

Heurísticas Para Otimização Combinatória

Prof. Ademir A. Constantino

Departamento de Informática
Universidade Estadual de Maringá

www.din.uem.br/~ademir

Resumo

- As anotações desta apresentação reune informações coletadas em várias fontes, principalmente artigos científicos. Recomenda-se a leitura dos textos indicados para maiores informações.
- **Conteúdo:**
 - Otimização combinatória
 - Heurística
 - Algoritmo heurístico
 - Algoritmo Construtivo
 - Algoritmo Melhorativo, também conhecido como Algoritmo de Busca Local (ou na Vizinhança)
 - Meta-Heurística

O que é Otimização Combinatória (definição informal)

- Problemas de otimização combinatória
 - * Dentre um conjunto finito (grande) de soluções, escolher a melhor.
 - * Esses problemas são modelados como problemas de maximizar (ou minimizar) uma função cujas variáveis devem obedecer certas restrições.

Exemplos Clássicos

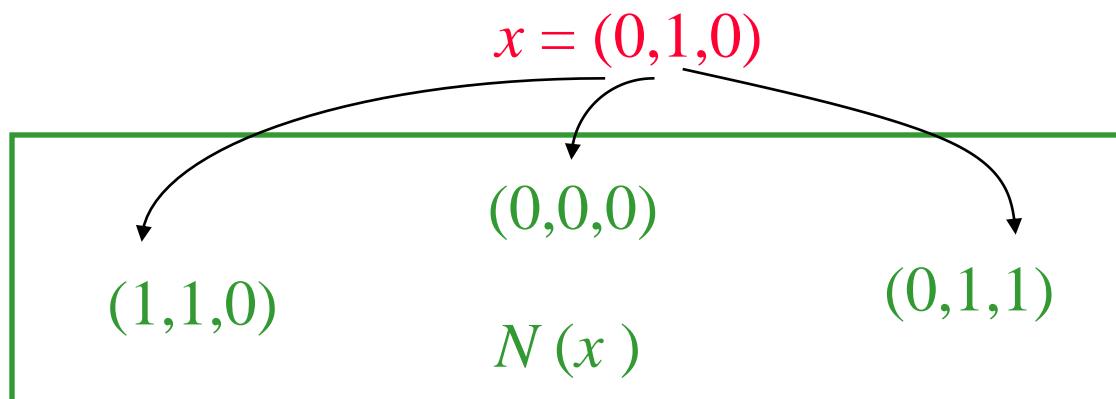
- Problema da Mochila
 - Dados n objetos que posso armazenar em uma mochila, onde cada um tem um peso e uma utilidade, quanto de cada objeto devo escolher de tal modo que o peso total não seja maior que P e o somatório das utilidades seja o maior possível?
- Problema do Caixeiro Viajante
 - Dadas n cidades, encontrar um caminho passando por todas elas de maneira que o percurso total seja o menor possível.

Aplicações de Otimização Combinatória

- projetos de sistemas de distribuição de energia elétrica,
- posicionamento de satélites,
- projetos de computadores e de chips VLSI,
- roteamento ou escalonamento de veículos,
- alocação de trabalhadores ou máquinas à tarefas,
- empacotamento de caixas em containers,
- corte de barras e placas,
- seqüenciamento de genes e DNA,
- classificação de plantas e animais,
- etc.

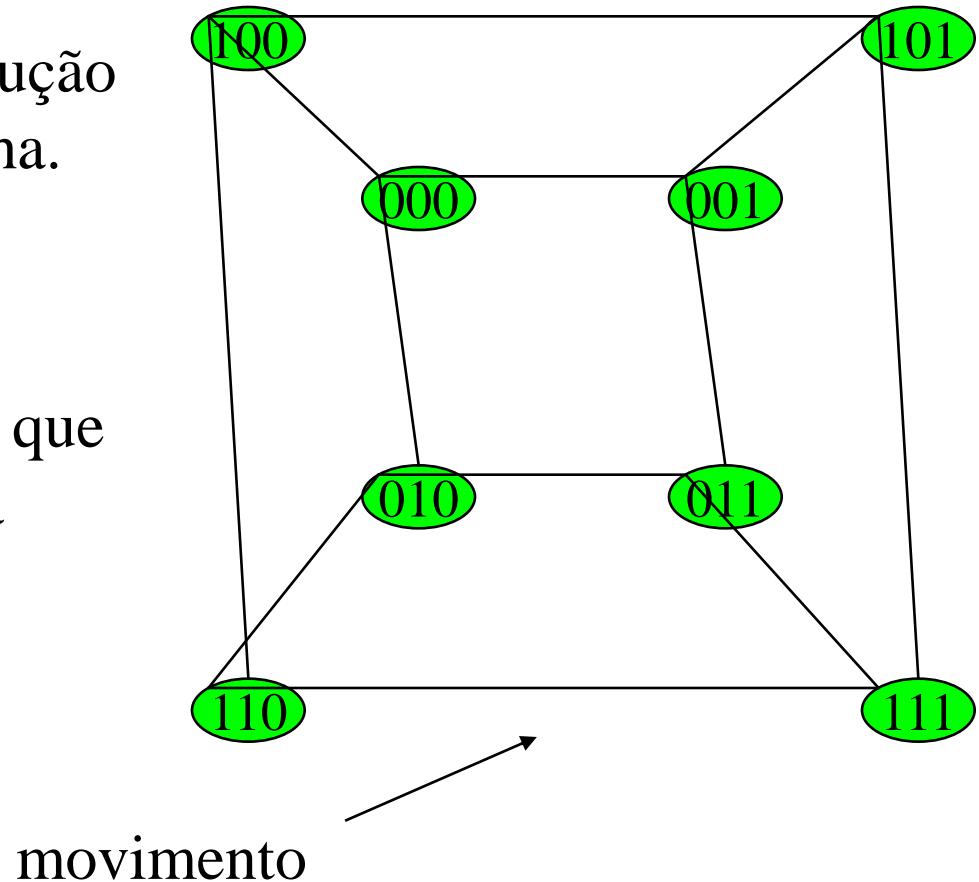
Conceitos básicos

- Espaço de soluções
 - * O conjunto de todas as soluções (viáveis) possíveis (satisfazendo as restrições do problema)
- Vizinhança
 - * Dada uma solução x , os elementos da vizinhança $N(x)$ de x são aquelas soluções y que pode ser obtida aplicando uma perturbação elementar sobre x .
 - * Exemplo: Considere $x = (0,1,0)$ e a vizinhança 1-flip de um vetor 0/1.



Espaço de busca

- Movimento
 - * É a transição de uma solução para outra solução vizinha.
- Espaço de busca
 - * O conjunto das soluções que obtidas por meio de uma vizinhança.

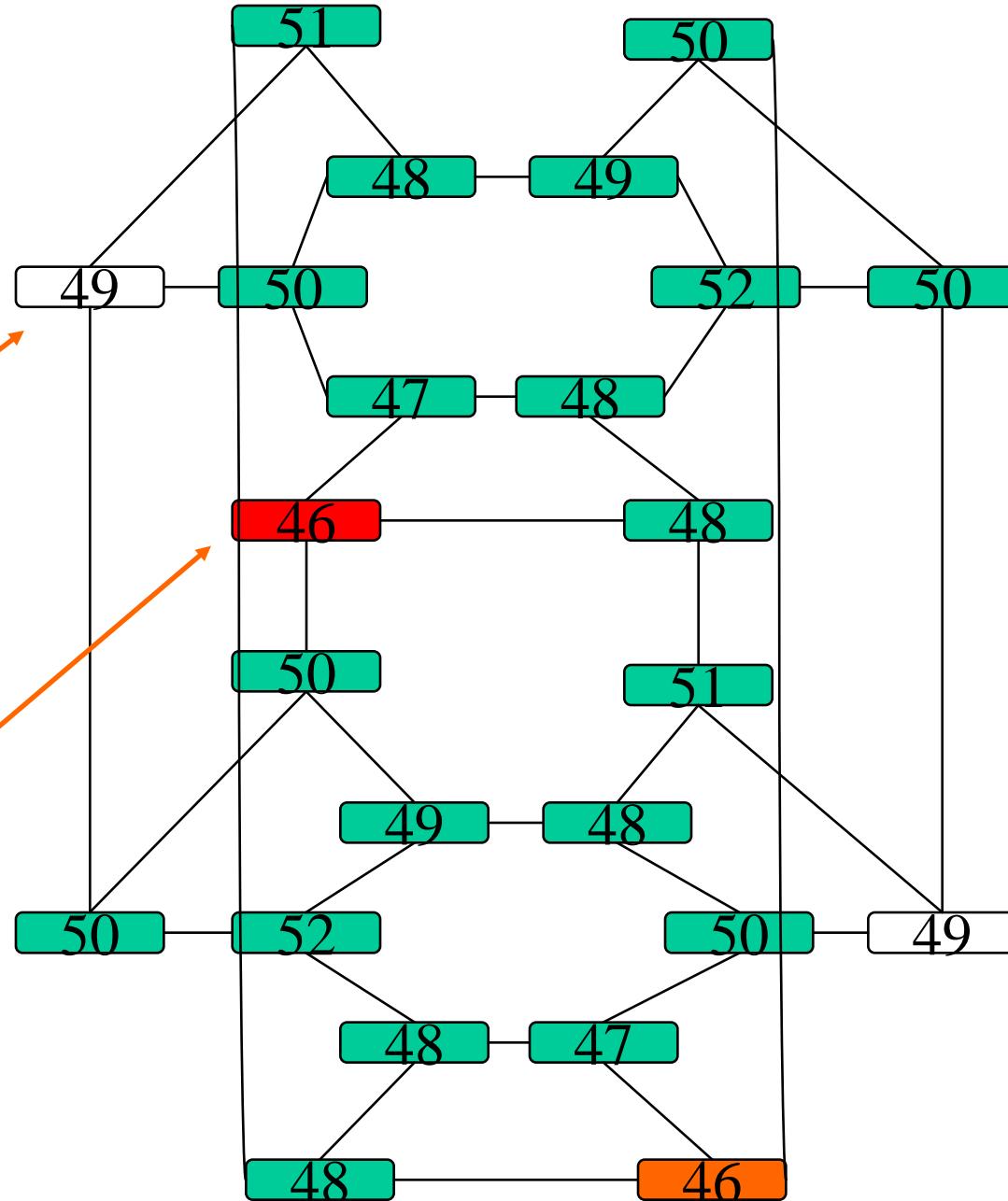


Conceitos

- Solução Incumbente;
 - * é a solução corrente mantida pelo algoritmo
- Ótimo local
 - * é uma solução tão boa ou melhor do que qualquer das soluções vizinhas
- Ótimo global (solução ótima).
 - * É a melhor solução dentre todos os ótimos locais.

Ótimo local

Ótimo global



Complexidade Computacional

- Problemas da classe:
 - NP-difícil ou NP-árduo (NP-hard)
 - NP-Completo.
- Todo NP-Completo é NP-Hard
- Nem todo NP-Hard é NP-Completo
- NP-Hard é tão difícil quanto NP-Completo

Desafios Computacionais

- Resolver problemas de otimização combinatória pertencentes a classe NP-hard ou NP-completo com pouco esforço computacional (tempo de execução).
- Encontrar soluções ótimas, ou mesmo aproximadas, para esses tipos de problemas é um desafio nem sempre fácil de ser vencido

Resolução de Problemas NP-difícil e NP-completo

- Algoritmos exatos não-polinomiais:
 - * programação dinâmica , *branch-and-bound*, *branch-and-cut* , planos-de-corte, *backtracking*, etc.
- Algoritmos aproximativos: é a denominação do algoritmo que fornece soluções dentro de um *limite de qualidade absoluto* ou assintótico, assim como um limite assintótico polinomial de complexidade (pior caso) comprovado matematicamente;

- **Algoritmos probabilísticos:** são algoritmos que convergem para a solução do problema dentro de um valor esperado.
- **Algoritmos heurísticos:** são denominações para o algoritmo que fornece soluções sem um limite formal de qualidade, tipicamente avaliado empiricamente em termos de complexidade (média) e qualidade das soluções.

Heurística

- **Etimologia:**
 - * a palavra **heurística** vem da palavra grega *Heuriskein*, que significa descobrir (e que deu origem também ao termo Eureca)
- **Heurística:** pode ser entendida como informação e intuição a respeito da instância do problema e da sua estrutura para resolvê-lo de forma rápida.
- Uma **heurística** é um conjunto de regras e métodos que conduzem à descoberta, à invenção e à resolução de problemas.

Algoritmo Heurístico

- Um algoritmo heurístico é um algoritmo que utiliza alguma heurística na sua concepção.
- Em geral, os algoritmos heurísticos não são capazes de garantir a solução ótima do problema de otimização.
- O objetivo de uma heurística é tentar encontrar uma solução “boa” de maneira simples e rápida.
- Existe algoritmo heurístico que garante a solução ótima de um problema? A resposta é **SIM**.
 - * Exemplos: Algoritmo A* com heurística admissível, algoritmos Prim e Kruskal para árvore expandida de custo mínimo.

Justificativas

- O que justifica o uso de um algoritmo heurístico sem garantia de solução ótima?
- Só é justificável o uso de algoritmo heurístico para um problema quando
 - * não é possível modelar o problema
 - (usando grafos, programação matemática, etc.)
 - * ou não é possível aplicar um algoritmo exato
 - (em função do tempo computacional excessivo).

Algoritmos Heurísticos

- Heurística
 - de propósito geral para domínios variados
 - para domínios específicos

Algoritmos Heurísticos (classificação)

1. Método construtivo

- processo iterativo que inicia com uma solução vazia e adiciona um novo elemento a cada iteração até a obtenção de uma solução.

2. Busca local (ou melhorativo)

- inicia em uma solução (podendo ser obtida a partir de outra heurística) e caminha sobre as soluções vizinhas.

3. Meta-heurísticas Busca local

- estrutura genérica de heurística que podem ser adaptadas para diversos problemas.

Algoritmos Heurísticos (classificação)

4. Métodos de decomposição

- consistem em dividir o problema em subproblemas menores, de modo que a resolução de todos os subproblemas possam compor uma solução para o problema maior.

5. Relaxação

- modificar o modelo de tal forma que ele fique mais fácil de resolver.
- Ex. relaxação linear, relaxação lagrangeana.

Método Construtivo

- Problema de otimização combinatória:

Dado um conjunto finito de elementos $E = \{1, 2, \dots, n\}$
e uma função de custo $c: 2^E \rightarrow \mathbb{R}$

encontrar $s^* \in F$ tal que $c(s^*) \leq c(s) \quad \forall s \in F$

onde $F \subseteq 2^E$ é uma coleção de subconjuntos de E (soluções viáveis do problema).

S : conjunto de soluções viáveis (satisfaz as restrições do problema);

s^* : definida como solução ótima do problema.

- Exemplos:
- Problema do caixeiro viajante (PCV)

E : conjunto de arestas

F : subconjuntos de E que formam um ciclo hamiltoniano (CH)

$c(S) = \sum_{e \in S} c_e$, c_e = custo da aresta e .

- Problema da Mochila (Knapsack)

E : conjunto de itens

F : subconjuntos de E que satisfazem a restrição de peso

$$\sum_{e \in S} u_e \leq b, \quad u_e = \text{utilidade de } e$$

Método Construtivo

- Forma geral de um algoritmo heurístico construtivo : selecionar sequencialmente elementos de E , eventualmente descartando alguns já selecionados, de tal forma que ao final se obtenha uma solução viável, i.e. pertencente a F .

Método Construtivo

- Exemplos de algoritmos construtivos:

1. Problema do Caixeiro Viajante

- Vizinho mais próximo
- Inserção do Mais Próxima
- Inserção do Mais Distante
- Método das economias (Clarke e Wright)

2. Árvore Geradora de Custo Mínimo

- Kruskal
- Prim
- Dijkstra

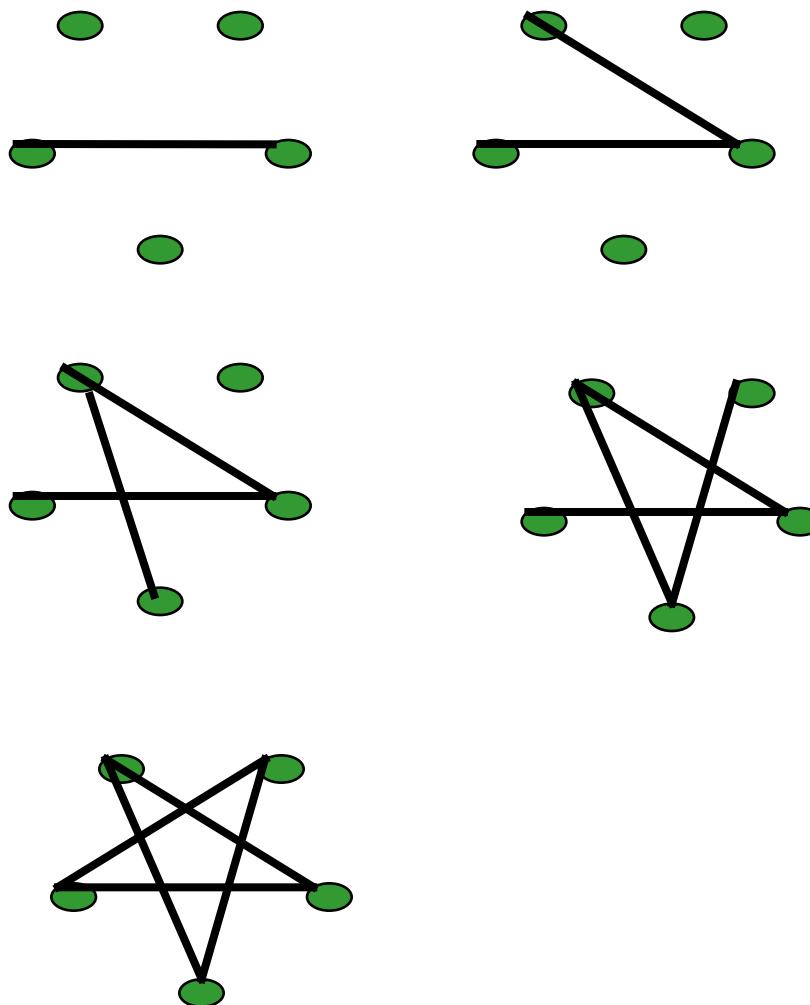
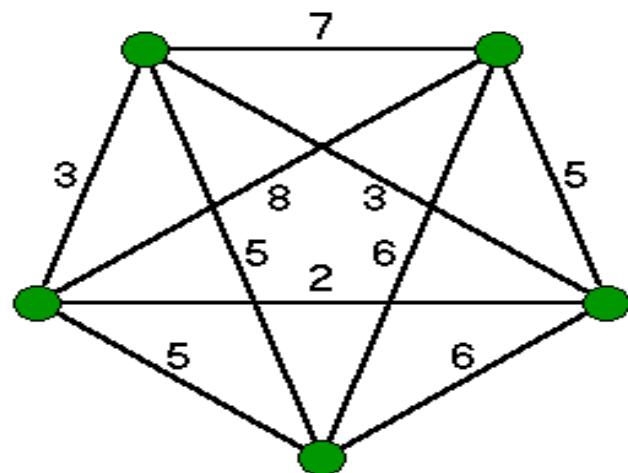
Continuação

3. Bin-packing (empacotamento)

- Botton Left
 - Ex: <http://www.cs.gmu.edu/~jarboe/bottomLeft.html>
- Next Fit
 - Ex: <http://www.cs.gmu.edu/~jarboe/nf.html>
- First Fit
 - Ex: <http://www.cs.gmu.edu/~jarboe/ff.html>

Método Construtivo

Heurística vizinho mais próximo



Algoritmos gulosos (greedy): uma classe de algoritmos construtivos

- Algoritmos gulosos (greedy): a construção de uma solução gulosa consiste em selecionar seqüencialmente um elemento de E que minimiza o incremento no custo da solução “parcial”, eventualmente descartando alguns já selecionados, de tal forma que ao final se obtenha uma solução viável. O custo de cada elemento é calculado por uma função denominada *função gulosa* definida como $g: E \rightarrow R$.
- O critério de otimização de um algoritmo guloso é meramente local, não sendo possível, portanto, garantir a solução ótima global, exceto para estruturas do tipo **MATROOIDES** que veremos mais tarde.
- Cada iteração o algoritmo considera apenas a próxima decisão, levando a ser chamado também de algoritmo **míope**, pois “enxerga” somente o que está mais próximo.

Esquema Geral de um Algoritmo Greedy (minimização)

- Passo 1. Ordene os elementos de E em ordem crescente: $g(e_1) \leq g(e_2) \leq \dots \leq g(e_n)$, sendo g uma função gulosa.
- Passo 2. $S \leftarrow \emptyset$
- Passo 3. Para $i=1$ até n faça:
Se $S \cup \{e_i\} \in F$ então $S = S \cup \{e_i\}$
- Passo 4. A Melhor solução é encontrada:
 $S^* \leftarrow S$ e $c(S^*) = \sum_{e \in S^*} c(e)$

Heurística - Método Construtivo

Exemplo: Problema da Mochila - *Knapsack Problem*

- Dada uma mochila de capacidade limitada de peso e um conjunto itens distintos com peso conhecido e cujo o transporte resulta num determinado lucro. Encontrar uma combinação dos itens a serem transportados maximizando o lucro total.

Heurística - Método Construtivo

- Exemplo: Problema da mochila (*Knapsack Problem*)

E : conjunto de itens

F : subconjuntos de E que satisfazem à

restrição $\sum_{e \in S} a_e \leq b$

custo de solução S : $c(S) = \sum_{e \in S} c_e$

c_e : lucro do item e

a_e : peso do item e

b : capacidade da mochila

Heurística - Método Construtivo

Exemplo de um algoritmo guloso problema da mochila

Passo 0:

Ordenar os elementos de E em ordem decrescente com base no valor da função gulosa para cada elemento. Suponha que: $c_1/a_1 \geq c_2/a_2 \geq \dots \geq c_n/a_n$;

$S \leftarrow \emptyset$;

Passo 1:

Para i de 1 a n faça

Se $S \cup \{e_i\} \in F$

então $S \leftarrow S \cup \{e_i\}$

Obs:

Função gulosa $g(e) = c_e/a_e$ (densidade do elemento, custo/benefício)

Exemplo: Ag. Guloso para o PM

Dados do problema:

$\frac{5}{3}$	$\frac{4}{5}$	$\frac{6}{7}$	$\frac{3}{4}$	$\frac{5}{2}$	$\frac{8}{9}$	$\frac{5}{5}$	$\frac{9}{8}$
1	2	3	4	5	6	7	8

Capacidade da mochila: 15

c_e/a_e Função gulosa
 i Item

Iterações do algoritmo:

Item	Peso	Lucro (acumulado)
5	2	5
5, 1	5	10
5, 1, 8	13	19

Heurística - Método Construtivo

Outras funções gulosas para o problema da mochila

- Escolher os itens por ordem decrescente do valor (c_i).
- Escolher os itens por ordem crescente do valor (a_i).

Heurística - Método Construtivo

- Algoritmo guloso aleatorizado (maximização):

Passo 0:

Ordenar os elementos de E de modo que $c_1 \geq c_2 \geq \dots \geq c_n$;
 $S \leftarrow \emptyset$;

Passo 1:

Para i de 1 a n faça

Criar uma lista $L \subseteq \{1, 2, \dots, n\} \setminus S$ tal que

$$S \cup \{e\} \in F, \forall e \in L$$

Selecionar aleatoriamente um elemento $e \in L$

$$S \leftarrow S \cup \{e\}$$

Heurística - Método Construtivo

- Algoritmo guloso aleatorizado
 - * Algoritmo guloso encontra sempre a mesma solução para um dado problema, exceto por eventuais empates
 - * Aleatorização permite alcançar diversidade nas soluções encontradas
 - * Criar uma lista de candidatos L e forçar uma escolha aleatória a cada iteração
- A diversidade das soluções encontradas depende da cardinalidade da lista L

Heurística - Busca Local

- Algoritmos de busca local são construídos como uma forma de exploração do espaço de busca.
- Partida: solução inicial obtida através de um método construtivo
- Iteração: melhoria sucessiva da solução corrente através de uma busca na sua vizinhança
- Parada: primeiro ótimo local encontrado, ou seja, não existe solução vizinha aprimorante.

Representação de soluções

- Os métodos de busca local são dependentes da forma adotada para representar as soluções, pois a vizinhança de uma solução é obtida a partir da sua representação.
- Representação de uma solução: indicar quais elementos de E estão presentes e quais não estão.
- As formas mais adotadas são:
 - * Vetores de Pertinência
 - * Combinações
 - * Permutações

Representação de soluções

Vetores de pertinência

Vetores onde o mapeamento de cada elemento da solução é feito para cada uma das posições (ou dimensões) do vetor. A presença de um determinado elemento na resposta é representada pelo 1 na posição correspondente. Sua ausência é representada pelo 0 (ou vice-versa)

Combinações

Conjuntos contendo exatamente os elementos presentes na resposta (ou exatamente os elementos ausentes).

Permutações

Conjunto ordinal (contendo todos os elementos) indicando a ordem com que estes aparecem na solução. (ex.: lista de cidades visitadas para o problema do caixeiro viajante).

Representação de soluções

- Vetor de pertinência para o problema da mochila:
n itens, vetor 0-1 com n posições, $x_j = 1$ se o item j é selecionado, $x_j = 0$ caso contrário.

1	2	3	4	5	6	7	8	9	10
1	1	0	0	0	1	0	1	1	0

$$S[i] = \begin{cases} 1 & \text{se o objeto } i \text{ está na resposta} \\ 0 & \text{caso contrário} \end{cases}$$

$$\sum_{i=1}^{10} S[i]W_i \leq C$$

Representação de soluções

- Combinação para o problema da mochila:
Uma solução S é formada pelo conjunto de itens selecionados.

$$S = \{1, 2, 6, 8, 9\}$$

$$\sum_{i \in S} W_i \leq C$$

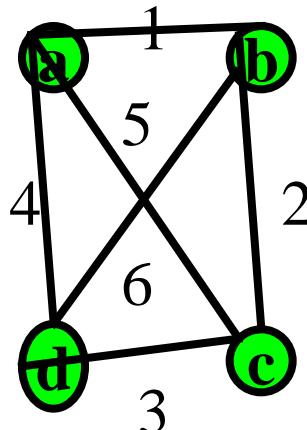
Representação de soluções

Vetor de pertinência para o PCV:

Solução é um vetor de $n = |E|$ posições

$v_e = 1$, se a aresta e pertence ao CH

0, caso contrário.



Soluções viáveis:

$(1,1,1,1,0,0), (1,0,1,0,1,1), (0,1,0,1,1,1)$

Representação de soluções

Permutação para o PCV:

Cada solução é representada pela ordem em que os vértices são visitados, isto é, como uma permutação circular dos n vértices (já que o primeiro vértice é arbitrário)

- | | |
|---------|----------|
| (a) bcd | (b) bdc |
| (c) cbd | (d) cdb |
| (e) dbc | (f) dc b |

Vizinhança

Problema combinatório:

$$f(s^*) = \text{mínimo } \{ f(s) : s \in S\}$$

S é um conjunto discreto de soluções viáveis.

Vizinhança: elemento que introduz a noção de proximidade entre as soluções em S .

Uma vizinhança é um mapeamento

$$N: S \rightarrow 2^S$$

que associa as soluções de S em um subconjunto deste mesmo conjunto de soluções.

Vizinhança

$N(s) = \{s_1, s_2, \dots, s_k\}$ soluções vizinhas de s

Boas vizinhanças permitem representar de forma compacta/eficiente o conjunto de soluções vizinhas a qualquer solução s .

Espaço de busca: definido pelo conjunto de soluções S e por uma vizinhança N

Vizinhança

Exemplo de vizinhanças no espaço de permutações:

Solução $\pi = (\pi_1, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_j, \dots, \pi_n)$

$N1(\pi) = \{(\pi_1, \dots, \pi_{i+1}, \pi_i, \dots, \pi_n) : i=1, \dots, n-1\}$

Vizinhos de $(1,2,3,4) = \{(2,1,3,4), (1,3,2,4), (1,2,4,3)\}$

$N2(\pi) = \{(\pi_1, \dots, \pi_j, \dots, \pi_i, \dots, \pi_n) : i=1, \dots, n-1; j=i+1, \dots, n\}$

Vizinhos de $(1,2,3,4) = \{(2,1,3,4), (1,3,2,4), (1,2,4,3), (4,2,3,1), (2,3,1,4), (1,4,3,2)\}$

Vizinhança

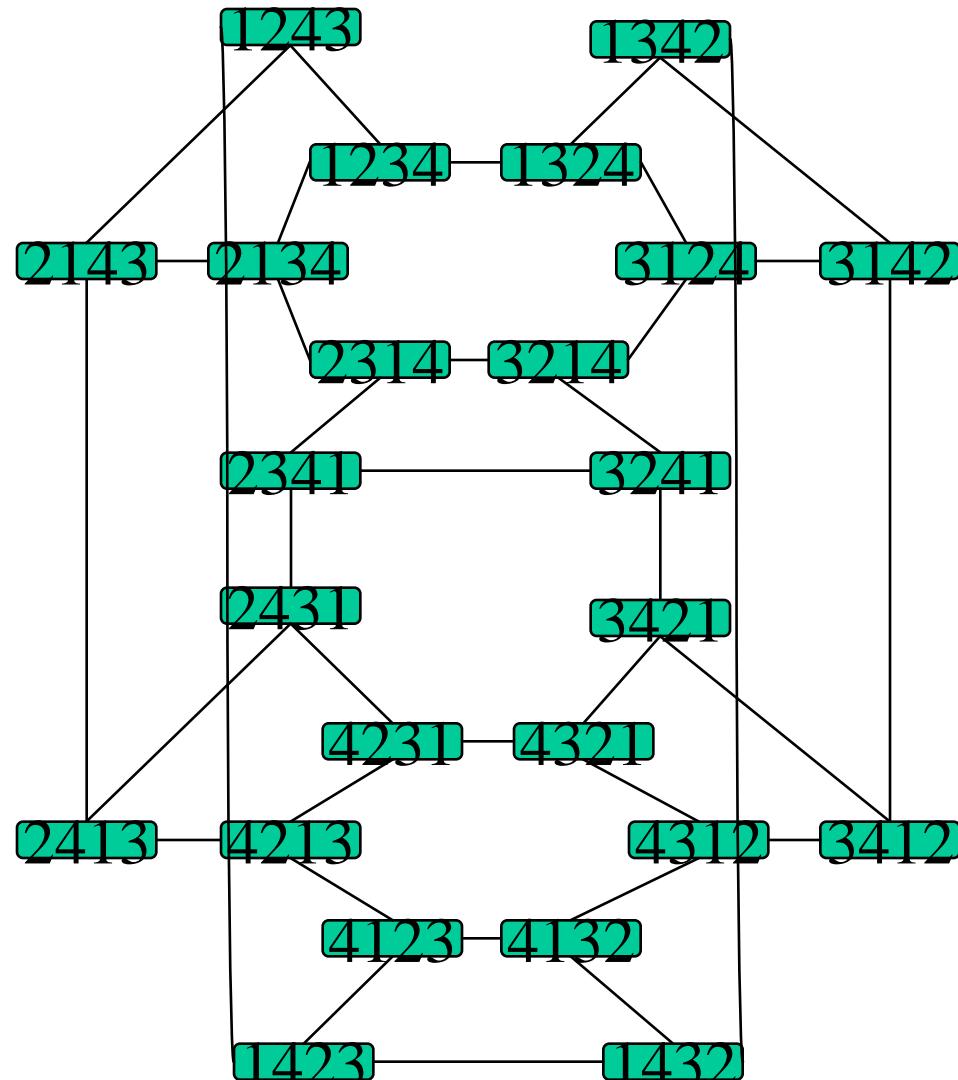
O espaço de busca pode ser visto como um grafo onde os vértices são as soluções e existem arestas entre pares de vértices associados a soluções vizinhas.

Um movimento é uma operação que transforma uma solução em uma solução vizinha.

Um caminho no espaço de busca consiste numa seqüência de soluções, onde duas soluções consecutivas quaisquer são vizinhas.

Vizinhança/Espaço de Busca

Exemplo 1: espaço de busca para a vizinhança N_1 sobre um conjunto solução de 4 dígitos.



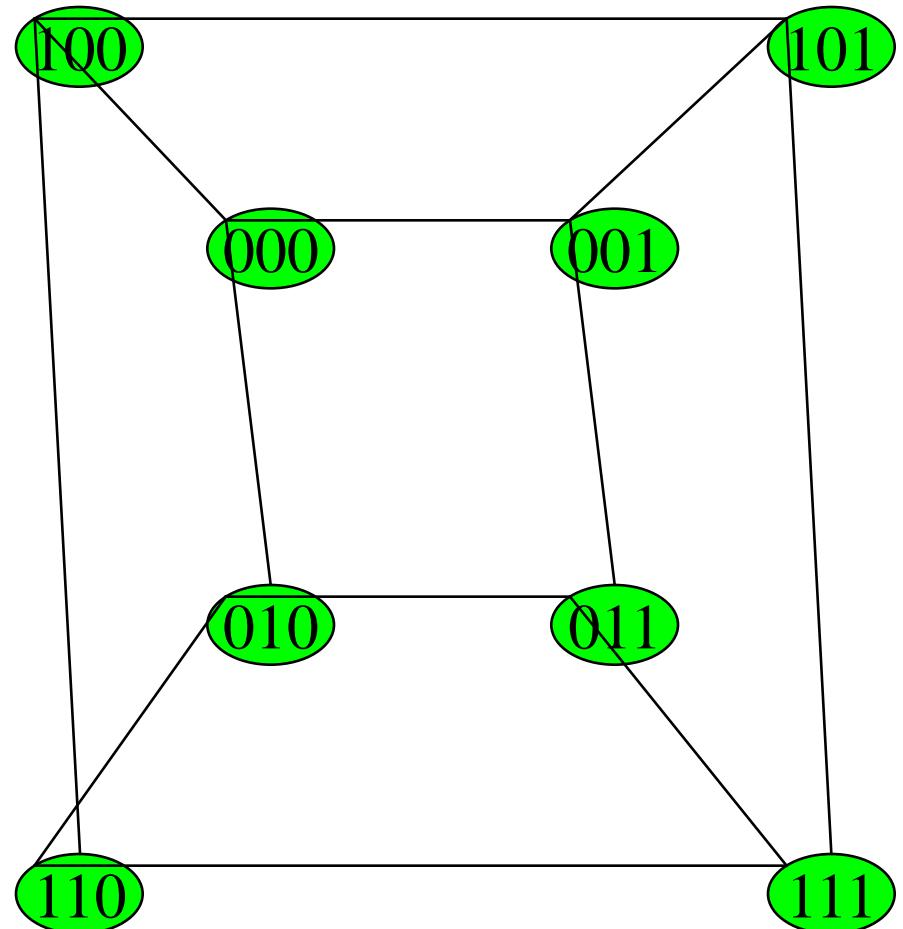
Vizinhança/Espaço de Busca

Exemplo 2: vetores de pertinência 0-1 com a vizinhança $N3$ definida como:

$$v = (v_1, \dots, v_i, \dots, v_n)$$

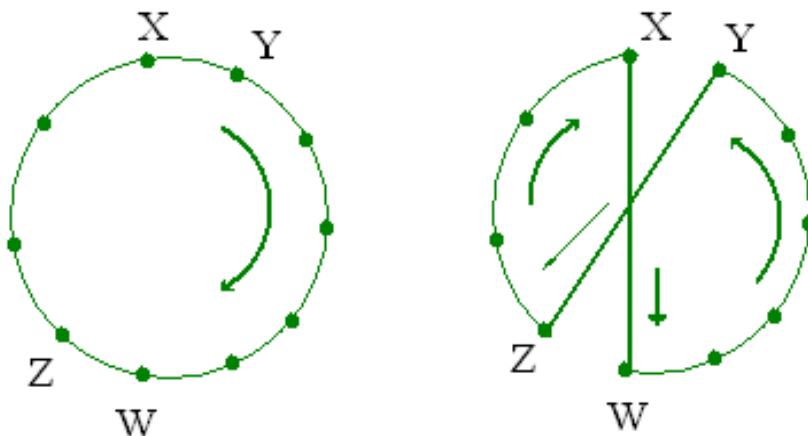
$$N3(v) = \{(v_1, \dots, 1-v_i, \dots, v_n) : i=1, \dots, n\}$$

Vizinhos de $(1,0,1) = \{(0,0,1), (1,1,1), (1,0,0)\}$



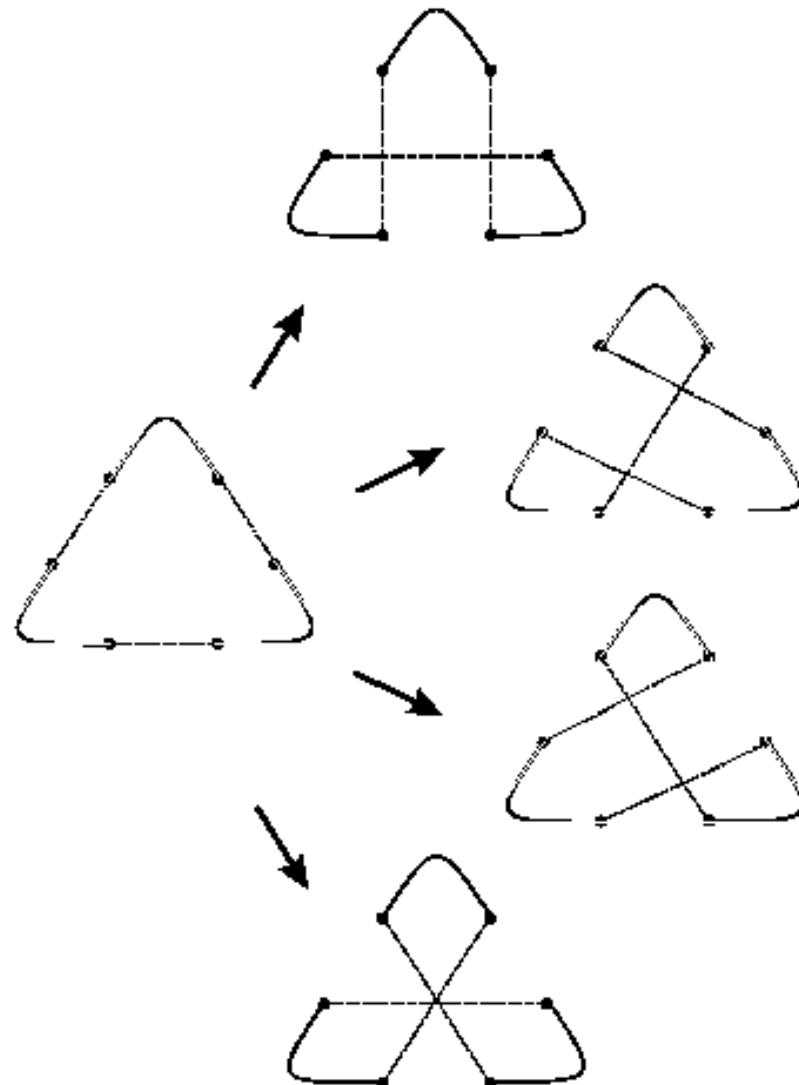
Vizinhança - mais exemplos

- K-opt: uma heurística de busca local aplicada no problema do caixeiro viajante. A vizinhança $N(s)$ é formada por ciclos obtidos pela substituição de k arestas do ciclo corrente S por k arestas que não estão no ciclo.
- Ex: 2-opt

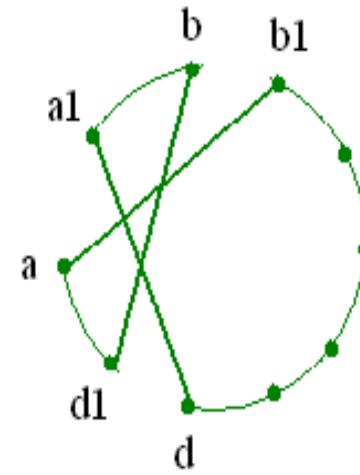
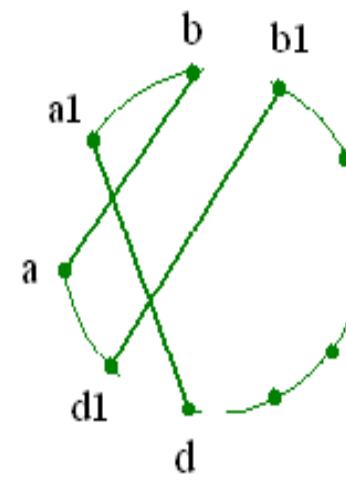
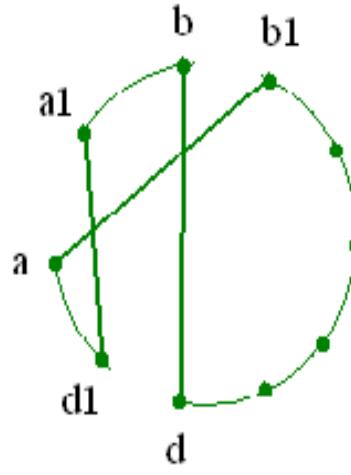
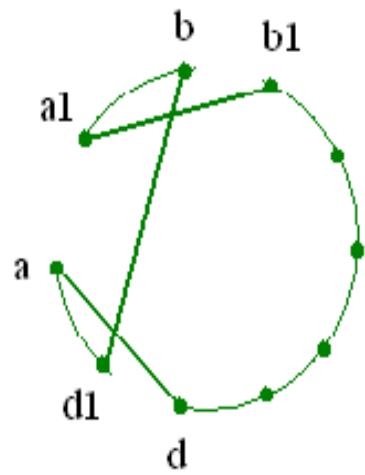
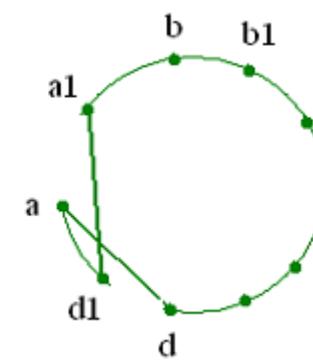
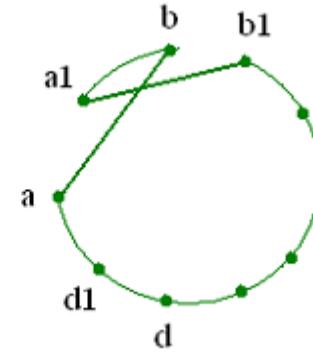
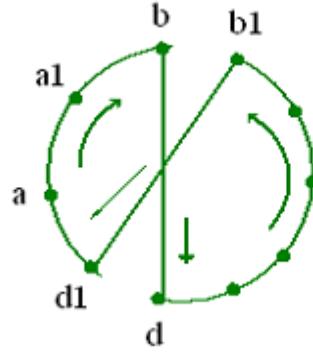
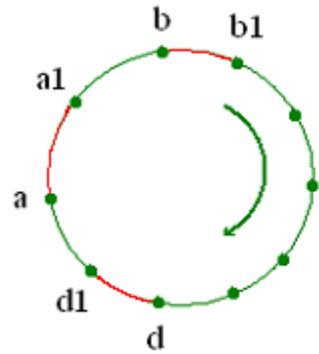


Vizinhança - mais exemplos

Ex: vizinhança 3-opt
para o PCV.



Vizinhança 3-Opt para o PCV



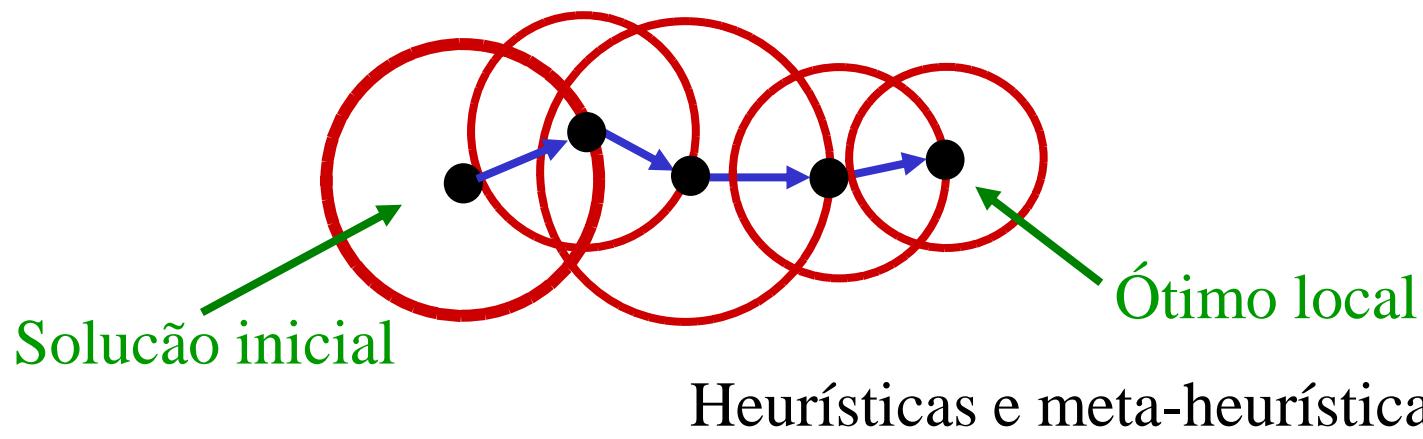
Busca Local

Um algoritmo de Busca Local explora a estrutura de vizinhança definida para o problema em questão.

Existem variações de algoritmos de Busca Local.
Dois exemplo clássicos:

First Improvement - Melhoria Iterativa;

Best Improvement - Descida Mais Rápida;



Busca Local

Métodos clássicos de busca loca:

Best Improvement

First Improvement

Busca Local

- Melhoria iterativa: a cada iteração, selecionar qualquer (eventualmente a primeira) solução aprimorante na vizinhança

```
procedure FirstImprovement( $s_0$ )
     $s \leftarrow s_0$ ; melhoria  $\leftarrow$  .verdadeiro.
    while melhoria do
        melhoria  $\leftarrow$  .falso.
        for-all  $s' \in N(s)$  e melhoria = .falso. do
            if  $f(s') < f(s)$  then
                 $s \leftarrow s'$ ; melhoria  $\leftarrow$  .verdadeiro.
            end-if
        end-for-all
    end-while
return  $s$ 
end Melhoria-Iterativa
```

Busca Local

Descida mais rápida (Steepest Descent – Best Improvement):
selecionar a melhor solução aprimorante na vizinhança

```
procedure BestImprovement( $s_0$ )
     $s \leftarrow s_0$ ; melhoria  $\leftarrow$  .verdadeiro.
    while melhoria do
        melhoria  $\leftarrow$  .falso.;  $f_{\min} \leftarrow +\infty$ 
        for-all  $s' \in N(s)$  do
            if  $f(s') < f_{\min}$  then
                 $s_{\min} \leftarrow s'$ ;  $f_{\min} \leftarrow f(s')$ 
            end-if
        end-for-all
        if  $f_{\min} < f(s)$  then
             $s \leftarrow s_{\min}$ ; melhoria  $\leftarrow$  .verdadeiro.
        end-if
    end-while
    return  $s$ 
end Descida-Mais-Rápida
```

Busca Local

Exemplo: algoritmo de descida mais rápida aplicado ao problema de ordenação

Espaço de busca: permutações de n elementos

Solução $\pi = (\pi_1, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_j, \dots, \pi_n)$

Vizinhança: $N1(\pi) = \{(\pi_1, \dots, \pi_{i+1}, \pi_i, \dots, \pi_n) : i=1, \dots, n-1\}$

Custo de uma permutação: $f(\pi) = \sum_{i=1, \dots, n} i \cdot \pi_i$

Busca Local

procedure BL-Perm-N1(π_0)

$\pi \leftarrow \pi_0$; melhoria \leftarrow .verdadeiro.

while melhoria **do**

melhoria \leftarrow .falso.; $f_{\min} \leftarrow +\infty$

for $i = 1$ **to** $n-1$ **do**

$\pi' \leftarrow \pi$; $\pi'_i \leftarrow \pi_{i+1}$; $\pi'_{i+1} \leftarrow \pi_i$;

if $f(\pi') < f_{\min}$ **then**

$\pi_{\min} \leftarrow \pi'$; $f_{\min} \leftarrow f(\pi')$

end-if

end-for

if $f_{\min} < f(\pi)$ **then**

$\pi \leftarrow \pi_{\min}$; melhoria \leftarrow .verdadeiro.

end-if

end-while

$\pi^+ \leftarrow \pi$

return π^+

end BL-Perm-N1

Busca Local - Prob. da Mochila

- (H1)
 - * Iniciar com uma solução (aleatória ou com um método de construção).
 - * retirar da mochila o item que possui o menor custo.
 - * colocar na mochila o item que possui o maior custo, tal que
 - melhore o valor da função objetivo.
 - não torne a solução infactível.
 - * Termine quando não houver mais movimentos ou um determinado número fixo de movimentos tenha sido realizado

Busca Local - Prob. da Mochila

- (H2)
 - * Iniciar com uma solução (aleatória ou com um método de construção).
 - * colocar na mochila o item com maior custo que esteja fora da mochila.
 - * Se infactível, então retire os elementos com menor custo até que a solução fique factível.
 - * Termine quando não houver mais movimentos ou um determinado número fixo de movimentos tenha sido realizado

Busca Local

Diferentes aspectos do **espaço de busca** influenciam o desempenho de algoritmos de busca local

Conexidade: deve existir um caminho entre qualquer par de soluções no espaço de busca

Distância entre duas soluções: número de soluções visitadas ao longo de um caminho mais curto entre elas

Diâmetro: distância entre duas das soluções mais afastadas (diâmetros reduzidos)

Bacia de atração de um ótimo local: conjunto de soluções iniciais a partir das quais o algoritmo de **descida mais rápida** leva a este ótimo local.

Método Construtivo X Busca Local

- Sensível ao ponto de partida.
- Complexidade de tempo linear : $O(|E|)$.
- Sensível à solução de partida
- A complexidade de tempo pode ser exponencial
- Sensível à vizinhança escolhida
- Sensível à estratégia de busca

Extensões para contornar algumas dificuldades da busca local

Redução da vizinhança: investigar um subconjunto da vizinhança da solução corrente (e.g. por aleatorização)

Multi-partida: repetir a busca local a partir de diferentes soluções

Multi-vizinhança: considera mais de uma vizinhança. Ao atingir um ótimo local com relação a uma vizinhança, inicia uma outra busca local empregando outra vizinhança. O algoritmo termina quando a solução corrente é um ótimo local em relação a todas as vizinhanças empregadas.

Segmentação da vizinhança: utilizada para aumentar a eficiência quando vizinhanças muito grandes são utilizadas, pode ser vista como uma estratégia multi-vizinhança. $N(s) = N_1(s) \cup N_2(s) \cup \dots \cup N_p(s)$

Meta-heurística

- Meta-heurísticas são modelos gerais que servem como guia para construção de algoritmos heurísticos.
- Muitos dos modelos são baseados na natureza (físicos, biológicos, teoria da evolução)
- As estratégias meta-heurísticas tem como objetivo superar as falhas da busca local, como por exemplo, o término prematuro em um ótimo local.

Meta-heurística

Algoritmos Evolucionário:

- Algoritmos Genéticos
- Programação Genética
- Evolução Diferencial
- Algoritmos Culturais
- Algoritmos Meméticos

Algoritmos Físicos:

- Simulated Annealing*
- Busca Harmônica
- Electromagnetismo

Algoritmos Populacionais

- PSO - Particle Swarm Optimization*
- Ant System*
- Colônia de Formigas - Ant Colony System*
- Colméia de Abelhas - Bees Algorithm*
- Bacterial Foraging Optimization Algorithm*

Algoritmos Ad-hocs

- GRASP – Greedy Randomized Adaptive Search Procedure*
- ILS - Iterated Local Search*
- GLS - Guided Local Search*
- VNS - Variable Neighborhood Search*
- Busca Dispersa - Scatter Search*
- Busca Tabu*
- Time Assíncrono*

Algumas Meta-heurísticas

- Variable Neighborhood Search (VNS)
- Variable Neighborhood Descent (VND)
- GRASP (Greedy Randomized Adaptive Search Procedures)
- *Simulated Annealing*
- Busca Tabu
- Algoritmos Genéticos
- *Ant System*

Vantagens das Meta-heurísticas

- Flexibilidade de adaptação à diversos problemas
- Facilidade de escapar de soluções locais

Bibliografia

- A. Diaz, F. Glover, H. M. Ghaziri, J. L. González, M. Laguna, P. Moscato e F. T. Tseng, *Optimización Heurística Y Redes Neurolales*, Editorial Paraninfo, Espanha, 1996.
- Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-verlag, 1992.
- C. R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, 1993.
- F. Glover e M. Laguna, *Tabu Search*, Kluwer Academic Publishers, USA, 1997.
- V. J. Rayward-smith, I. H. Osman, C. R. Reeves e G. D. Smith, *Modern Heuristic Search Methods*, Wiley, Inglaterra, 1996.
- <http://www.cleveralgorithms.com/nature-inspired/index.html>