METAHEURÍSTICAS

METAHEURÍSTICA → Encontrar soluções boas ou até mesmo ótima

- Consiste em aplicar iterativamente uma heurística subordinada (busca local)
- Tem algum mecanismo para escapar de ótimos locais (vales)

Se dividem basicamente em duas categorias:

METAHEURÍSTICAS COM UMA ÚNICA SOLUÇÃO (GERA UMA SEQUENCIA DE SOLUÇÕES): explora o espaço das soluções por meio de movimentos aplicados a cada passo sobre uma solução corrente. Normalmente mantém uma única solução corrente.

MULT-START

GRASP - Greedy Randomized Adaptive Search Procedure

VNS – Variable Neighborhood Search

ILS – Iterated Loal Search

BUSCA TABU - Tabu Search

Simulated Annealing

METAHEURÍSTICA → Encontrar soluções boas ou até mesmo ótimas

- Consiste em aplicar iterativamente uma heurística subordinada (busca local)
- Tem algum mecanismo para escapar de ótimos locais (vales)

Se dividem basicamente em duas categorias:

METAHEURÍSTICAS COM VÁRIAS SOLUÇÕES (BUSCA POPULACIONAL): mantém um conjunto de boas soluções e as combina para produzir soluções ainda melhores. Normalmente não realizam procedimento de refinamento das soluções, ou sejam, de busca local. Deve manter um conjunto de soluções correntes.

ALGORITMOS GENÉTICOS

COLÔNIA DE FORMIGAS

COLÔNIA DE ABELHAS

BBO - BIOGEOGRAPHY-BASED-OPTIMIZATION

METAHEURÍSTICA COM UMA ÚNICA SOLUÇÃO

As metaheurísticas se diferenciam pelos seguintes aspectos:

1. Critério de escolha da solução inicial
2. Definição da vizinhança N(s) de uma solução s
3. Critério de seleção de uma solução vizinha em N(s)
4. Estratégia adotada para escapar de ótimos locais. Principal característica que diferencia duas metaheurísticas desta classe.

METAHEURÍSTICA COM UMA ÚNICA SOLUÇÃO

1. Critério de escolha da solução inicial
    - *Gulosa, pseudo-gulosa, randômica*
2. Definição da vizinhança N(s) de uma solução s
    - *usa uma única vizinhança, ou várias vizinhanças*
3. Critério de seleção de uma solução vizinha em N(s)
    - *melhor vizinho, primeiro melhor, vizinho aleatório*
4. Critério de parada
    - *tempo CPU, núm. de iterações sem melhora, % de melhoria*

Mult-Start

Faz uma amostragem do espaço de soluções iniciais, gerando-as randomicamente. Para cada solução inicial aplica um método de refinamento/busca local.

```
procedimento MultiStart(f(.), N(.), CriterioParada, s)
1   s* ← s;                    {Melhor solução até então}
2   f* ← ∞;                    {Valor associado a s* }
3   enquanto (Critério de parada não atendido) faça
4       s ← ConstruaSolucao();  {Gere uma solução s do espaço de soluções}
5       s ← BuscaLocal(s);      {Aplique um procedimento de melhora em s}
6       se (f(s) < f(s*)) então
7           s* ← s;
8           f* ← f(s);
9       fim-se;
10  fim-enquanto;
11  s ← s*;
12  Retorne s;
fim MultiStart;
```

Greedy Randomized Adaptative Search Procedure

Tem sua ênfase na construção da solução inicial pseudo-gulosa.

A escolha de um novo elemento a compor a solução é feita aleatoriamente dentro de uma lista das α melhores soluções.

```
procedimento GRASP(f(.), g(.), N(.), GRASPmax, s)
1  f* ← ∞;
2  para (Iter = 1, 2, ..., GRASPmax) faça
3      Construcao(g(.), α, s);
4      BuscaLocal(f(.), N(.), s);
5      se (f(s) < f*) então
6          s* ← s;
7          f* ← f(s);
8      fim-se;
9  fim-para;
10 s ← s*;
11 Retorne s;
fim GRASP
```

Fase de Construção do GRASP p/ minimização

```
procedimento Construcao(g(.), α, s);
1   s ← ∅;
2   Inicialize o conjunto C de candidatos;
3   enquanto (C ≠ ∅) faça
4       g(t_min) = min{g(t) | t ∈ C};
5       g(t_max) = max{g(t) | t ∈ C};
6       LCR = {t ∈ C | g(t) ≤ g(t_min) + α(g(t_max) − g(t_min))};
7       Selecione, aleatoriamente, um elemento t ∈ LCR;
8       s ← s ∪ {t};
9       Atualize o conjunto C de candidatos;
10  fim-enquanto;
11  Retorne s;
fim Construcao;
```

Fase de Construção do GRASP p/ minimização

- A seleção do próximo elemento a ser incorporado na solução é determinada avaliando todos os candidatos de acordo com uma função gulosa $g(s)$.
- Esta função $g(s)$ normalmente representa a taxa de crescimento da função de custo (FO) com a inclusão do elemento.
- A avaliação dos elementos a serem incluídos gera uma lista restrita de candidatos (LRC) formada pelos melhores elementos, i.e. aqueles cuja inclusão na solução parcial resulta nos menores aumentos na função de custo do problema.
- O elemento a ser incorporado à solução parcial é selecionado randomicamente da lista LRC. Este é o aspecto probabilístico da metaheurística.
- Se $\alpha = 1$ temos uma escolha aleatória. Se $\alpha = 0$, a escolha será gulosa.

METAHEURÍSTICAS COM UMA ÚNICA SOLUÇÃO

MULT-START

GRASP

} tem sua ênfase na construção da solução inicial para escapar de ótimos locais

MULT-START => solução inicial totalmente randômica

GRASP => solução inicial pseudo-gulosa, ou seja, escolhe aleatoriamente considerando um sub-grupo das melhores opções

Variable Neighborhood Search - VNS

Idéia: Troca de estruturas de vizinhança “onde” são realizadas as buscas locais.

1. Gera uma solução QUALQUER na vizinhança atual da solução corrente

2. Faz uma busca local a partir desta solução

3. Se a busca local melhorar a solução

4. Então retorna à primeira estrutura de vizinhança

5. Senão vai para a próxima estrutura de vizinhança

   (mais “distante”)

6. Retorna ao passo 1.

Variable Neighborhood Search

```
procedimento VNS()
1  Seja s_0 uma solução inicial;
2  Seja r o número de estruturas diferentes de vizinhança;
3  s ← s_0;                {Solução corrente}
4  enquanto (Critério de parada não for satisfeito) faça
5      k ← 1;              {Tipo de estrutura de vizinhança corrente}
6      enquanto (k ≤ r) faça
7          Gere um vizinho qualquer s' ∈ N^(k)(s);
8          s'' ← BuscaLocal(s');
9          se (f(s'') < f(s))
10             então
11                 s ← s'';
12                 k ← 1;
13             senão
14                 k ← k + 1;
15         fim-se;
16     fim-enquanto;
17 fim-enquanto;
18 Retorne s;
fim VNS;
```

Variable Neighborhood Descent

VND - Método de busca local originalmente utilizado no VNS

```
procedimento VND(f(.),N(.),r,s)
1  Seja r o número de estruturas diferentes de vizinhança;
2  k ← 1;                {Tipo de estrutura de vizinhança corrente}
3  enquanto (k ≤ r) faça
4      Encontre o melhor vizinho s' ∈ N^(k)(s);
5      se (f(s') < f(s))
6          então
7              s ← s';
8              k ← 1;
9          senão
10             k ← k + 1;
11     fim-se;
12 fim-enquanto;
13 Retorne s;
fim VND;
```

Variable Neighborhood Search - VNS

- O VNS utiliza diferentes estruturas de vizinhança (diferentes vizinhanças) para escapar de soluções ótimas locais.
- As vizinhanças devem estar aninhadas das mais “próximas” para as mais “distantes” da solução corrente. Pode ser diferente!
- Parte do princípio de que uma solução ótima global é ótima local para qualquer estrutura de vizinhança.

VARIAÇÕES DO VNS:

- *Reduced* VNS (RVNS) – não faz a busca local completa
- *Variable Neighborhood Decomposition Search* (VNDS) – decompoe o problema devido sua estrutura
- Skewed VNS – modifica o critério de aceitação

Iterated Local Search

A busca local pode ser melhorada perturbando-se uma solução ótima local, gerando novas soluções de partida da busca local.

Componentes da metaheurística:

a) Gera solução inicial

b) Busca local

c) Perturbação

d) Critério de Aceitação

Busca local: qualquer método de descida ou uma outra metaheurística

Perturbação: deve ser forte suficiente para escapar do ótimo local, mas fraca o suficiente para manter "boas características" do ótima local corrente!!!

Aceitação: decide de qual solução continuar a exploração e a perturbação a ser aplicada

Aceitação + Perturbação $\leftrightarrow$ intesificação e diversificação

Iterated Local Search

```
procedimento ILS
1   s_0 ← GeraSolucaoInicial();
2   s ← BuscaLocal(s_0);
3   enquanto (os critérios de parada não estiverem satisfeitos) faça
4       s' ← Perturbacao(histórico, s);
5       s'' ← BuscaLocal(s');
6       s ← CriterioAceitacao(s, s'', histórico);
8   fim-enquanto;
fim ILS;
```

Perturbação: procedimentos do tipo destrói – reconstrói (ruin-recreate)

Diversificação x Intensificação

- No desenvolvimento de uma metaheuristica, dois critérios contraditórios devem ser levados em conta:
- a pesquisa do espaço de soluções (diversificação) e a exploração das melhores soluções encontradas (intensificação)
- Na intensificação, as regiões promissoras são exploradas de forma minuciosa, na esperança de encontrar soluções melhores.
- Na diversificação, regiões não exploradas devem ser visitadas para assegurar que “todas” as regiões do espaço de busca estão sendo exploradas equilibradamente e que a pesquisa não está limitada a um pequeno número de regiões.

Busca Tabu

Usa memória para continuar explorando o espaço mesmo sem melhora na solução e evitando ciclos.

Inicia por uma solução $s^* = s^0$ e explora um **subconjunto** V de N(s*). A melhor solução vizinha s' torna-se a nova solução corrente mesmo sem melhora.

Para evitar ciclos, existe a lista tabu T dos movimentos proibidos. T contém os últimos |T| movimentos realizados e tem disciplina de fila.

Para não haver restrição muito grande, existe também uma função de aspiração que pode retirar um movimento da lista T.

Assim, uma solução s' em V obtida por um movimento proibido em T, pode ser aceita se $f(s') \leq A(f(s^*))$ para problema de minimização.

Explo. $f(s') \leq f(s^*) + 1$, ou

$f(s') \leq 1{,}05 \times f(s^*)$, ou ...

Busca Tabu

Parâmetros do método

1. o tamanho máximo da lista T
    - A cada iteração, o último movimento é adicionado ao início da lista, e o movimento mais velho é removido do final da lista.
2. função de aspiração A(.)
3. O tamanho do subconjunto V
4. BTmax = número máximo de iteração sem melhora
5. f_{min} valor de referência para aceitar uma solução

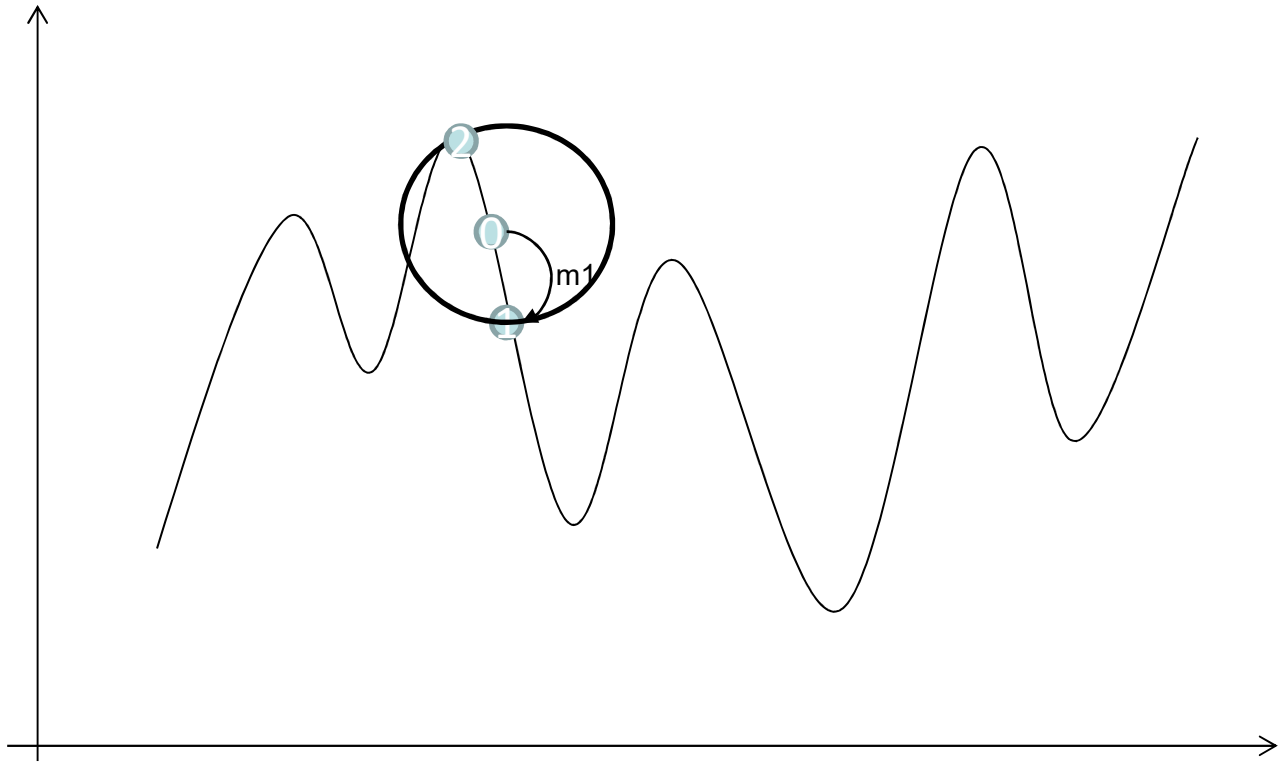
Busca Tabu

procedimento $BT(f(.), N(.), A(.), |V|, f_{min}, |T|, BTmax, s)$

1 $s^{\star} \leftarrow s$; {Melhor solução obtida até então}
2 $Iter \leftarrow 0$; {Contador do número de iterações}
3 $MelhorIter \leftarrow 0$; {Iteração mais recente que forneceu $s^{\star}$}
4 $T \leftarrow \emptyset$; {Lista Tabu}
5 Inicialize a função de aspiração A;
6 enquanto $(f(s) > f_{min}$ e $Iter - MelhorIter \leq BTmax)$ faça
7 $Iter \leftarrow Iter + 1$;
8 Seja $s' \leftarrow s \oplus m$ o melhor elemento de $V \subseteq N(s)$ tal que o movimento m não seja tabu $(m \notin T)$ ou s' atenda a condição de aspiração $(f(s') < A(f(s)))$;
9 Atualize a lista tabu T;
10 $s \leftarrow s'$;
11 se $(f(s) < f(s^{\star}))$ então
12 $s^{\star} \leftarrow s$;
13 $MelhorIter \leftarrow Iter$;
14 fim-se;
15 Atualize a função de aspiração A;
16 fim-enquanto;
17 $s \leftarrow s^{\star}$;
18 Retorne s;

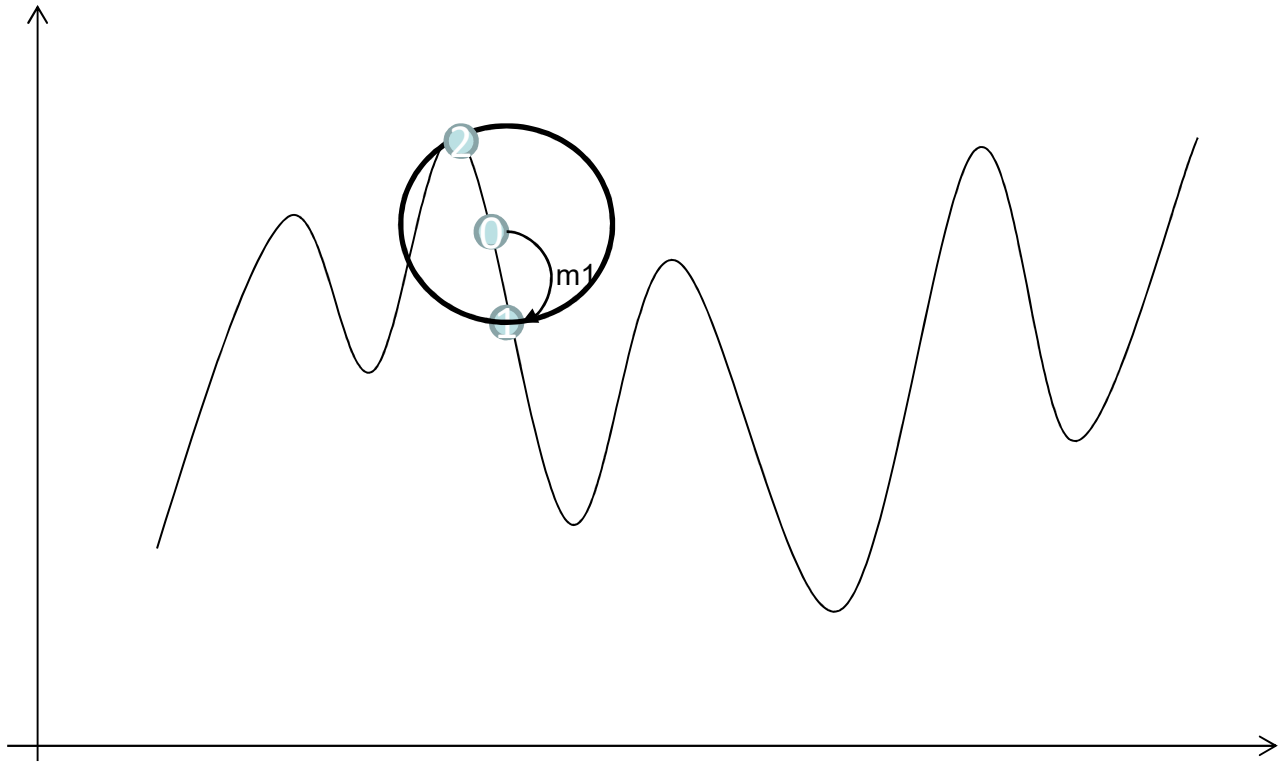
fim BT;

Busca Tabu - 1



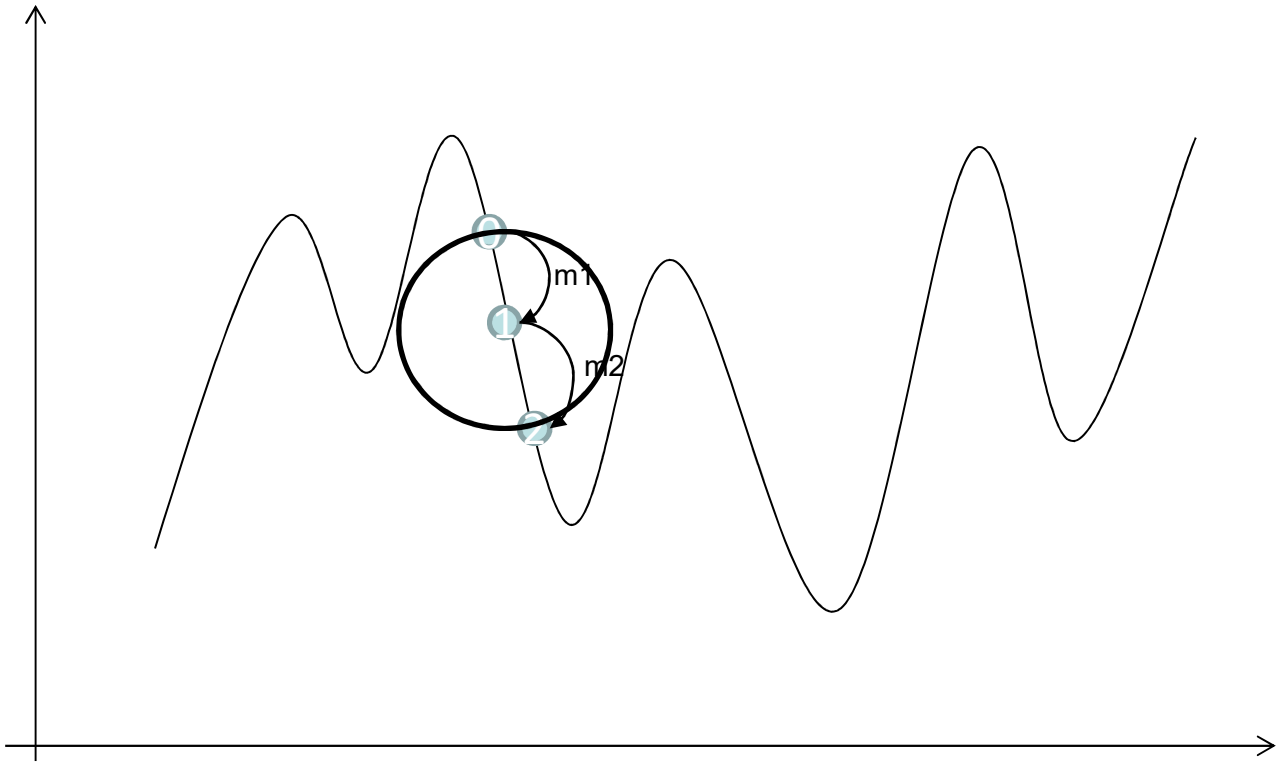
Lista = { }, ou seja, não tem movimento proibido
Tamanho máximo da lista = 4

Busca Tabu - 1



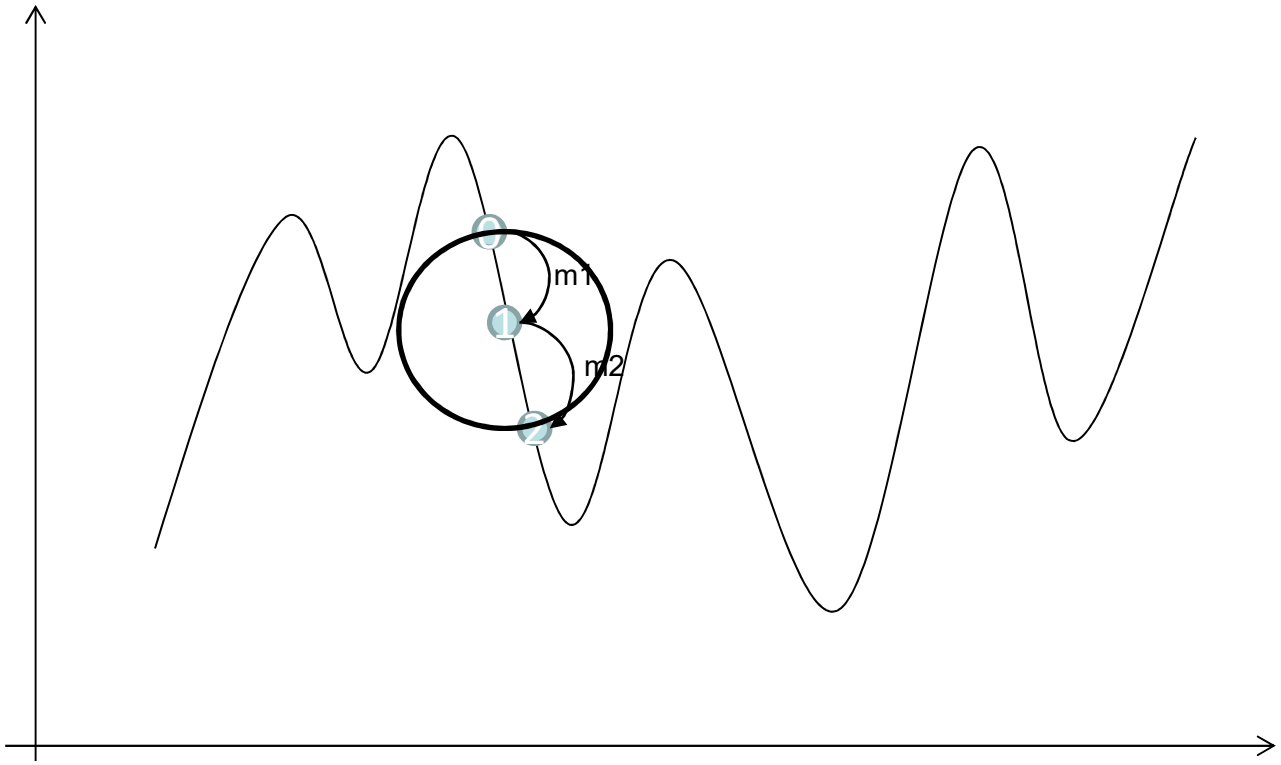
Lista = {m1}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 2



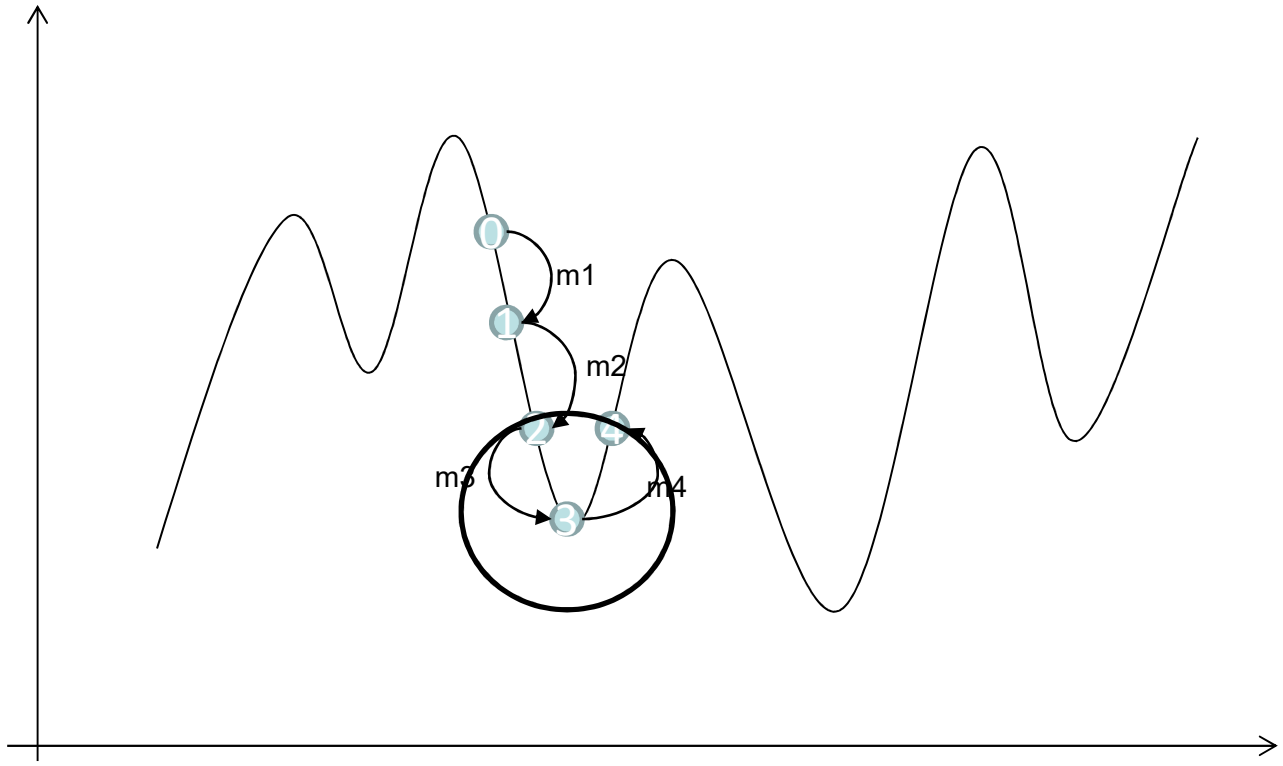
Lista = {m1}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 2



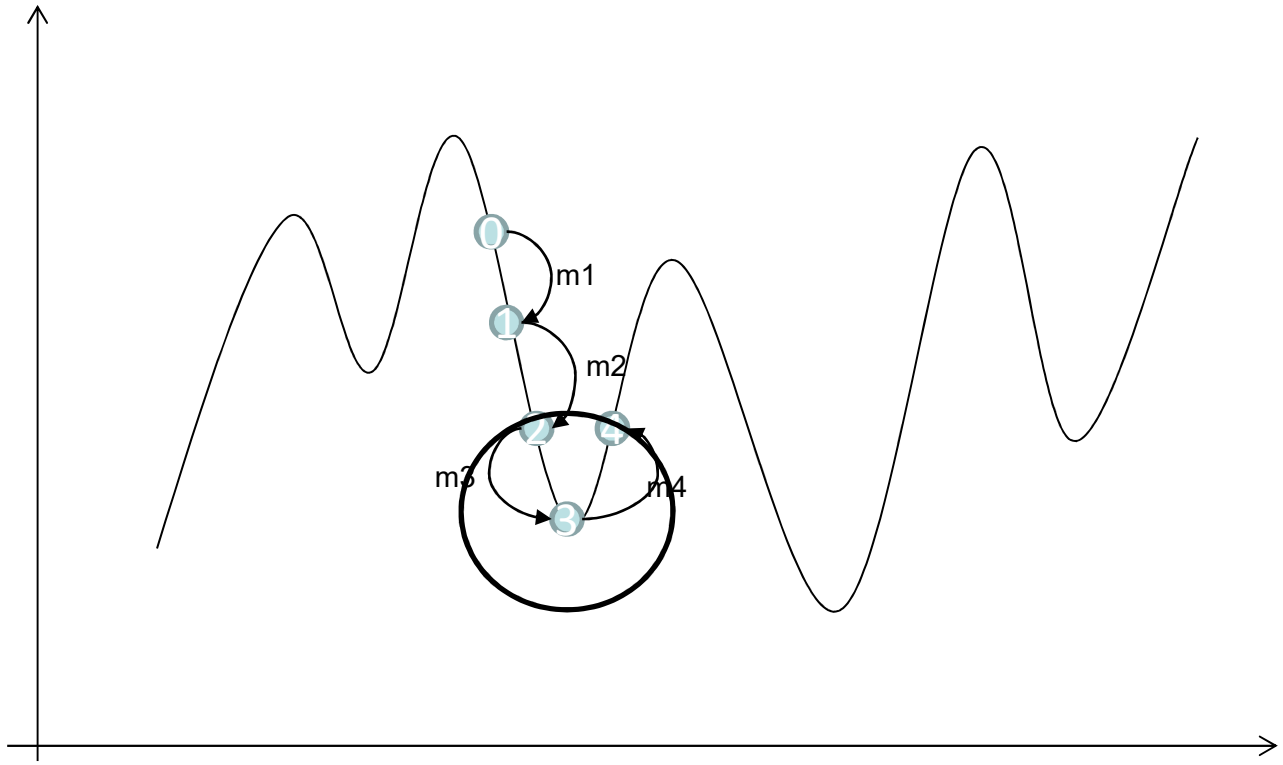
Lista = {m1, m2}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 3



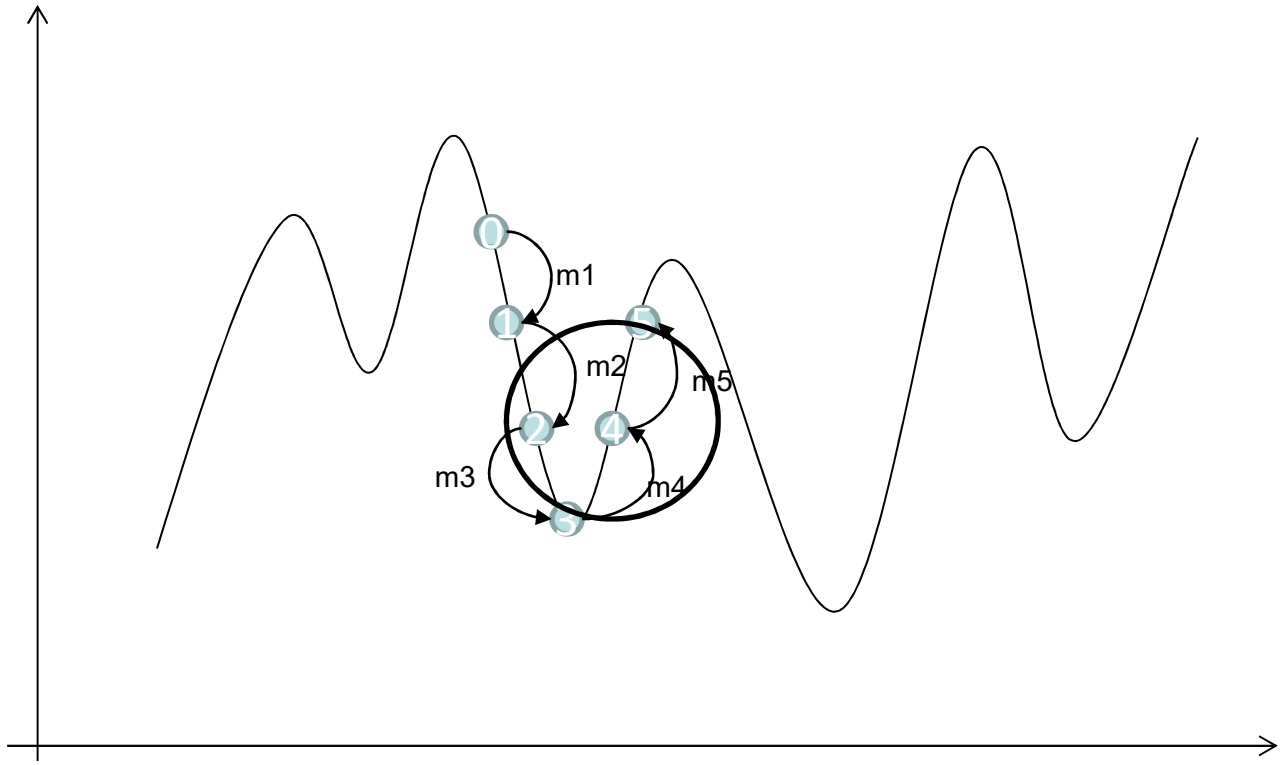
Lista = {m1, m2, m3}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 3



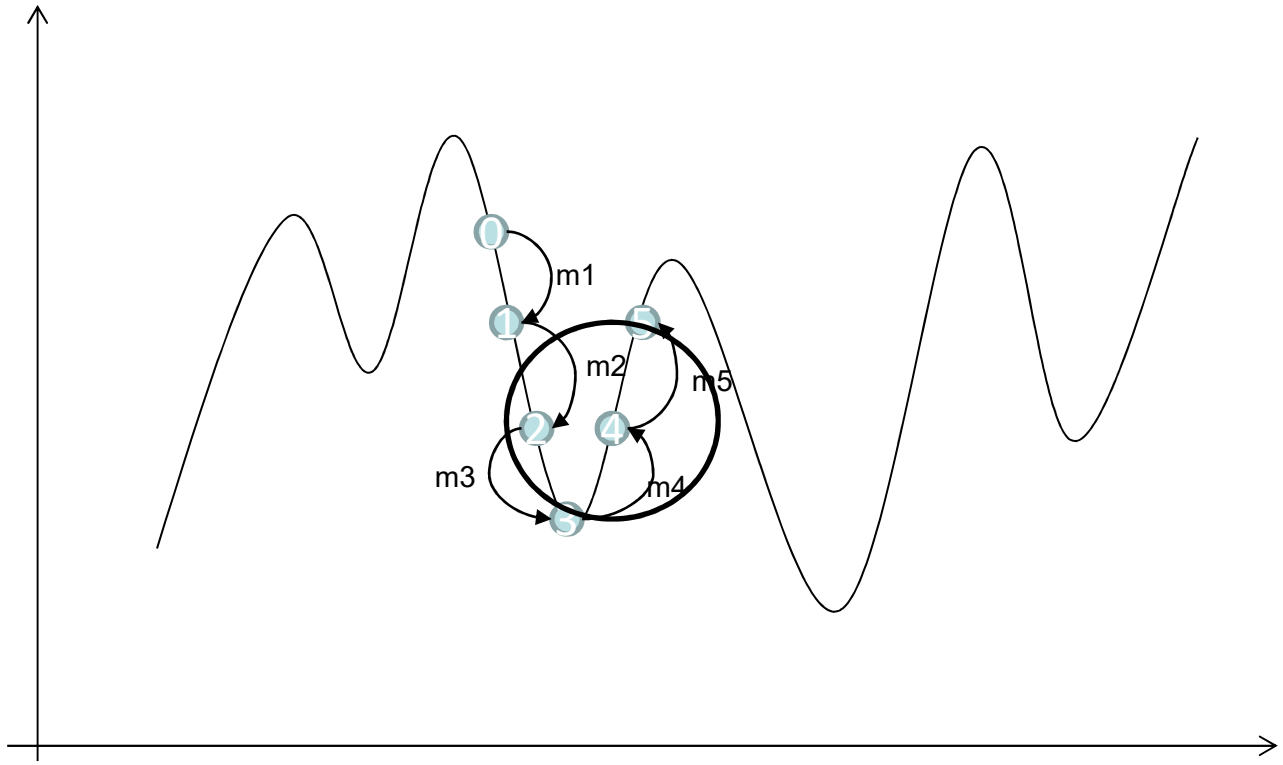
Lista = {m1, m2, m3, m4}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 4



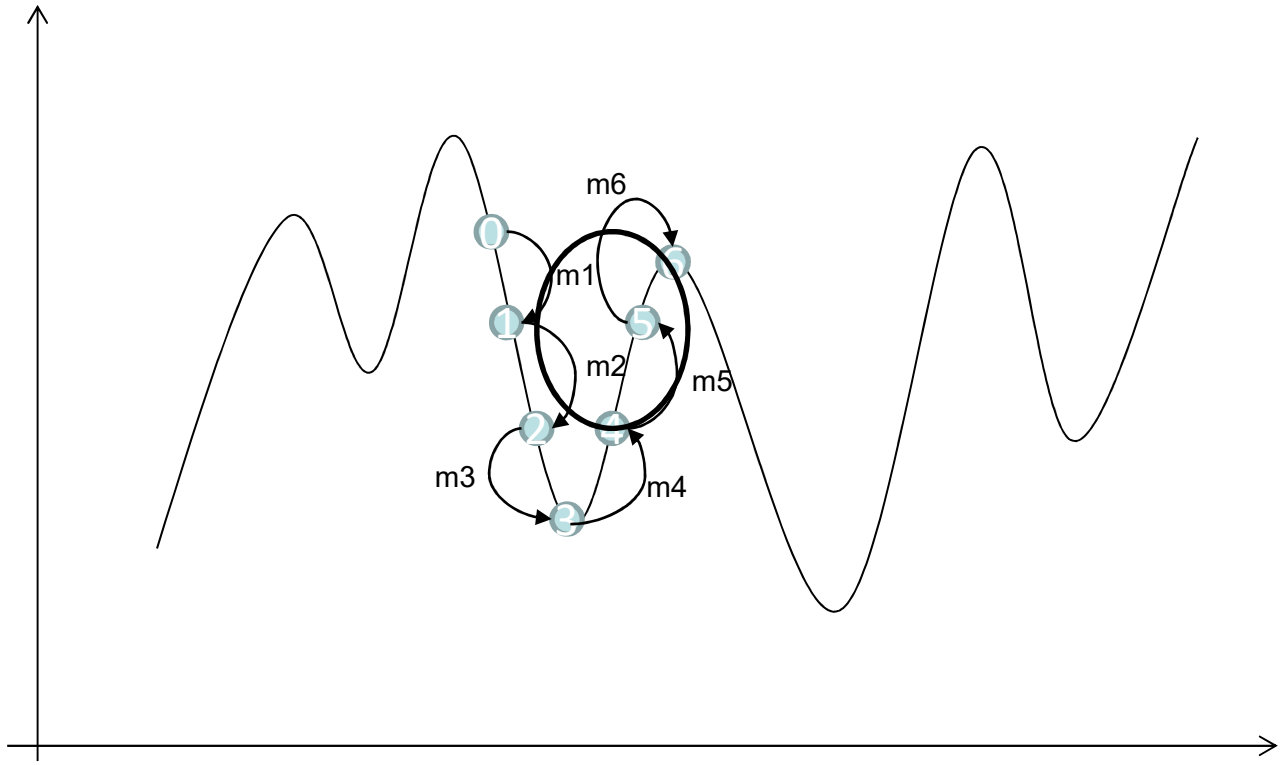
Lista = {m1, m2, m3, m4}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 4



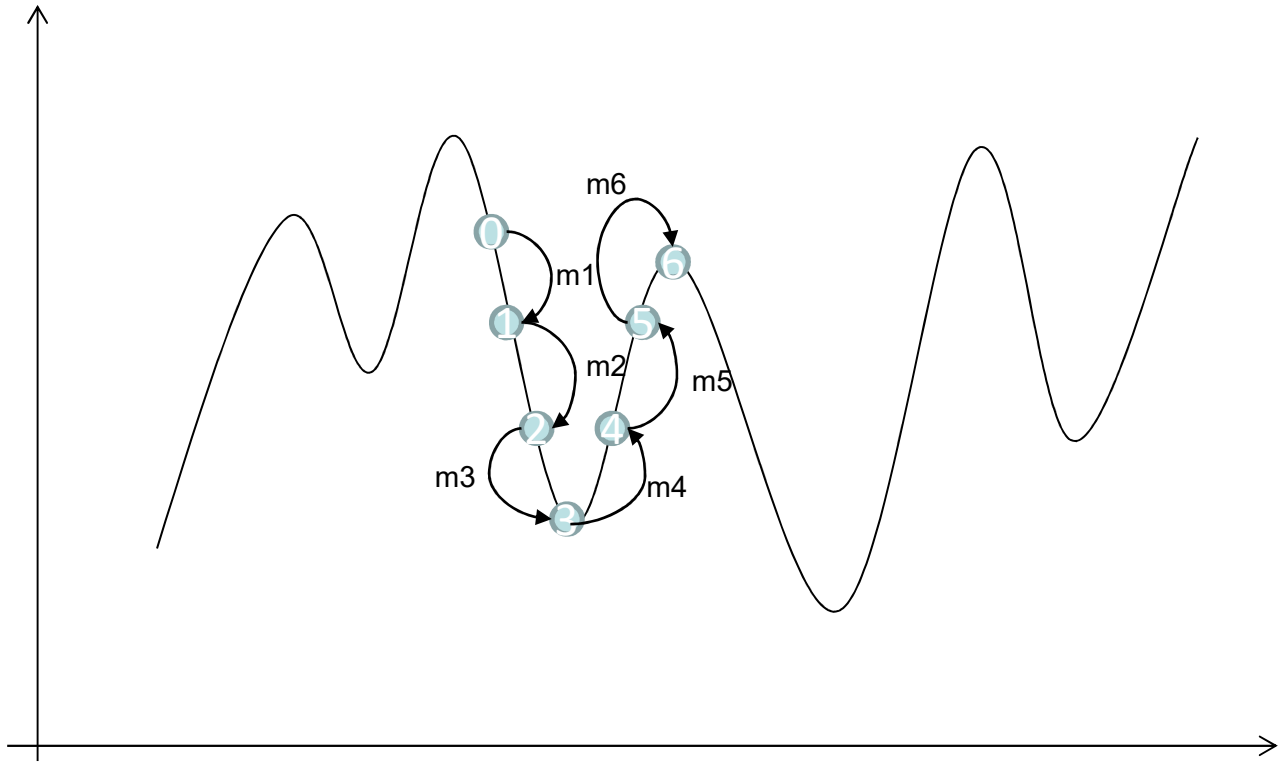
Lista = {m2, m3, m4, m5}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 5



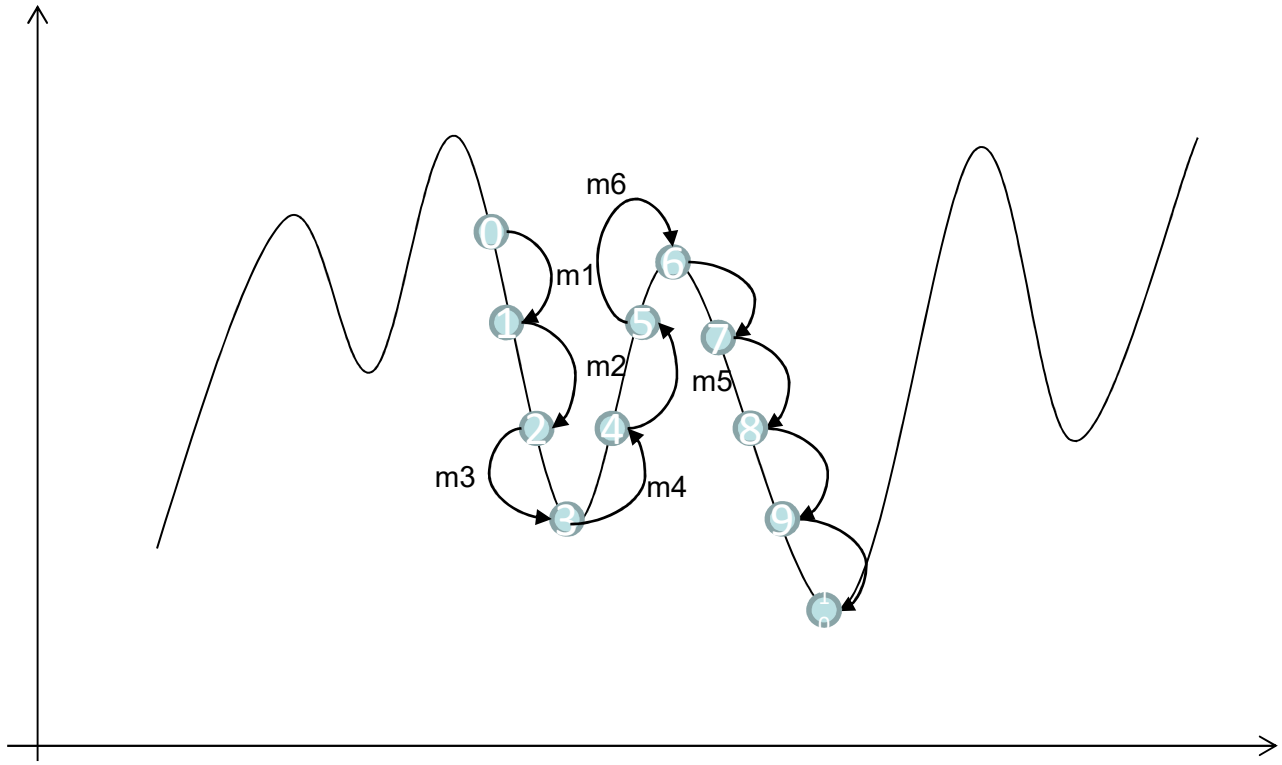
Lista = {m2, m3, m4, m5}, ou seja, não deve desfazer estes movimentos

Busca Tabu - 5



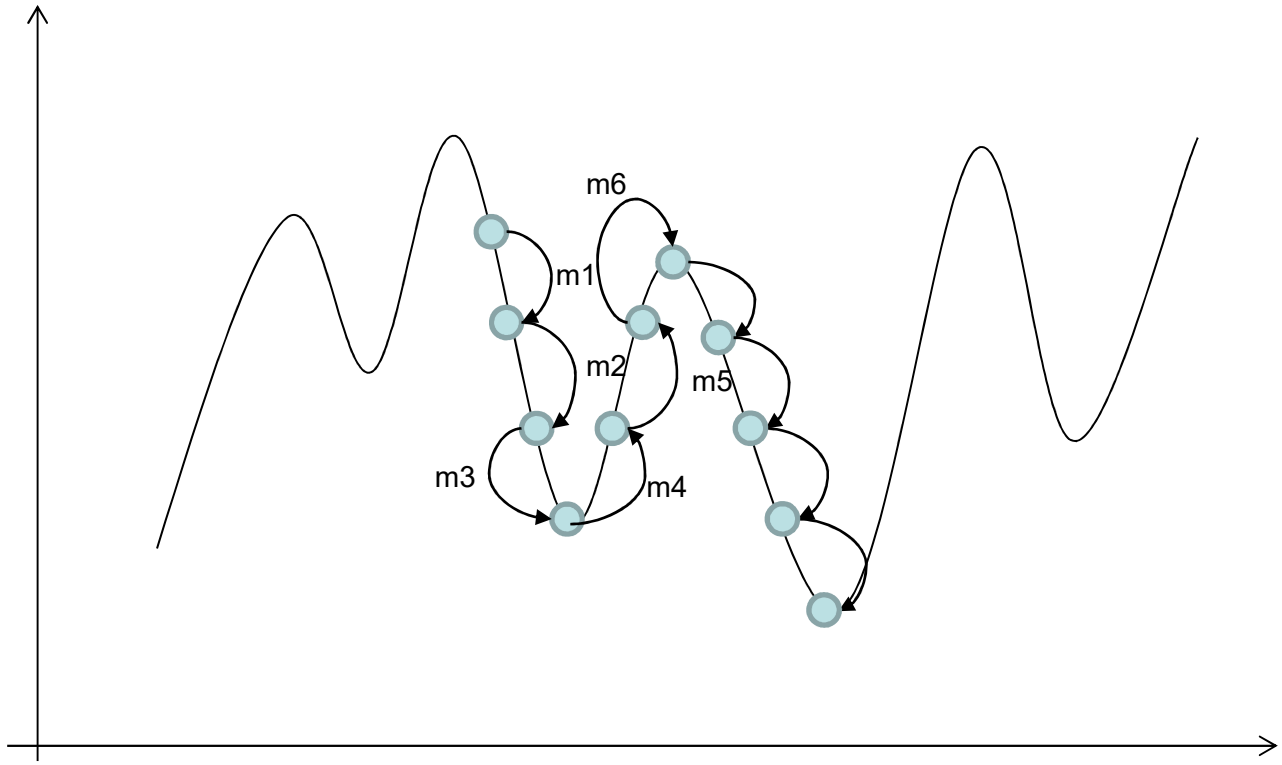
Lista = {m3, m4, m5, m6}, ou seja, não deve desfazer estes movimentos

Busca Tabu – Último



Lista = {m3, m4, m5, m6}, ou seja, não deve desfazer estes movimentos

Busca Tabu – Último



Lista = {m3, m4, m5, m6, ... }, ou seja, não deve desfazer estes movimentos

Busca Tabu

- Usa o mecanismo de memória para de guardar uma série dos últimos movimentos realizados os quais são proibidos de serem “desfeitos”.
- Desta forma a BT explora o espaço de soluções “subindo a colina” e explorando outras regiões e outras soluções ótimas locais.
- Admite movimento ainda na lista (proibidos) se este levar a uma solução que melhore a melhor solução.
- O tamanho da lista de movimentos proibidos evita a geração de ciclos e deve ser definida pelo usuário/desenvolvedor!
- Uma lista pequena leva à intensificação e uma lista grande leva à diversificação da metaheurística.

Simulated Annealing

No laço principal, é gerada *aleatoriamente,* uma solução vizinha s' da solução corrente s.

Seja Δ a variação de valor da FO ao mover-se para s', ou seja,

$\Delta = f(s') - f(s)$ (estamos minimizando f(s))

Se $\Delta < 0$ então houve uma melhora e o método aceita o movimento e a solução vizinha passa a ser a nova solução corrente.

Senão, se $\Delta > 0$, a solução vizinha candidata também poderá ser aceita,

mas neste caso, com uma probabilidade $e^{-\Delta/T}$, onde T é um parâmetro do método, chamado temperatura e que regula a probabilidade de se aceitar soluções de pior custo.

A temperatura inicial, T_0 recebe um valor elevado. Após um número fixo de iterações a temperatura é gradativamente diminuída

$$T = \alpha T, \text{ sendo que } 0 < \alpha < 1$$

Com esse procedimento, no início temos uma chance maior para escapar de mínimos locais. À medida que T aproxima-se de zero, o algoritmo torna-se um método de descida, uma vez que diminui a probabilidade de se aceitar movimentos de piora, pois

se $T \Rightarrow 0$, então $e^{-\Delta/T} \Rightarrow 0$

Simulated Annealing

O procedimento para quando a temperatura chega a um valor próximo de zero e nenhuma solução pior do que a solução corrente é aceita, isto é, quando o sistema está estável.

A solução obtida quando o sistema encontra-se nesta situação é um ótimo local.

Os parâmetros são:
1. a razão de resfriamento α,
2. o número de iterações para cada temperatura (*SAmax*) *e*
3. a temperatura inicial T_0.

A SA tem como principal estratégia para escapar de ótimos locais aceitar soluções de piora segundo uma dada probabilidade e distância da solução até a melhor solução.

Para explorar um vale, existe o mecanismo de resfriamento que diminui a probabilidade da SA aceitar soluções de piora.

Simulated Annealing

```
procedimento SA(f(.), N(.), α, SAmax, T0, s)
1  s* ← s;                    {Melhor solução obtida até então}
2  IterT ← 0;                 {Número de iterações na temperatura T}
3  T ← T0;                    {Temperatura corrente}
4  enquanto (T > 0) faça
5      enquanto (IterT < SAmax) faça
6          IterT ← IterT + 1;
7          Gere um vizinho qualquer s' ∈ N(s);
8          Δ = f(s') − f(s);
9          se (Δ < 0)
10             então
11                 s ← s';
12                 se (f(s') < f(s*)) então s* ← s';
13             senão
14                 Tome x ∈ [0, 1];
15                 se (x < e^(−Δ/T)) então s ← s';
16         fim-se;
17     fim-enquanto;
18     T ← α × T;
19     IterT ← 0;
20 fim-enquanto;
21 s ← s*;
22 Retorne s;
fim SA;
```

Metaheurísticas Populacionais

- Inicia com um conjunto de soluções distintas (população inicial). Cada indivíduo representa uma solução.
- Iterativamente gera uma nova população e troca a população corrente por uma nova população de soluções.
    - Na fase de geração, uma nova população de soluções é gerada a partir da população anterior.
    - Na fase de troca, a seleção acontece considerando a população corrente e a nova população de soluções. As melhores soluções vão compor a nova população.
- Este processo continua iterativamente até que algum critério de parada seja satisfeito ou a população se estabilize (todas as soluções ficam praticamente iguais).

Algoritmos Genéticos

Cromossomo = uma solução do problema

= um indivíduo da população

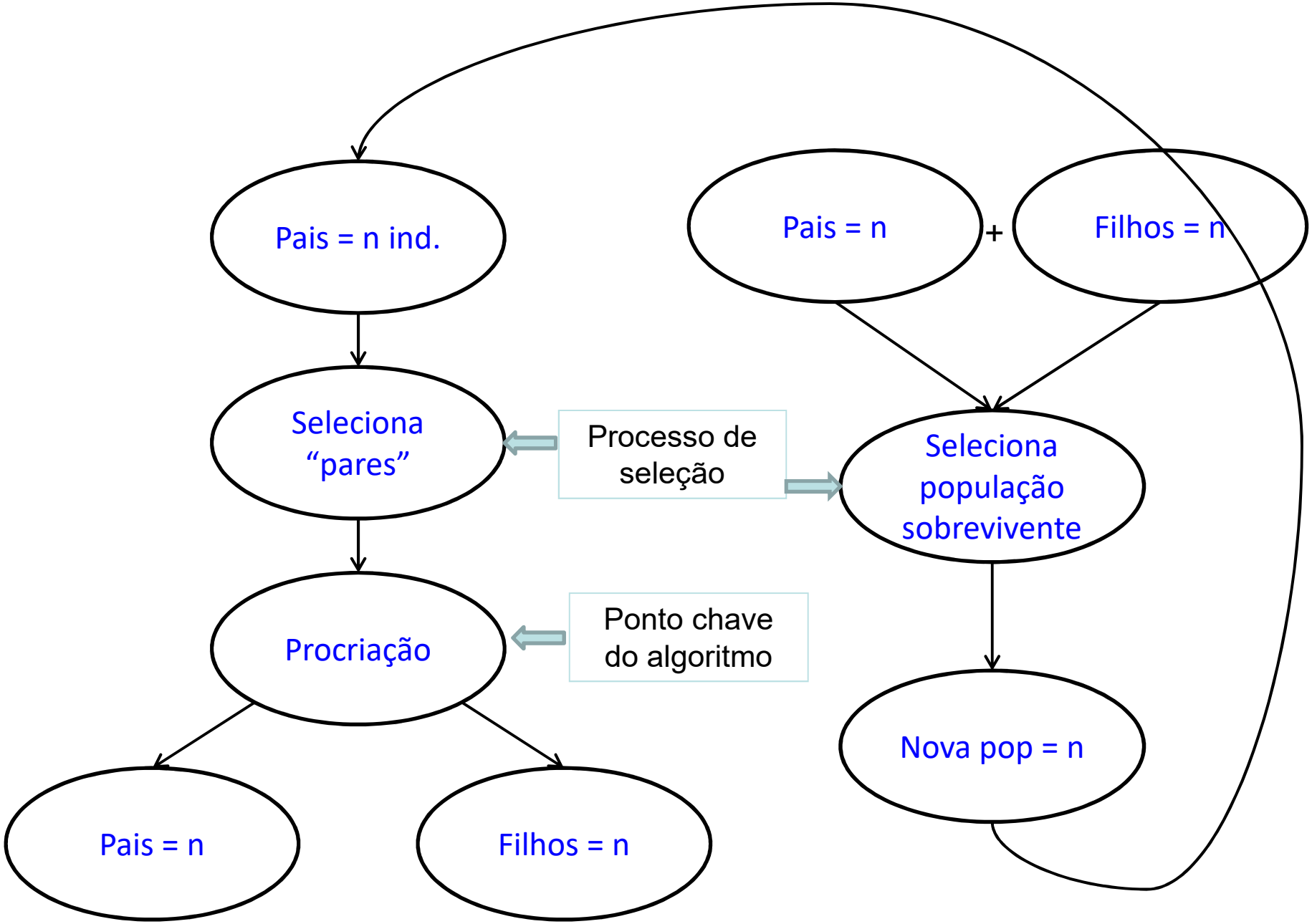
Gene = um componente da solução

O Mecanismo de reprodução é um movimento aplicado na população para explorar o espaço de soluções e obter soluções melhores para o problema.

O algoritmo inicia com uma população de *n* soluções $\{s_1^0, s_2^0, ..., s_n^0\}$ no tempo 0.

Cria a população $\{s_1^t, s_2^t, ..., s_n^t\}$ no tempo *t* a partir da população do tempo $t - 1$.

Os indivíduos passam de $t - 1$ para *t* devido à fase de reprodução proveniente de operações de recombinação e/ou mutação genética.

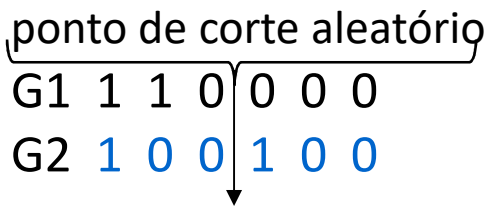

Pais = n ind.
Pais = n
+
Filhos = n
Seleciona "pares"
Processo de seleção
Seleciona população sobrevivente
Procriação
Ponto chave do algoritmo
Nova pop = n
Pais = n
Filhos = n

Algoritmos Genéticos

Operadores Genéticos – Recombinação/Crossover

Indivíduos selecionados (e reproduzidos na população seguinte) são recombinados através do operador de ***crossover*** (com uma probabilidade p_c). O *crossover* (cruzamento) é uma característica fundamental nos AGs.

Considere as soluções G1 e G2 escolhidos com *uma certa aleatoriedade*



Os cromossomos criados a partir do operador de crossover são submetidos à operação de **mutação** (com uma probabilidade p_m).

Algoritmos Genéticos

Operadores Genéticos - Mutação

Mutação é um operador exploratório que tem por objetivo aumentar a diversidade na população.

A mutação troca o conteúdo de uma posição do cromossomo, com uma determinada probabilidade, em geral baixa (< 1%).

C1 1 1 1 1 0 [0] antes
C1 1 1 1 1 0 1 depois da mutação

Crossover

Clássico

$(1\ 0\ 1 \mid 0\ 1\ 0) = p_1$

$(1\ 1\ 1 \mid 0\ 0\ 0) = p_2$

↑

ponto de corte

⇩

$O_1 = (1\ 0\ 1 \mid 0\ 0\ 0)$

$O_2 = (1\ 1\ 1 \mid 0\ 1\ 0)$

PMX

$p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$

$p_2 = (4\ 2\ 6 \mid 1\ 8\ 5\ 9 \mid 3\ 7)$

pontos de corte

⇩

$O_1 = (4\ 2\ 3 \mid 1\ 8\ 5\ 9 \mid 6\ 7)$

$O_2 = (1\ 2\ 8 \mid 4\ 5\ 6\ 7 \mid 3\ 9)$

Algoritmos Genéticos

Gerada uma população do tempo *t*, define-se os sobreviventes, ou seja,

as *n* soluções da nova população.

Podem ser escolhidas as *n* melhores soluções dentre as 2*n* soluções:

n da geração *t* - 1 e *n* da geração *t*.

A sobrevivência depende da função objetivo e os critérios mais usados são:

1. Escolha aleatória dos indivíduos
2. Proporcional ao valor da função objetivo
3. Combinação dos critérios anteriores

Algoritmos Genéticos

Parâmetros e Critérios de Parada

1. Tamanho da População → número de pontos do espaço de busca.
2. Taxa de *Crossover* → probabilidade p_c de um indivíduo ser recombinado com outro.
3. Taxa de Mutação → probabilidade p_m do conteúdo de uma posição do cromossomo ser alterado.
4. Número de Gerações → total de ciclos de evolução de um AG.
5. Total de Indivíduos → total de tentativas em um experimento
   = (tamanho da população x número de gerações)

Algoritmos Genéticos