(e)$
  - $LRC = \{e \in E: c(e) \leq c^{min} + \alpha\left(c^{max} - c^{min}\right)\}$

Histograma

- da qualidade das soluções em função do valor de α



Cardinalidade Adaptativa



- $LRC = \{e \in E : c(e) \leq c^{min} + \alpha\left(c^{max} - c^{min}\right)\}$

$\propto = 0{,}5$

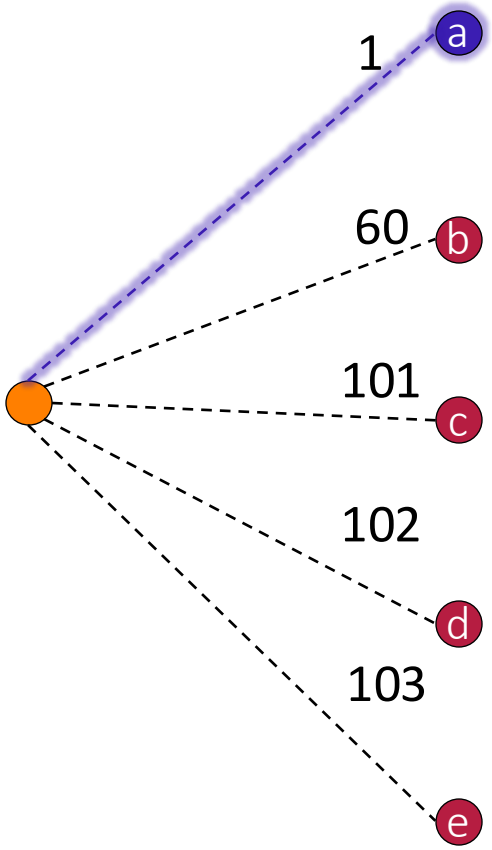
$c^{min} = 1$

$c^{min} = 103$

$LRC = \{e \in E : c(e) \leq 52\}$

$LRC = \{a, b\}$

Cardinalidade Adaptativa



- $LRC = \{e \in E : c(e) \leq c^{min} + \alpha(c^{max} - c^{min})\}$

$\propto = 0{,}5$

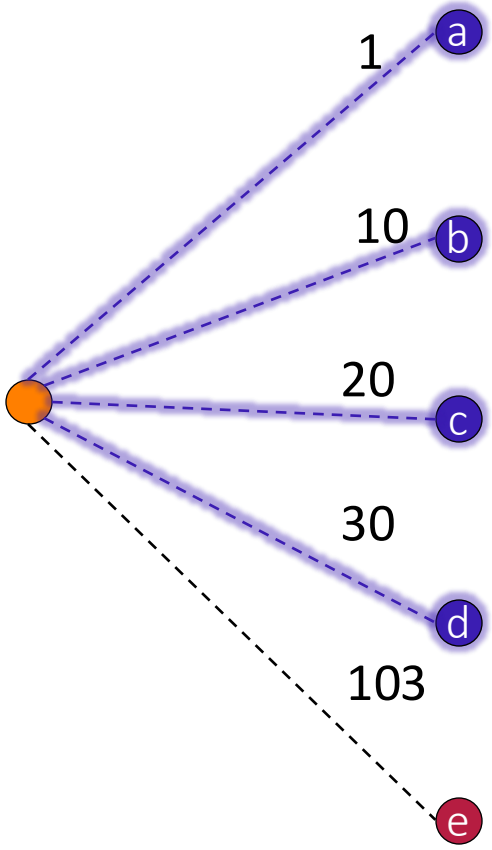
$c^{min} = 1$

$c^{min} = 103$

$LRC = \{e \in E : c(e) \leq 52\}$

$LRC = \{a\}$

Cardinalidade Adaptativa



- $LRC = \{e \in E : c(e) \leq c^{min} + \alpha(c^{max} - c^{min})\}$

$\propto = 0{,}5$

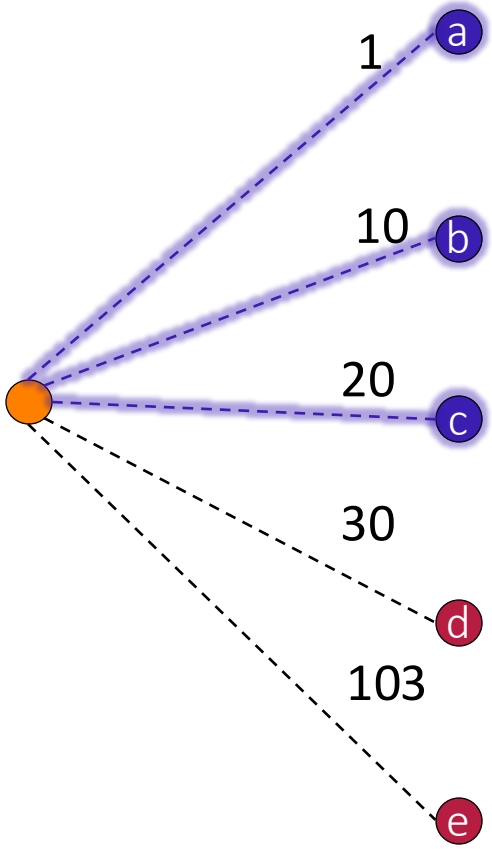
$c^{min} = 1$

$c^{min} = 103$

$LRC = \{e \in E : c(e) \leq 52\}$

$LRC = \{a, b, c, d\}$

Cardinalidade Adaptativa



- $LRC = \{e \in E : c(e) \leq c^{min} + \alpha(c^{max} - c^{min})\}$

$\propto = 0{,}25$

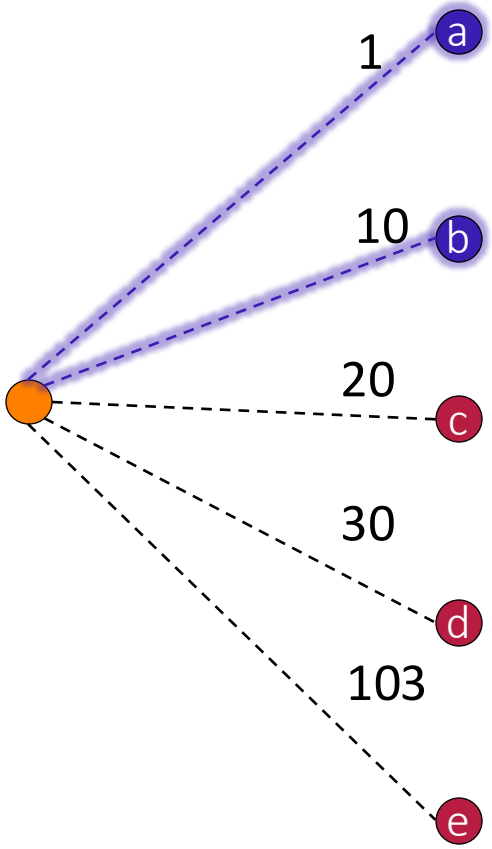
$c^{min} = 1$

$c^{min} = 103$

$LRC = \{e \in E : c(e) \leq 26{,}5\}$

$LRC = \{a, b, c\}$

Cardinalidade Adaptativa



- $LRC = \{e \in E : c(e) \leq c^{min} + \alpha(c^{max} - c^{min})\}$

$\propto = 0{,}1$

$c^{min} = 1$

$c^{min} = 103$

$LRC = \{e \in E : c(e) \leq 11{,}2\}$

$LRC = \{a, b\}$

GRASP Reativo

- Prais & Ribeiro (2000)

- A cada iteração
  - $\propto$ é escolhido aleatoriamente a partir de um conjunto de valores
    - $\{\propto_1, \cdots, \propto_m\}, \quad i \in \{1, \cdots, m\}$
  - Cada valor está associado a uma probabilidade
    - $\{p_1, \cdots, p_m\}$

GRASP Reativo

- Inicialmente os valores $\propto$ são equiprováveis
  - $p_i = \frac{1}{m}$
- Mas são atualizadas periodicamente
  - de modo a favorecer valores de $\propto$ que levam a melhores soluções

GRASP Reativo

- Atualiza a cada k iterações
  - $p_i = \frac{q_i}{\sum_{j=1}^{m} q_j}, \qquad i \in \{1, \cdots, m\}$
    - Onde, $q_i = \left(\frac{z^*}{A_i}\right)^{\delta}$

Filtro de Hashing

- Woodruff & Zemel (1993)

- Armazena todas as soluções iniciais
- Não aplica busca local a estas soluções

Filtro de qualidade

- Feo, Resende, & Smith (1994)

- Armazena a melhor solução conhecida
- Não aplica busca local a soluções baixa qualidade

Considerações Finais

- Intensificação
  - Busca local
- Diversificação
  - Heurística Gulosa Aleatorizada
- Memória
  - Utilizada para computar o valor de α

Variable Neighborhood Search

Generalização de VND

Princípio

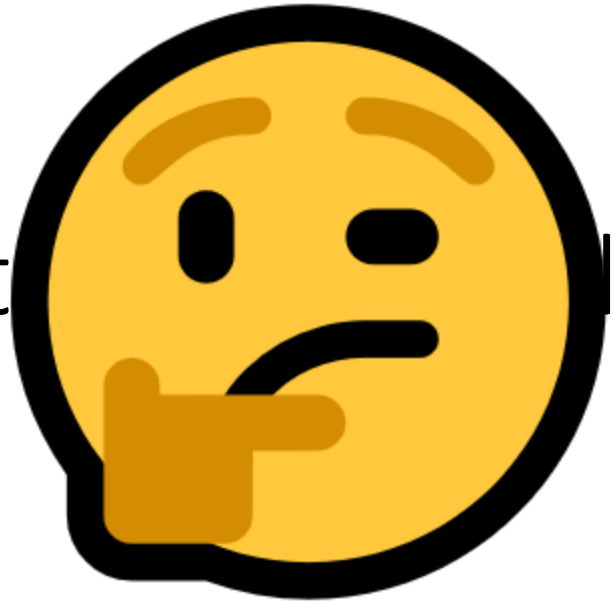
O que fazer quando chegar em um

Ótimo Local?

Princípio

Reinicie a partir de uma solução diferente

Princípio

Reinicie a part  lução diferente

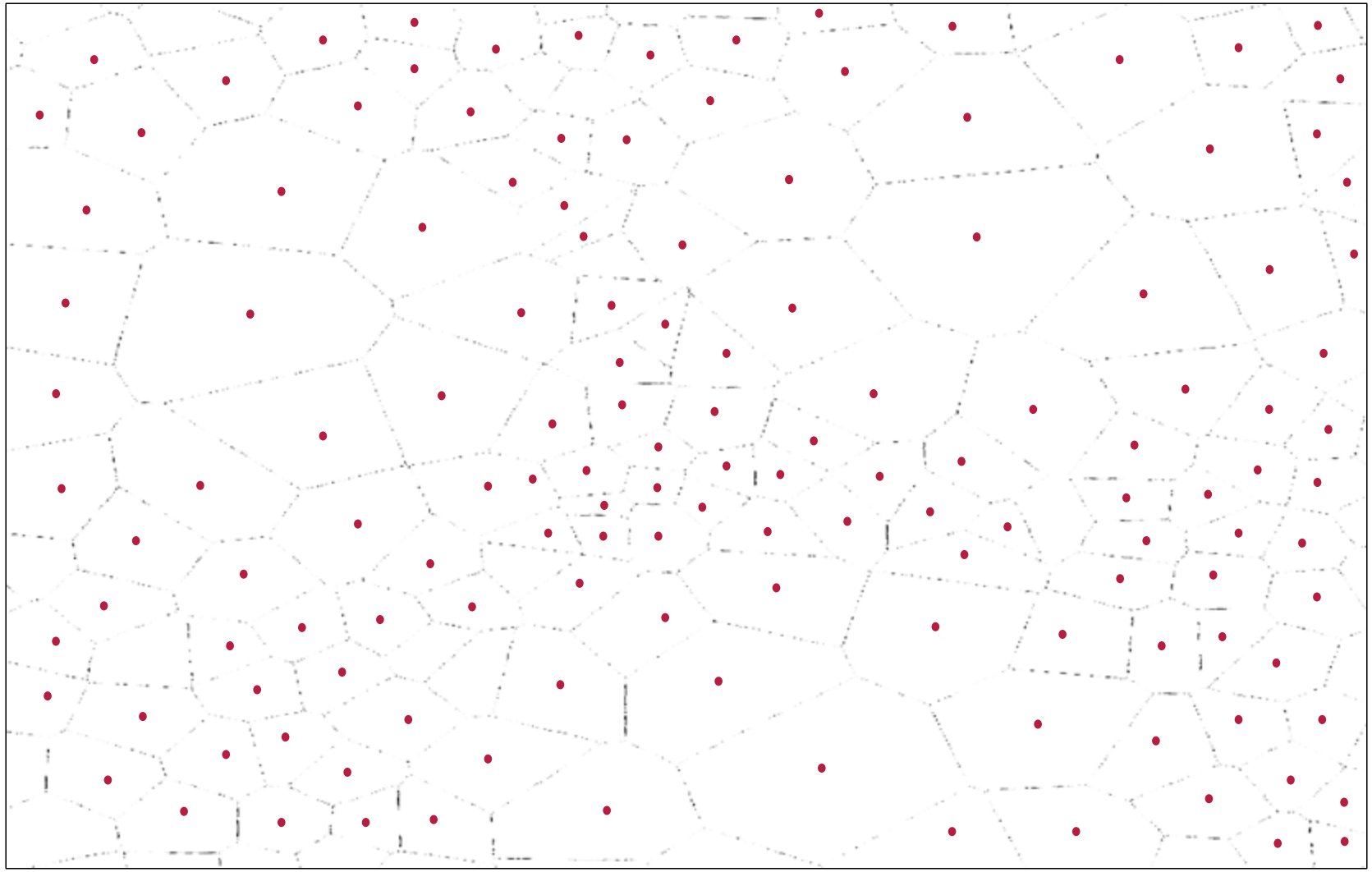
Princípio

Reinicia a partir de uma solução diferente,
mas parecida com o ótimo local

Princípio

Utiliza uma estratégia de destruir-e-reconstruir

Regiões de atração

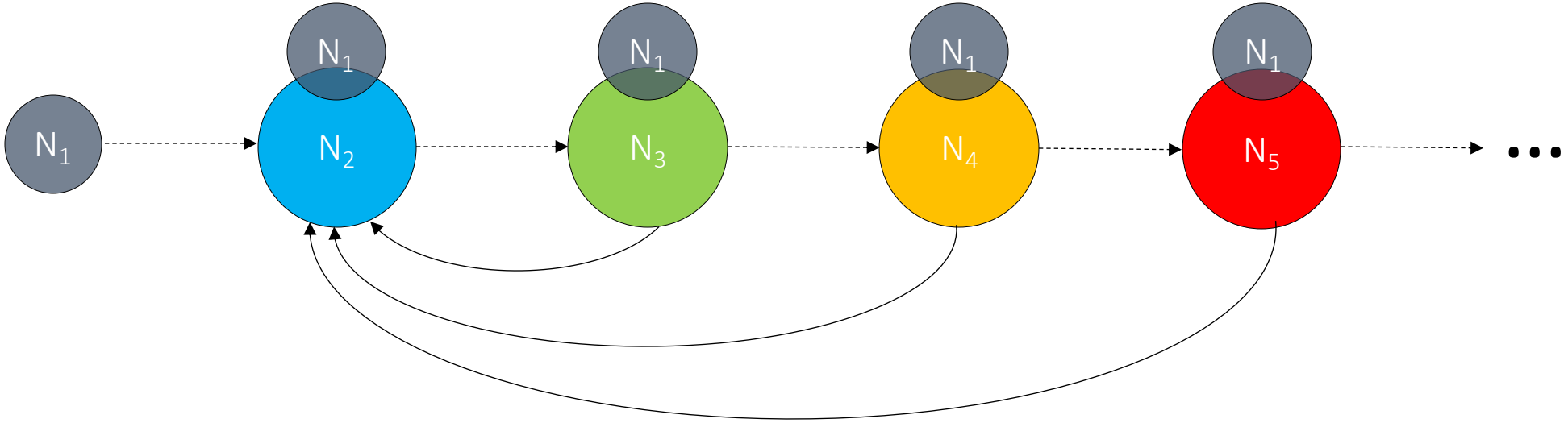


Greedy randomized adaptive search procedure

- A cada iteração
  - Fase construtiva
    - Constrói uma solução inicial com de uma heurística gulosa aleatorizada e adaptativa
  - Fase de busca local
    - Amostra o ótimo local que atrai esta solução

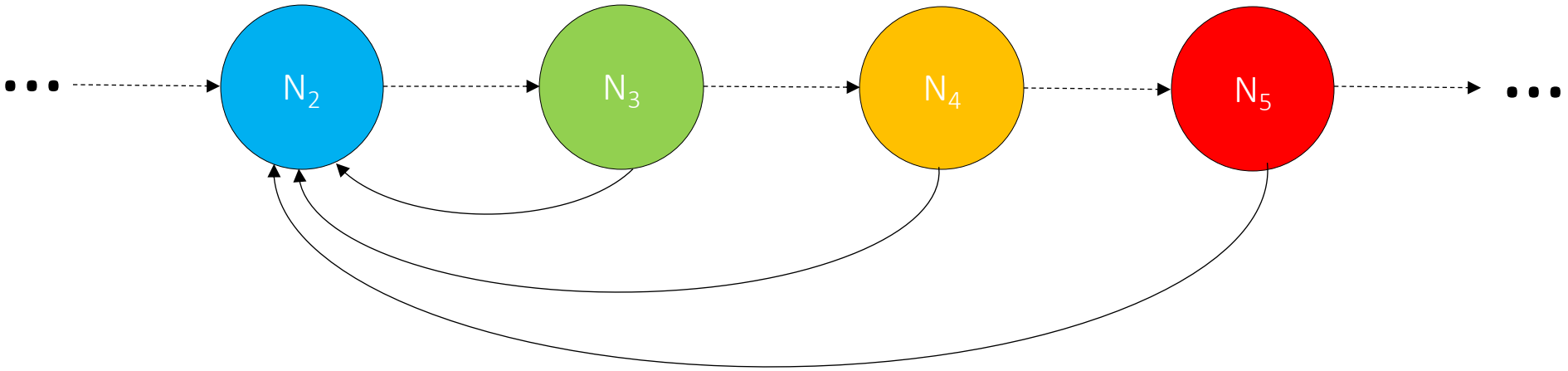
Variable Neighborhood Search

- Faz busca local em uma única vizinhança



Lembra um VND

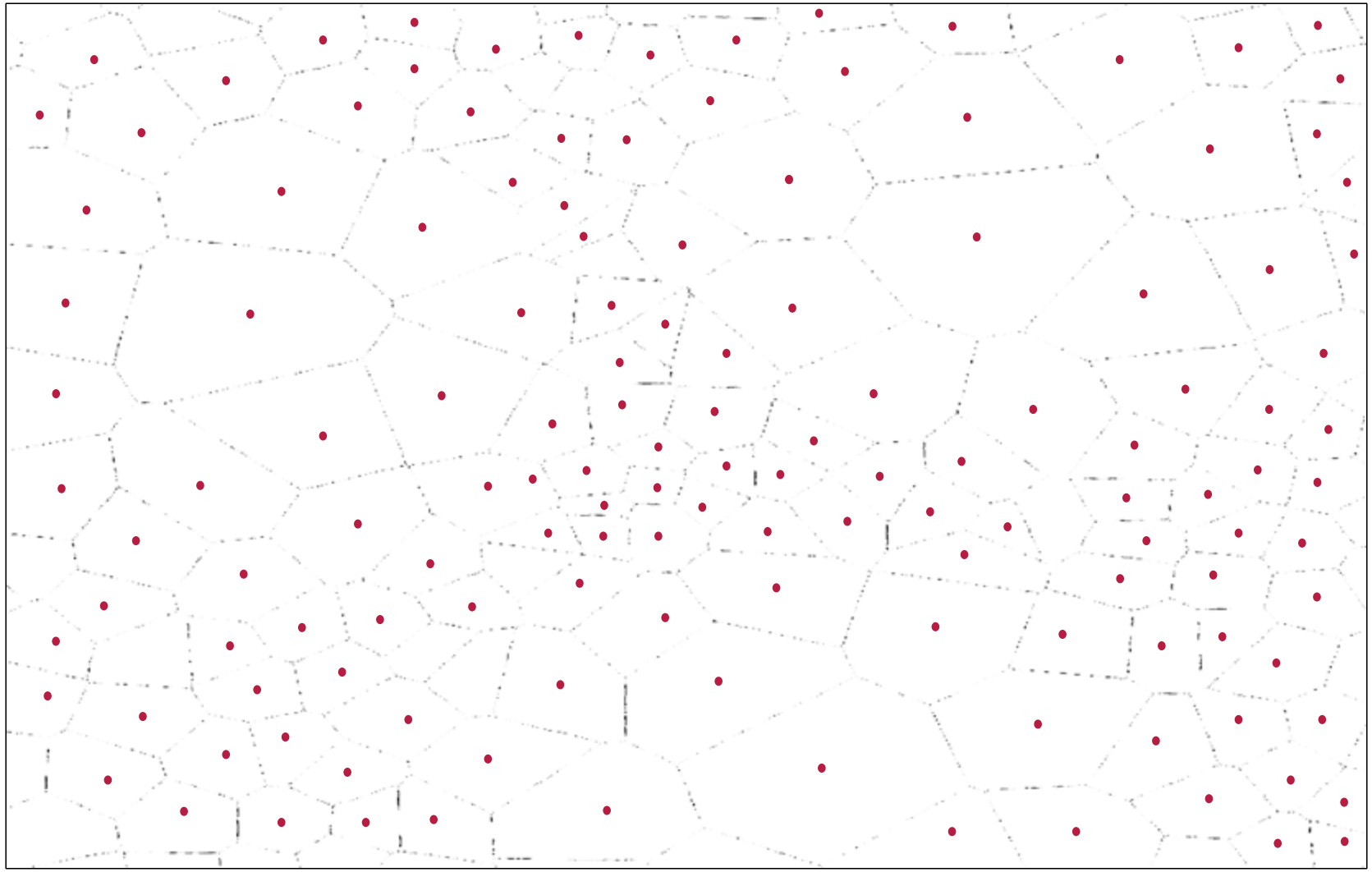
- Mas VND faz busca local em todas as vizinhanças



Consideração final

Se comporta como uma busca local
no espaço de ótimo locais

Regiões de atração



Metaheurísticas de busca em vizinhança

Thiago Noronha – tfn@dcc.ufmg.br