Optimização Combinatória: Modelos e Algoritmos

Slide 1

Transparências de apoio à leccionação de aulas teóricas

Versão 1

©2001

José Fernando Oliveira, Maria Antónia Carravilla – FEUP

Modelos de Optimização Combinatória

Slide 2

Problemas de Optimização

Instância de um Problema de Optimização

Uma instância de um Problema de Optimização é um par $(\mathcal{F}, c)$, onde:

$\mathcal{F}$ é um conjunto qualquer (o domínio das soluções admissíveis)

c é a função custo, e corresponde a um mapeamento $c : \mathcal{F} \rightarrow \mathcal{R}$

Slide 3

A solução óptima será um $f \in \mathcal{F}$ tal que:

$$\forall_{y \in \mathcal{F}} \quad c(f) \leq c(y)$$

Um **Problema de Optimização** é um conjunto de **Instâncias de um Problema de Optimização**

(in Papadimitriou, Steiglitz pp.4)

Optimização Combinatória *OC*

Slide 4

Problemas de Optimização

com variáveis contínuas

Problemas Contínuos

Solução: conjunto de números reais

com variáveis discretas

Problemas Combinatórios

Solução: objecto pertencente a um conjunto finito ou então infinito enumerável, por exemplo inteiros, conjuntos, permutações ou grafos.

Problema da Mochila
"Knapsack Problem" *KP*

Slide 5

Dados:

- um conjunto de tipos de objectos em que cada tipo tem um valor e um peso associados
- uma mochila com um limite de peso transportável

Pretende-se encher a mochila, não ultrapassando o limite máximo de peso e maximizando o valor total dos objectos transportados.

Problema da Mochila (*KP*)

Índices

j tipo de objecto, $j \in \{1, \ldots, n\}$;

Variáveis de decisão

x_j número de objectos do tipo j a colocar dentro da mochila

Slide 6

Coeficientes

w_j peso de cada um dos objectos do tipo j;

c_j valor de cada um dos objectos do tipo j;

b limite máximo de peso a transportar na mochila.

Função objectivo

$$\max\ Z = \sum_{j=1}^{n} c_j x_j$$

Restrições

$$\sum_{j=1}^{n} w_j x_j \leq b$$

$$\forall_j \quad x_j \geq 0 \quad \textit{inteiro}$$

Problema da Mochila 0–1 "0–1 Knapsack Problem"(*0–1KP*)

Slide 7

Dados:

- um conjunto de objectos diferentes em que cada objecto tem um valor e um peso associados
- uma mochila com um limite de peso transportável

Pretende-se encher a mochila, não ultrapassando o limite máximo de peso e maximizando o valor total dos objectos transportados.

Problema da Mochila 0–1 (*0–1KP*) Modelo

Slide 8

Índices

j objecto, $j \in \{1, \ldots, n\}$;

Variáveis de decisão

$$x_j = \begin{cases} 1 & \text{se objecto } j \\ & \text{for colocado na mochila} \\ 0 & \text{se não} \end{cases}$$

Coeficientes

w_j peso do objecto j;

c_j valor do objecto j;

b limite máximo de peso a transportar na mochila.

Função objectivo

$$\max\ Z \quad = \sum_{j=1}^{n} c_j x_j$$

Restrições

$$\sum_{j=1}^{n} w_j x_j \leq b$$

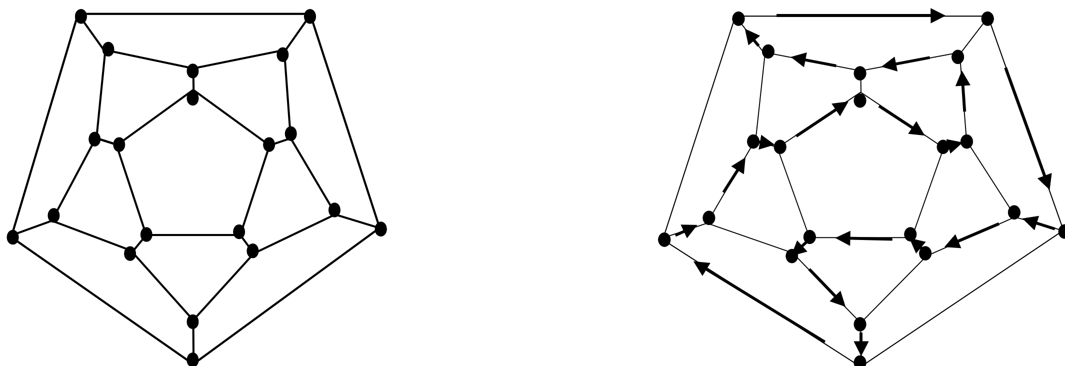
$$\forall_j \qquad x_j \in \{0, 1\}$$

Circuitos Hamiltonianos

Slide 9

Definição de Circuito Hamiltoniano: Um circuito diz-se Hamiltoniano, se passar uma e uma só vez por todos os vértices de uma rede.

A designação provém do islandês Hamilton que, em 1857 propôs um jogo denominado “Around the World”. Nesse jogo, os vértices de um dodecaedro de madeira representavam as 20 cidades mais importantes do mundo na época. O objectivo do jogo consistia em encontrar um percurso através dos vértices do dodecaedro, com início e fim no mesmo vértice (cidade) e que passasse por cada vértice (cidade) apenas uma vez.



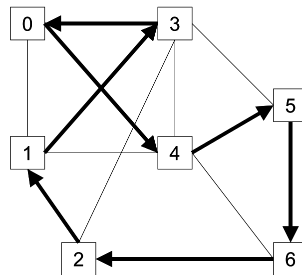
Problema do Caixeiro Viajante
"Travelling Salesman Problem"(*TSP*)

O problema do caixeiro viajante é um problema de optimização associado à determinação dos circuitos hamiltonianos num grafo qualquer.

Problema do Caixeiro Viajante:

Slide 10

Pretende-se encontrar o caminho mais curto para um caixeiro viajante que sai de uma cidade, visita n outras cidades e volta àquela de onde partiu, sem repetir nenhuma das cidades visitadas (Circuito Hamiltoniano mais curto).



Problema do Caixeiro Viajante (*TSP*)
Formulação de Dantzig-Fulkerson-Johnson

Formulação do TSP como um problema de programação binária sobre um grafo $G = (V, A)$, onde V é o conjunto de vértices (cidades) e A é o conjunto de arcos (percursos directos entre duas cidades)

Índices

Slide 11

i cidade, $i \in \{1, \ldots, n\}$

j cidade, $j \in \{1, \ldots, n\}$

Coeficientes

d_{ij} custo associado ao percurso (arco) entre cidade i e cidade j.

Variáveis de decisão

$$x_{ij} = \begin{cases} 1 & \text{se o percurso directo (arco) de } i \\ & \text{para } j \text{ estiver incluído na solução} \\ 0 & \text{se não} \end{cases}$$

Problema do Caixeiro Viajante (*TSP*)
Formulação de Dantzig-Fulkerson-Johnson (cont.)

Slide 12

Função objectivo

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

Restrições

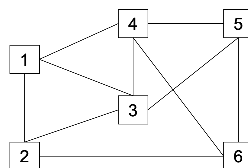
$$\begin{array}{rcll} \sum_{i=1}^{n} x_{ij} & = & 1 & \forall_{j \in V} \\ \sum_{j=1}^{n} x_{ij} & = & 1 & \forall_{i \in V} \\ \sum_{i,j \in S} x_{ij} & \leq & |S|-1 & \forall_{S \subset V} \\ x_{ij} & \in & \{0,1\} & \forall_{i,j \in V, i \neq j} \end{array}$$

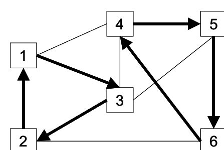
$|S|$ representa o número de vértices do subgrafo S.

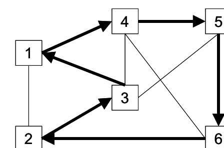
Note-se que em $S \equiv V$ não está considerado em $S \subset V$.

Problema do Caixeiro Viajante (*TSP*)
Formulação de Dantzig-Fulkerson-Johnson
Eliminação de subgrafos

Slide 13







$$S = \{1,3,4\}$$

$$x_{13} = 1 \leq |S| - 1 = 2$$

$$S = \{4,5,6\}$$

$$x_{45} + x_{56} + x_{64} = 3 \not\leq |S| - 1 = 2$$

$$S = \{1,2,3\}$$

$$x_{23} + x_{31} = 2 \leq |S| - 1 = 2$$

$$S = \{4,5,6\}$$

$$x_{45} + x_{56} = 2 \leq |S| - 1 = 2$$

etc...

Cobertura de conjuntos "Set Covering"

Slide 14

Dados:

- um conjunto de clientes
- um conjunto de armazéns
- uma matriz de ligações clientes–armazéns
- custos de abertura de cada um dos armazéns

Pretende-se fornecer todos os clientes, minimizando os custos de abertura dos armazéns.

Cobertura de conjuntos — Modelo

Slide 15

Índices

i cliente, $i \in \{1, \ldots, m\}$;

j armazém, $j \in \{1, \ldots, n\}$;

Variáveis de decisão

$$x_j = \begin{cases} 1 & \text{se armazém } j \text{ for aberto.} \\ 0 & \text{se não} \end{cases}$$

Coeficientes

a_{ij} 1 se cliente i pode ser fornecido pelo armazém j, 0 se não;

c_j custo associado à abertura do armazém j.

Função objectivo

$$\min Z = \sum_{j=1}^{n} c_j x_j$$

Restrições

$$\forall_i \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1$$

$$\forall_j \quad x_j \in \{0, 1\}$$

Partição de conjuntos "Set Partitioning"

Slide 16

Dados:

- um conjunto de clientes
- um conjunto de armazéns
- uma matriz de ligações clientes–armazéns
- custos de abertura de cada um dos armazéns

Pretende-se fornecer todos os clientes, minimizando os custos de abertura dos armazéns. **Cada cliente só pode ficar ligado a um armazém.**

Partição de conjuntos "Set Partitioning" — Modelo

Slide 17

Índices

i cliente, $i \in \{1, \ldots, m\}$;

j armazém, $j \in \{1, \ldots, n\}$;

Variáveis de decisão

$$x_j = \begin{cases} 1 & \text{se armazém } j \text{ for aberto.} \\ 0 & \text{se não} \end{cases}$$

Coeficientes

a_{ij} 1 se cliente i pode ser fornecido pelo armazém j, 0 se não;

c_j custo associado à abertura do armazém j.

Função objectivo

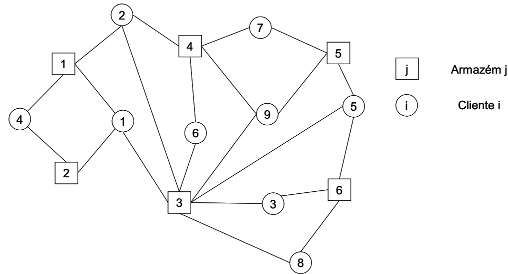
$$\min Z = \sum_{j=1}^{n} c_j x_j$$

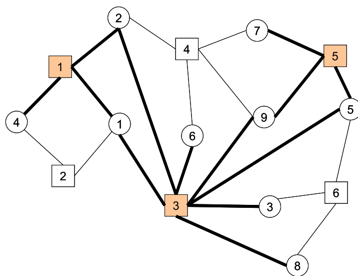
Restrições

$$\forall_i \quad \sum_{j=1}^{n} a_{ij} x_j = 1$$
$$\forall_j \quad x_j \in \{0, 1\}$$

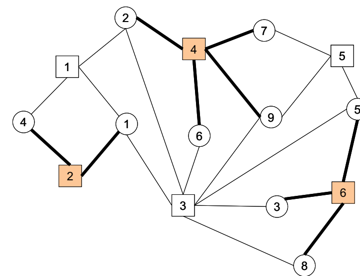
Cobertura e Partição de conjuntos – um exemplo

Slide 18





Uma solução possível para o problema de cobertura de conjuntos.



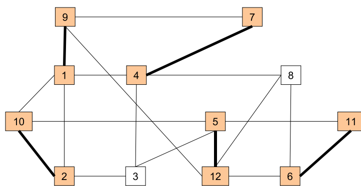
Uma solução possível para o problema de partição de conjuntos.

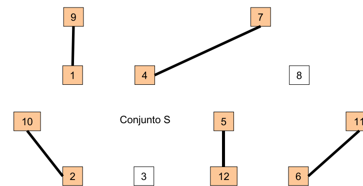
Emparelhamento "Matching"

Slide 19

Dado um grafo $G = (V, A)$, onde V é o conjunto de vértices e A é o conjunto de arcos, denomina-se "grau do vértice j"ao número de arcos que tocam o vértice j.

O problema básico de emparelhamento consiste em, dado um grafo $G = (V, A)$, encontrar o maior subconjunto S de arcos, tal que o grau de todos os vértices seja ≤ 1.





Emparelhamento "Matching" — Modelo

Slide 20

Índices

i arco, $i \in \{1, \ldots, m\}]$;

j vértice, $j \in \{1, \ldots, n\}$;

Variáveis de decisão

$$x_i = \begin{cases} 1 & \text{se arco } i \text{ for seleccionado.} \\ 0 & \text{se não} \end{cases}$$

Função objectivo

$$\max\ Z \quad = \textstyle\sum_{i=1}^{n} x_i$$

Restrições

$$\forall_j \quad \textstyle\sum_{i \in \delta(j)} x_i \leq 1$$
$$\forall_i \quad x_j \in \{0, 1\}$$

$\delta(j)$ representa o conjunto de arcos que tocam o vértice j.

Localização "Location"

Slide 21

Dados:

- um conjunto de clientes
- um conjunto de armazéns
- uma matriz de custos de fornecimento de clientes pelos armazéns
- custos de abertura de cada um dos armazéns

Pretende-se fornecer todos os clientes, minimizando os custos de abertura dos armazéns e os custos de fornecimento dos clientes.

Localização "Location" Modelo

Slide 22

Índices

i cliente, $i \in \{1, \ldots, m\}$;

j armazém, $j \in \{1, \ldots, n\}$;

Variáveis de decisão

$$x_j = \begin{cases} 1 & \text{se armazém } j \text{ for aberto.} \\ 0 & \text{se não} \end{cases}$$

$y_{i,j}$ fracção da procura do cliente i a partir do armazém j;

Coeficientes

h_{ij} custo de fornecer cliente i a partir do armazém j;

c_j custo associado à abertura do armazém j.

Localização "Location" Modelo (cont.)

Slide 23

Função objectivo

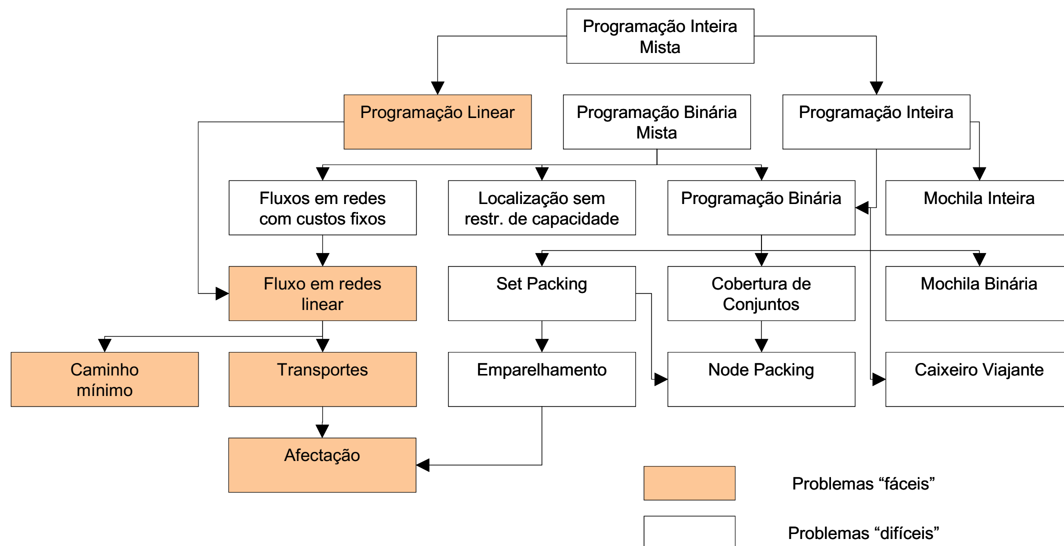
$$\min \ Z = \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ij} y_{ij}$$

Restrições

$$\begin{aligned} \forall_i \quad & \sum_{j=1}^{n} y_{ij} = 1 \\ \forall_i \forall_j \quad & y_{ij} - x_j \leq 0 \\ \forall_j \quad & x_j \in \{0, 1\} \end{aligned}$$

Os Problemas de Optimização Combinatória e a Teoria da Complexidade

Slide 24



(in Nemhauser, Wolsey pp.116)

Teoria da complexidade – definições

Slide 25

Algoritmo – procedimento composto por uma sequência de passos para resolver um problema.

Problema – domínio contendo instâncias do problema + uma questão que pode ser colocada acerca de qualquer uma das instâncias.

Instância – Realização ou concretização da abstracção a que se dá o nome de problema.

Exemplo: Problema do caixeiro viajante (TSP)

- Instância – número inteiro n (conjunto de cidades) e uma matriz $n \times n$ $\mathbf{C} = [c_{ij}]$, onde cada c_{ij} é um valor não-negativo (“distância”).
- Questão – Qual é a permutação cíclica π dos inteiros de 1 até n tal que a soma $\sum_{i=1}^{n} c_{i\pi i}$ é minimizada?

Diz-se que um algoritmo resolve um problema P se, dada qualquer instância I de P, ele gerar uma resposta à questão de P para I.

Como medir a eficiência de um algoritmo?

Slide 26

Estimativa do tempo de execução:

- dependerá da dimensão da instância (número de cidades n no TSP) e da velocidade do computador;
- logo faz-se uma análise da evolução do tempo de execução com n, em vez de uma análise pontual para uma dada dimensão, e usam-se passos de execução em vez de ciclos de relógio do processador.

A notação O-maiúsculo adapta-se particularmente a estes objectivos:

Diz-se que o tempo de execução de um algoritmo é $O(f(n))$ se existir uma constante c tal que o tempo de execução para todas as entradas (instâncias) de dimensão n seja limitado por $cf(n)$.

Como medir a eficiência de um algoritmo? (exemplo)

Slide 27

```
min = infinito
para todas as permutações cíclicas π
    custo = 0;
    para todo i = 1 até n
        custo = custo + c_{i π(i)}
        se custo < min então
            min = custo
            melhor circuito = π
resultado = melhor circuito
```

As instruções dentro do ciclo 2 são executadas n vezes. Por sua vez, o ciclo 2, porque está dentro do ciclo 1, é executado $(n-1)!$ vezes (há $(n-1)!$ permutações cíclicas de n números). Então, as instruções que são executadas mais vezes são executadas $n \times (n-1)! = n!$. Diz-se que o algoritmo é de ordem n factorial – $O(n!)$.

Análise "worst-case"

Este algoritmo é bem comportado, no sentido de que requer sempre o mesmo número de ciclos. Por vezes os algoritmos abandonam os ciclos a meio, ou há ciclos cuja execução é condicional ⇒ tempo de execução a variar de instância para instância, mesmo com iguais tamanhos (n).

Alternativas:

Slide 28

- análise "pior dos casos" – considera-se que todos os ciclos e instruções são sempre executados; ⟶ análise mais frequente.
- análise de valores médios – simulam-se muitas instâncias e toma-se o número médio de passos que a sua resolução implicou.

Algoritmos "bons" e algoritmos "maus"

Slide 29

Evolução do crescimento de várias funções com n:

Função	Valores aproximados		
n	10	100	1 000
$n \log n$	33	664	9 966
n^3	1000	1 000 000	10^9
$10^6\ n^8$	10^{14}	10^{22}	10^{30}
2^n	1 024	1.27×10^{30}	1.05×10^{301}
$n^{\log n}$	2 099	1.93×10^{13}	7.89×10^{29}
$n!$	3 628 800	10^{158}	4×10^{2567}

Polinomial – **BOM**

Exponencial – **MAU**

Classificação dos problemas

Problemas de decisão (supõem apenas uma resposta do tipo SIM ou NÃO).

Exemplo: Para uma dada instância do TSP há algum circuito cujo custo (distância total percorrida) seja inferior a K?

Slide 30

- Classe $\mathcal{P}$ – Conjunto de problemas de decisão para os quais existe um algoritmo que corre em tempo polinomial.
- Classe $\mathcal{NP}$ – Conjunto de problemas de decisão para os quais não se conhece um algoritmo polinomial mas que que pode ser resolvido em tempo polinomial por uma abstracção algorítmica chamada “algoritmo não determinístico”.[a]

[a]Incorpora instruções do tipo “go to both label1, label2” originando um árvore de processos a correr em paralelo. O primeiro ramo que responder “SIM” pára a execução e o algoritmo responde “SIM”. Se esse ramo tiver respondido após um número polinomial de passos, então o algoritmo diz-se não-determinístico

Classificação dos problemas (cont.)

- Classe $\mathcal{NP}$ – *completa* – Sub-conjunto de problemas da classe $\mathcal{NP}$ aos quais qualquer outro problema da classe pode ser reduzido.

  Se qualquer problema da classe $\mathcal{NP}$ puder ser **reduzido** a um problema P então diz-se que P pertence à classe $\mathcal{NP}$ – *completa*.

Slide 31

Problemas de optimização (achar a melhor solução)

São naturalmente reduzidos a uma sequência de problemas de decisão: repete-se a pergunta, com valores sucessivamente mais exigentes, até a resposta ser não.

Abordagens à resolução de Problemas de Optimização Combinatória

Slide 32

Técnicas exactas — obtêm e garantem uma solução óptima.

Atingir a solução óptima pode ser difícil (muito demorado), ou mesmo impossível (o tempo correspondente à vida passada do sistema solar poderia não ser suficiente) e nem sequer ser especialmente importante para a aplicação concreta que se pretende resolver.

Técnicas aproximadas ou métodos heurísticos — ou não obtêm a solução óptima ou, se a obtêm, não o sabem... Em compensação são capazes de obter "boas" soluções muito rapidamente.

Bibliografia

Slide 33

- Goldbarg, Marco Cesar e Luna, Henrique Pacca (2000). *Otimização Combinatória e Programação Linear*, Editora CAMPUS.
- Golden, B.L. and Stewart, W.R. (1985). *Empirical analysis of heuristics* in The Traveling Salesman Problem, John Wiley & Sons, Inc..
- Nemhauser, George L. e Wolsey, Laurence A. (1988). *Integer and Combinatorial Optimization* John Wiley & Sons, Inc..
- Papadimitrio, Christus H. e Steiglitz, Kenneth (1982). *Combinatorial Optimization – Algotithms and Complexity* Prentice Hall, Inc..
- Sousa, Jorge Pinho (1991). *Apontamentos de Optimização Combinatória.*

Técnicas exactas para resolução de problemas de optimização

Slide 34

Técnicas exactas

Slide 35

- Enumeração explícita — por definição de problema de Optimização Combinatória, *gerando* e *avaliando* todas as *soluções admissíveis* é possível obter a *solução óptima*.
- Enumeração implícita — não se gerando todas as soluções admissíveis, elas são no entanto consideradas e *implicitamente avaliadas*.

  Exemplos: Método de pesquisa em árvore com enumeração e limitação ("branch and bound"); limites superiores e inferiores ao valor da solução óptima.
- Formulação dos problemas em modelos de programação inteira (variáveis de decisão assumem valores no conjunto dos números inteiros), ou mesmo binária (variáveis apenas com dois valores possíveis: 0 ou 1), e consequente resolução com algoritmos apropriados.

  Nota: Estas formulações podem também ser usadas para obter limites para o valor da solução óptima através de **relaxações**.

Técnicas exactas e relaxações

Relaxação — Não consideração de uma ou mais restrições do problema original PO, transformando-o num problema mais simples de resolver PR, sabendo-se que os valores óptimos das funções objectivo obedecem à seguinte relação (assumindo um problema de minimização):

Slide 36

$$f^{\star}_{PR} \leq f^{\star}_{PO}$$

(tradução: ao tirar restrições a solução só pode melhorar, ou ficar na mesma).

Relaxação linear – transformação de um problema em números inteiros num problema com variáveis reais (deixa-se cair a restrição "e inteiros" ou "$\in \mathcal{N}_0$" $\longrightarrow$ utilização do método simplex para a sua resolução em vez dos muito mais complexos (e extraordinariamente mais demorados) métodos de pesquisa em árvore).

Método de "branch and bound"

Baseia-se na ideia de uma enumeração inteligente das soluções candidatas a solução óptima inteira de um problema, efectuando *sucessivas partições* do espaço das soluções e *cortando a árvore de pesquisa* através da consideração de limites calculados ao longo da enumeração.

Slide 37

Representação gráfica

Slide 38

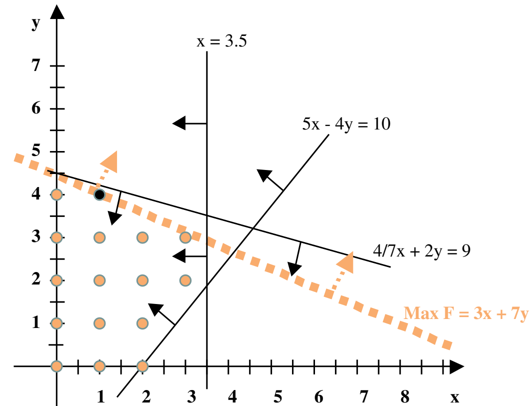
Considere-se o seguinte problema de Programação Inteira:
Maximizar:

$$F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcl} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \textit{ e inteiras} \end{array}$$

e a sua representação gráfica:



Solução óptima inteira: $x = 1$ e $y = 4$.

Resolução da relaxação linear

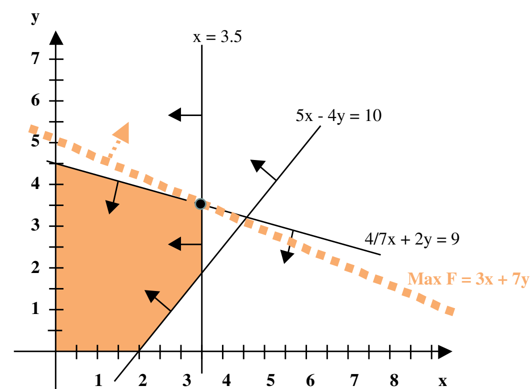
Slide 39

Problema $\mathcal{PL}0$:

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcl} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \end{array}$$



Solução óptima não inteira:
$x = 3.5$ e $y = 3.5$; $F = 35$

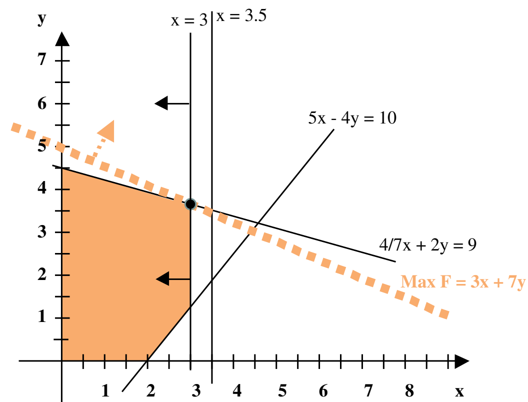
Ramificação em x: $x \leq 3$

Slide 40

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcr} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \end{array}$$



Solução (não inteira):
$x = 3$ e $y = 3.6$; $F = 34.5$

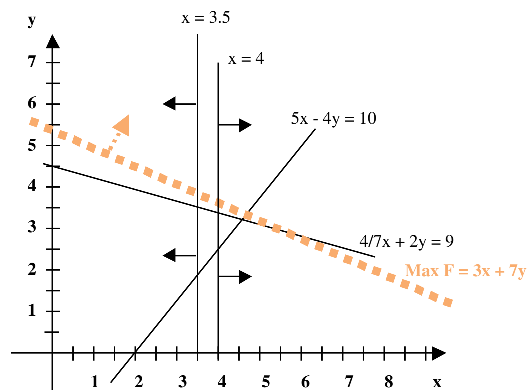
Ramificação em x: $x \geq 4$

Slide 41

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcr} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \geq & 4 \end{array}$$



Sem soluções admissíveis.

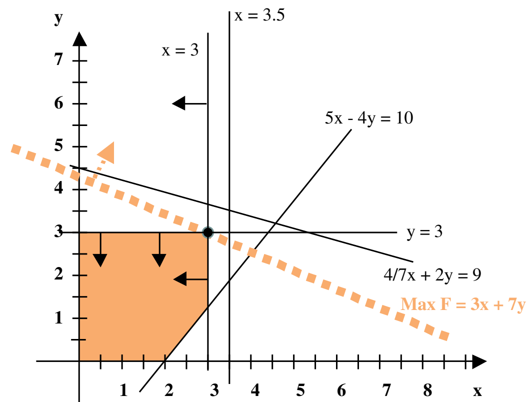
Ramificação em y: $y \leq 3$

Slide 42

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcl} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \\ & & y & \leq & 3 \end{array}$$



Solução (inteira):
$x = 3$ e $y = 3$; $F = 30$
Obtenção de um limite inferior $\Rightarrow$ Soluções não inteiras com valor de F inferior ou igual a 30 não precisam de ser exploradas!

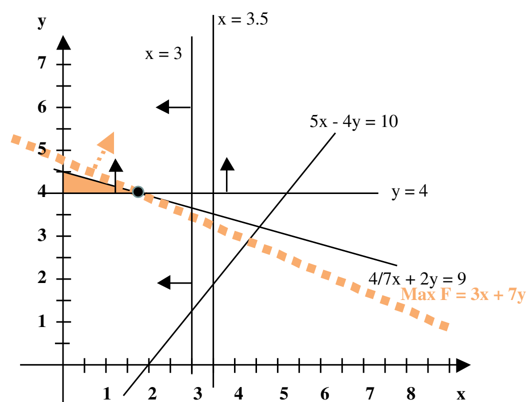
Ramificação em y: $y \geq 4$

Slide 43

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcl} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \\ & & y & \geq & 4 \end{array}$$



Solução (não inteira):
$x = 1.7$ e $y = 4$; $F = 33.2$

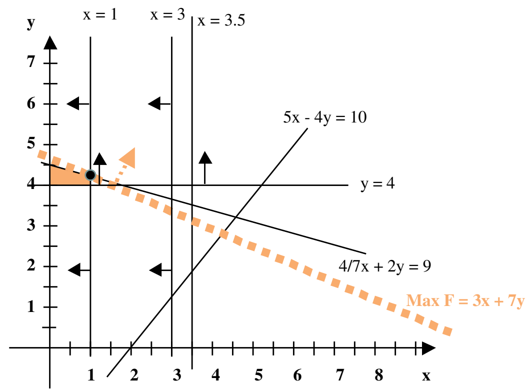
Ramificação em x: $x \leq 1$

Slide 44

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcr} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \\ & & y & \geq & 4 \\ x & & & \leq & 1 \end{array}$$



Solução (não inteira):
$x = 1$ e $y = 4.2$; $F = 32.5$

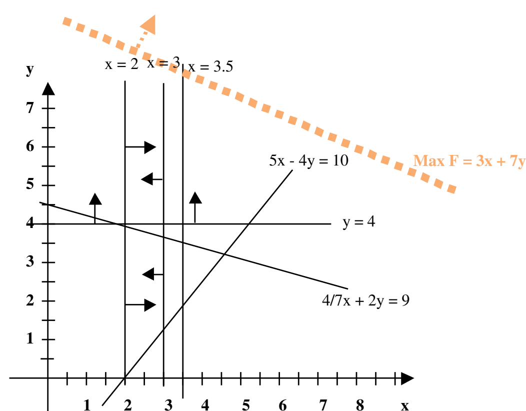
Ramificação em x: $x \geq 2$

Slide 45

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcr} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \\ & & y & \geq & 4 \\ x & & & \geq & 2 \end{array}$$



Sem soluções admissíveis.

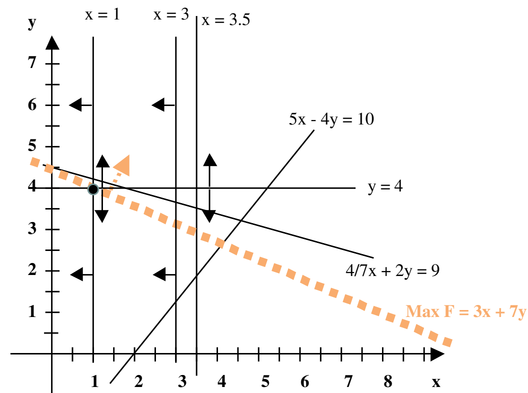
Ramificação em y: $y \leq 4$

Slide 46

$$\max F = 3x + 7y$$

suj. a:

$$\begin{array}{rcrcr} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \\ & & y & \geq & 4 \\ x & & & \leq & 1 \\ & & y & \leq & 4 \end{array}$$



Solução (inteira):
$x = 1$ e $y = 4$; $F = 31$
Melhor solução inteira até ao momento!

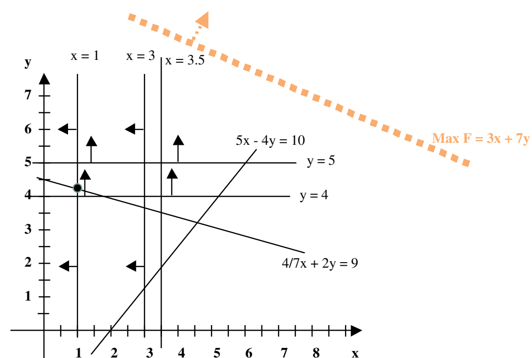
Ramificação em y: $y \geq 5$

Slide 47

$$\max F = 3x + 7y$$

suj. a:

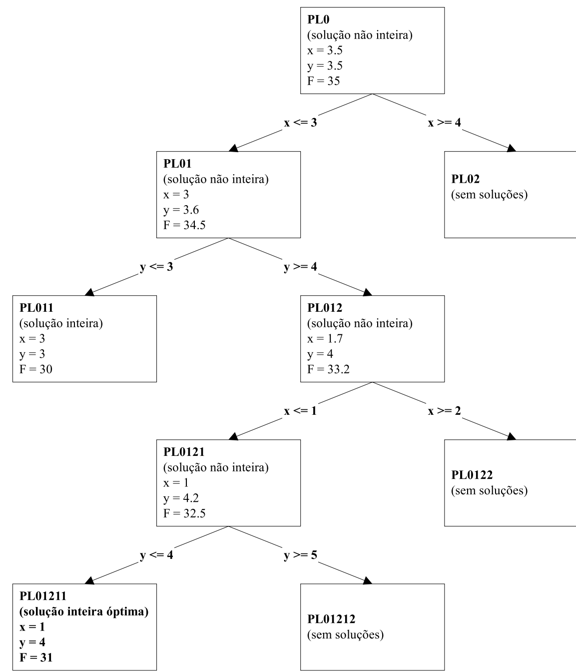
$$\begin{array}{rcrcr} x & & & \leq & 3.5 \\ 5x & - & 4y & \leq & 10 \\ \frac{4}{7}x & + & 2y & \leq & 9 \\ x & , & y & \geq & 0 \\ x & & & \leq & 3 \\ & & y & \geq & 4 \\ x & & & \leq 1 & \\ & & y & \geq & 5 \end{array}$$



Sem soluções admissíveis.

Árvore de pequisa do "Branch-and-Bound"

Slide 48



Limites

Slide 49

Limites (inferiores e superiores):

- tornam o algoritmo de "branch & bound" mais eficiente ao permitir descartar nós da árvore de pesquisa ainda não completamente explorados, pela certeza de que nunca originarão soluções melhores do que as que já temos.
- permitem "medir a distância" (em termos de valor da função objectivo) a que estamos da solução óptima.

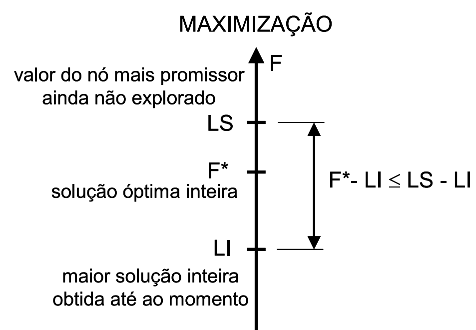
Limites

Num problema de **maximização**:

- um limite **inferior** LI é dado por uma solução inteira que se tenha já obtido – a solução óptima $F^{\star}$ nunca poderá ser pior (inferior) que a solução inteira que já temos;

Slide 50

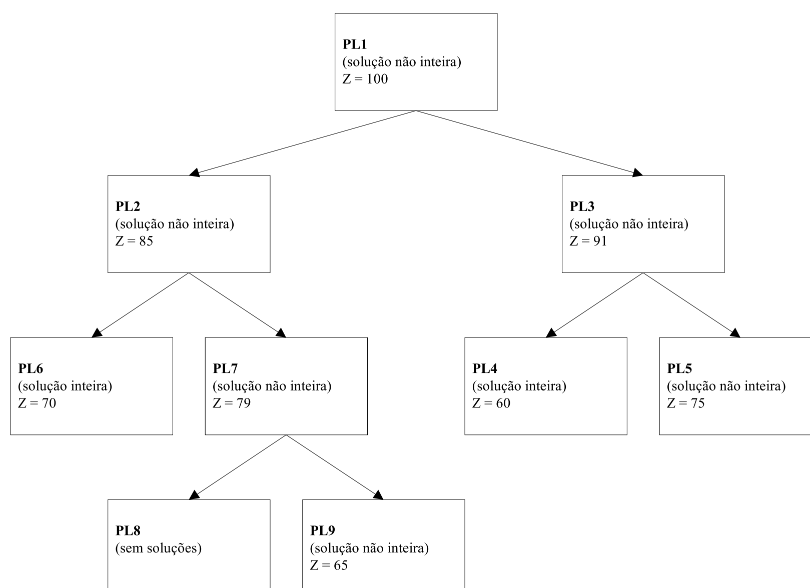
- um limite **superior** LS será dado pelo maior valor da função objectivo de entre todos os nós ainda não completamente explorados (a maior esperança que ainda temos de encontrar uma solução inteira melhor do que aquela que já temos).



Limites – exemplo

Considere um problema de maximização exclusivamente com variáveis inteiras. Resolvendo o problema através de "Branch-and-Bound", obtém-se, num dado estágio, a seguinte árvore:

Slide 51



Limites – exemplo

Slide 52

- Qual é, nesta altura, o melhor **limite superior** sobre a solução inteira óptima?

  O melhor **limite superior** sobre a solução inteira óptima no momento de resolução retratado na árvore é dado pela solução do problema $PL5$ e é igual a 75. Qualquer solução inteira que surja a partir da exploração desse nó terá um valor da função objectivo ≤ 75

- Qual é, nesta altura, o melhor **limite inferior** sobre a solução inteira óptima?

  Os limites inferiores são dados por valores de soluções admissíveis (variáveis já inteiras) que ainda se desconhece se são ou não óptimas. Neste caso temos já 2 soluções inteiras, para PL6 e para PL4. A que tem o maior valor da função objectivo fornece o melhor **limite inferior**, 70 neste caso.

Limites – exemplo

Slide 53

- Indique que **nós já** foram **explorados** e explique porquê.

  Já foram explorados os nós $PL1$, $PL2$, $PL3$ e $PL7$ porque já têm ramos.

  Os nós $PL4$ e $PL6$ já foram explorados porque deram origem a soluções inteiras.

  O nó $PL8$ já foi explorado porque corresponde a um problema sem solução admissível.

  O nó $PL9$ já foi explorado porque pode ser cortado. Corresponde a um problema com solução óptima não inteira e com um valor para a função objectivo inferior ao valor da solução inteira do problema $PL6$.

- Indique os nós que **ainda não** foram **explorados** e explique porquê.

  O nó $PL5$ ainda não foi explorado, dado que corresponde a um problema com solução óptima não inteira, mas com um valor para a função objectivo superior ao valor da melhor solução inteira obtida até ao momento (problema $PL6$).

Limites – exemplo

Slide 54

- Já foi atingida a solução óptima do problema inteiro? Porquê?

  Não se sabe ainda se já foi obtida a solução óptima do problema inteiro, porque ainda há nós por explorar ($PL5$). Só quando os melhores limites inferiores e superiores coincidirem é que se pode afirmar que a melhor solução inteira obtida é a óptima.

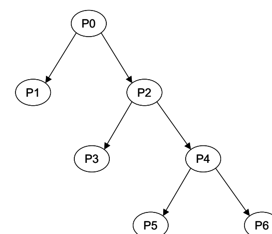
- Qual o valor máximo do erro absoluto sobre a solução óptima inteira, se o algoritmo for terminado neste ponto?

  O valor máximo do erro absoluto sobre a solução óptima inteira, se o algoritmo for terminado neste ponto será 5, isto é, a diferença entre os melhores limites superior e inferior.

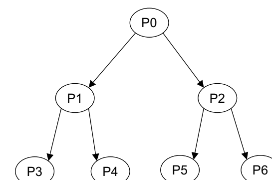
Questões em aberto

Slide 55

- **Estratégias de ramificação** – Dado um conjunto de nós ainda não explorados, como escolher o nó seguinte a explorar?
  - Pesquisa em profundidade: Selecção do nó que está mais fundo na árvore (último nó a ser gerado).



  - Pesquisa em largura: Selecção do nó que está mais acima na árvore (o nó mais antigo ainda não explorado).



  - O nó mais promissor: Selecção do nó que tem melhor valor de função objectivo (aquele que potencialmente nos pode levar à melhor solução inteira).

Questões em aberto

Slide 56

- **Selecção da variável a ramificar** – Seleccionado o nó a explorar, que variável escolher para ramificação, de entre todas as variáveis que não tomam valores inteiros?

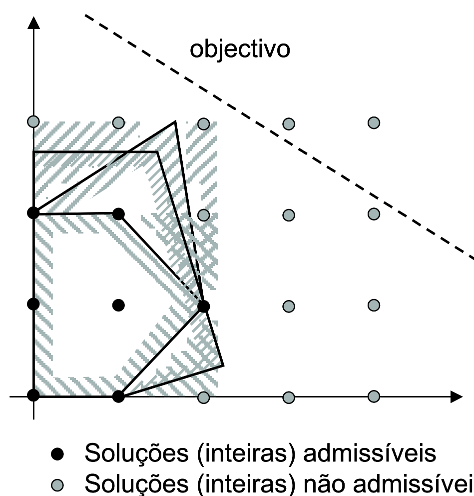
  Algumas estratégias foram apresentadas na literatura, mas o seu desempenho revelou-se extremamente dependente do problema concreto a que são aplicadas.

  Estratégias dependentes da aplicação e do significado físico das variáveis.

Formulações em programação inteira

Slide 57

Um mesmo problema de programação inteira pode ter diferentes formulações, isto é, diferentes conjuntos de restrições que definem o mesmo conjunto de soluções inteiras.

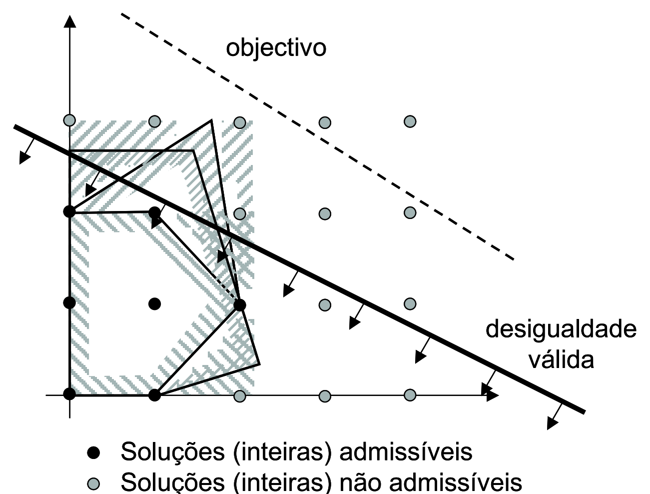


O ideal seria ter o invólucro convexo das soluções inteiras admissíveis.

Desigualdades válidas

Slide 58

Para melhorar a qualidade das formulações podem-se introduzir **desigualdades válidas**: restrições que não fazem parte do problema original mas que, não cortando soluções inteiras admissíveis, cortam à região admissível, melhorando assim o desempenho da pesquisa da solução óptima inteira.



Exemplo de uma desigualdade válida

Slide 59

Considere a seguinte instância de um problema mochila 0-1:

$$\begin{array}{lrcl} \max & 12x_1 + 14x_2 + 15x_3 + 24x_4 + 24x_5 + 17x_6 & & \\ \text{suj. a} & 15x_1 + 14x_2 + 14x_3 + 18x_4 + 17x_5 + 12x_6 & \leq & 60 \\ & x_i \in \{0,1\} & & \end{array}$$

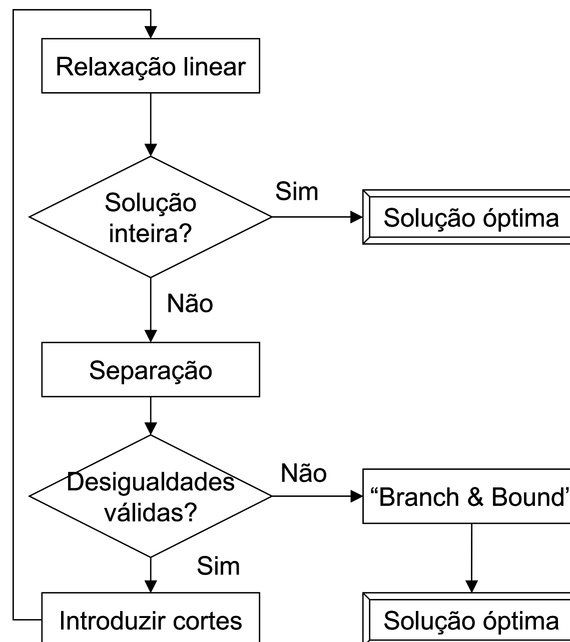
Observando que, por exemplo, os items 1, 2, 4 e 5 nunca poderão fazer parte simultaneamente de nenhuma solução inteira admissível, poder-se-ia introduzir a seguinte desigualdade válida:

$$x_1 + x_2 + x_4 + x_5 \leq 3$$

Algoritmo de planos de corte

Como proceder para aplicar sistematicamente desigualdades válidas na resolução de um problema?

Slide 60



Algoritmo de planos de corte

Slide 61

- **Relaxação linear** – resolução do problema sem as restrições de integralidade.
- **Separação** – fase em que se geram as desigualdades válidas. Utilizando uma (ou mais) propriedade observada (e descoberta) anteriormente aplica-se à instância concreta e/ou à solução actual. Caso a actual solução fraccionária viole esta nova desigualdade, então ela é válida (no sentido de que vai cortar à região admissível do problema relaxado).
- **Introduzir cortes** – introdução das desigualdades válidas, encontradas na fase de separação, no modelo.

Exemplo – um problema (simples) de planeamento da produção

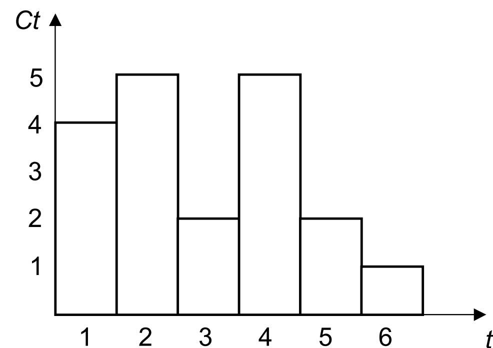
Slide 62

Dados – 6 períodos e 8 encomendas.

Objectivo – produzir o mais próximo possível da data de entrega.

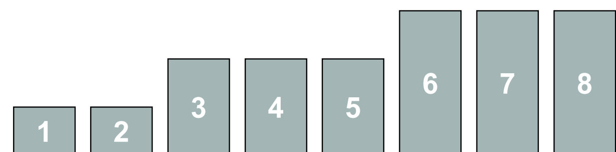
t	1	2	3	4	5	6
C_t	4	5	2	5	2	1

Capacidade disponível C_t em cada período t



e	1	2	3	4	5	6	7	8
q_e	1	1	2	2	2	3	3	3
d_e	2	1	3	1	2	5	1	3

Encomendas e, com quantidades q_e e datas de entrega d_e



Exemplo – continuação

Slide 63

Variáveis de decisão: $x_{et} \in \{0, 1\}$ que valem 1 se a encomenda e é produzida no período t.

Restrições:

- Cada encomenda tem que ser produzida uma e uma só vez: $\sum_t x_{et} = 1$
- As capacidades dos períodos têm que ser respeitadas: $\forall_t \sum_e q_e \times x_{et} \leq C_t$

Observação: há encomendas que, dadas as respectivas quantidades e as capacidades dos períodos, nunca poderão ser produzidas em simultâneo.

Exemplo – continuação

Slide 64

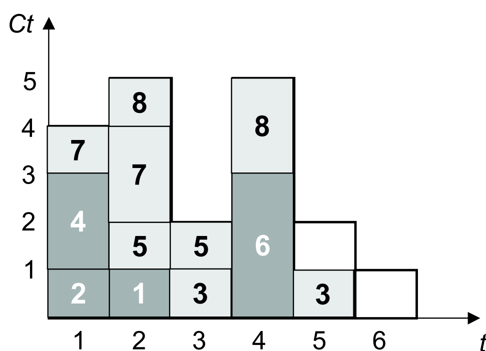
Regras para a geração de desigualdades válidas:

1. No período 1 (capacidade 4) não se podem produzir simultaneamente duas encomendas com quantidades 2 e 3, ou 3 e 3.
2. Nos períodos 2 e 4 (capacidade 5) não se podem produzir simultaneamente duas encomendas com quantidades 3.
3. Nos períodos 3 e 5 (capacidade 2) não se podem produzir simultaneamente duas encomendas com quantidades 2, duas encomendas com quantidades 1 e 2, nem qualquer encomenda com quantidade 3.
4. No período 6 (capacidade 1) apenas se podem produzir encomendas com quantidade 1.

Exemplo – conclusão

Slide 65

Solução da relaxação linear do exemplo:



Esta solução viola uma desigualdade do tipo 1 e uma desigualdade do tipo 2.

São então desigualdades válidas:

$$\begin{aligned} x_{41} + x_{71} &\leq 1 \\ x_{64} + x_{84} &\leq 1 \end{aligned}$$

Bibliografia

Slide 66

- Alves, José Carlos (1989). *Provas de Aptidão Científica e Capacidade Pedagógica.* FEUP.
- Carravilla, Maria Antónia (1996). *Modelos e Algoritmos para o Planeamento Hierárquico da Produção – Aplicações a um Caso de Estudo*, Tese de Doutoramento, FEUP.
- Goldbarg, Marco Cesar e Luna, Henrique Pacca (2000). *Otimização Combinatória e Programação Linear*, Editora CAMPUS.
- Nemhauser, George L. e Wolsey, Laurence A. (1988). *Integer and Combinatorial Optimization* John Wiley & Sons, Inc..

Algoritmos para Resolução Aproximada de Problemas de Optimização Combinatória

Slide 67

Técnicas aproximadas para a resolução de problemas de Optimização Combinatória

Métodos Heurísticos

Têm como objectivo obter muito boas soluções de uma forma eficiente. Não obtêm a solução óptima, ou pelo menos não são capazes de garantir que a solução boa que obtêm é de facto a óptima.

Slide 68 **Características dos algoritmos heurísticos**

- Tempos de execução "curtos"
- Facilidade de implementação
- Flexibilidade
- Simplicidade

Tipos de algoritmos heurísticos

Slide 69

- **Construtivos** – Constroem uma solução, passo a passo, segundo um conjunto de regras pré-estabelecido.
- **de Melhoramentos** – Partem de uma solução admissível qualquer e procuram melhorá-la através de sucessivas pequenas alterações.
- **Compostos** – Têm primeiro uma fase construtiva e depois uma fase de melhoramentos.

Estes tipos de heurísticas serão apresentados e exemplificados utilizando como caso de estudo o Problema do Caixeiro Viajante.

Algoritmos (heurísticos) construtivos

Slide 70

Constroem uma solução, passo a passo, segundo um conjunto de regras pré-estabelecido. Estas regras estão relacionadas com:

- a escolha do sub-ciclo inicial (ou o ponto inicial) – *inicialização*;
- um critério de escolha do elemento seguinte a juntar à solução – *selecção*;
- a selecção da posição onde esse novo elemento será inserido – *inserção.*

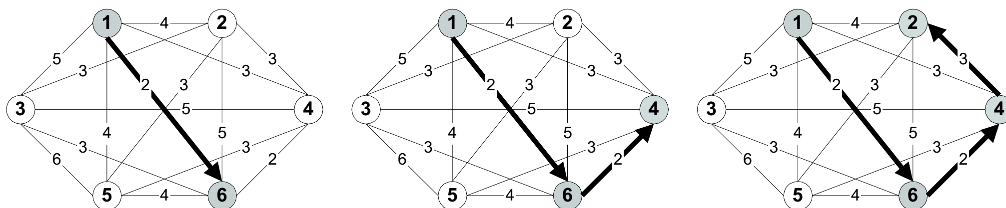
TSP – Vizinho mais próximo

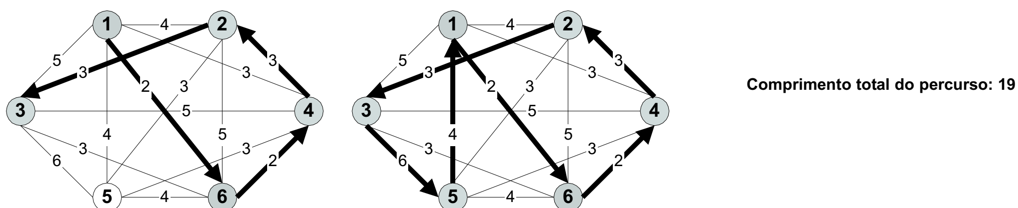
Slide 71

1. *Inicialização* – Começar com um circuito parcial constituído por uma cidade i sozinha, escolhida arbitrariamente;
2. *Selecção* – Seja $(1, \ldots, k)$ o percurso parcial actual $(k < n)$. Encontrar a cidade $k + 1$, que ainda não faz parte do circuito e que está mais próxima de k.
3. *Inserção* – Inserir $k + 1$ no fim do circuito parcial.
4. Se todas as cidades estão inseridas, PARAR, senão voltar a 2.

Vizinho mais próximo – exemplo

Slide 72





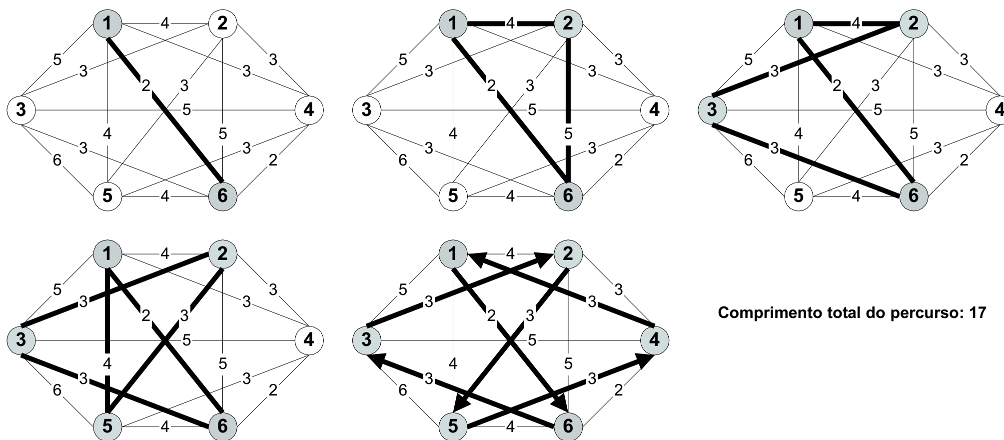
TSP – Inserção mais próxima de cidade arbitrária

Slide 73

1. *Inicialização* – Começar com um circuito parcial constituído por uma cidade i sozinha, escolhida arbitrariamente;
   Encontrar a cidade j tal que c_{ij} (distância de i a j) é mínima e formar o circuito parcial (i, j).
2. *Selecção* – Dado um circuito parcial, seleccionar arbitrariamente uma cidade k ainda não pertencente ao circuito parcial.
3. *Inserção* – Encontrar a aresta $\{i, j\}$ no circuito parcial que minimiza $c_{ik} + c_{kj} - cij$. Inserir k entre i e j.
4. Se todas as cidades estão inseridas, PARAR, senão voltar a 2.

Inserção mais próxima de cidade arbitrária – exemplo

Slide 74



TSP – Inserção mais próxima

Slide 75

1. *Inicialização* – Começar com um circuito parcial constituído por uma cidade i sozinha, escolhida arbitrariamente;
2. *Selecção* – Encontrar as cidades k e j (j pertencendo ao circuito parcial e k não pertencendo) tal que c_{kj} é minimizado.
3. *Inserção* – Encontrar a aresta $\{i, j\}$ no circuito parcial que minimiza $c_{ik} + c_{kj} - cij$. Inserir k entre i e j.
4. Se todas as cidades estão inseridas, PARAR, senão voltar a 2.

Esta heurística tem a variante "Inserção mais distante" que substitui o passo de selecção por:

2. *Selecção* – Encontrar as cidades k e j (j pertencendo ao circuito parcial e k não pertencendo) tal que c_{kj} é maximizado.

TSP – Inserção mais barata

Slide 76

1. *Inicialização* – Começar com um circuito parcial constituído por uma cidade i sozinha, escolhida arbitrariamente;
2. *Selecção* – Encontrar as cidades k, i e j (i e j formando uma aresta do circuito parcial e k não pertencendo a esse circuito) tal que $c_{ik} + c_{kj} - cij$ é minimizado.
3. *Inserção* – Inserir k entre i e j.
4. Se todas as cidades estão inseridas, PARAR, senão voltar a 2.

TSP – Invólucro convexo[a]

Slide 77

1. *Inicialização* – Começar com um circuito parcial constituído pelo invólucro convexo de todas as cidades;
2. *Inserção* – Para cada cidade k não inserida no circuito parcial, encontrar a aresta $\{i, j\}$ do circuito parcial que minimiza $c_{ik} + c_{kj} - cij$.
3. *Selecção* – De entre todos os trios $\{i, j, k\}$ formados e avaliados no passo 2, determinar o trio $\{i^\star, j^\star, k^\star\}$ tal que $\frac{c_{i^\star k^\star} + c_{k^\star j^\star}}{c_{i^\star j^\star}}$ é mínimo.
4. Inserir $k^\star$ entre $i^\star$ e $j^\star$.
5. Se todas as cidades estão inseridas, PARAR, senão voltar a 2.

[a] Invólucro convexo do conjunto A – forma convexa que contém no seu interior ou fronteira todos os elementos do conjunto A

TSP – Fusão mais próxima

Slide 78

1. *Inicialização* – Começar com n circuitos parciais constituídos, cada um, por uma cidade i sozinha;
2. *Selecção* – Encontrar as cidades i e k (i pertencendo a um circuito parcial C e k pertencendo a um outro circuito C') tal que c_{ik} é minimizado.
3. *Inserção* – Sejam i, j, k e l cidades tais que a aresta $\{i, j\} \in C$, $\{k, l\} \in C'$ e $c_{ik} + c_{jl} - c_{ij} - c_{kl}$ é minimizado.
   Inserir $\{i, k\}\}$ e $\{j, l\}\}$ e retirar $\{i, j\}\}$ e $\{k, l\}\}$.
4. Se existir um único circuito, PARAR, senão voltar a 2.

O problema da árvore suporte de comprimento mínimo

- Definições (para grafos não orientados):
  - uma árvore é um grafo conexo que não contém ciclos;
  - um grafo diz-se conexo se existir uma cadeia (sequência de ramos) ligando qualquer par de nós entre si.

Slide 79

- Problema:

  Determinar a árvore de comprimento total mínimo que suporte todos os nós da rede (i.e. que ligue todos os nós da rede) ("minimal spanning tree").

- Aplicações:
  - redes de comunicações;
  - redes de distribuição de energia.

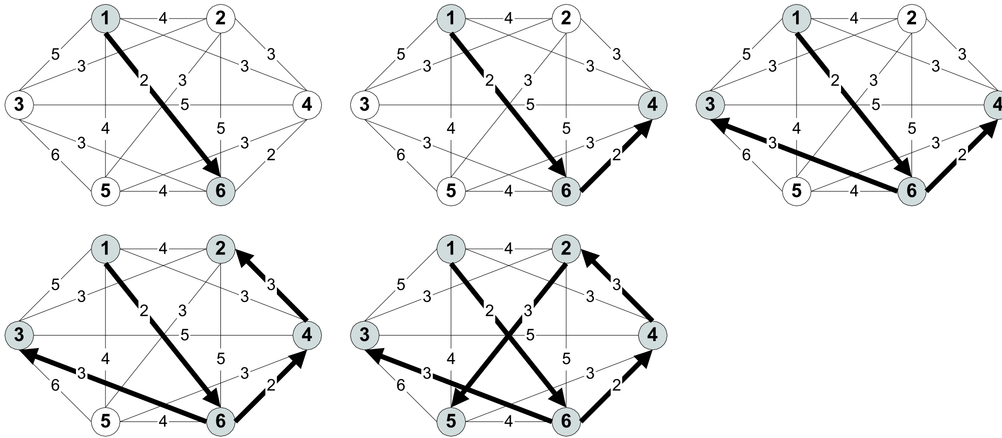
Algoritmo de Prim (guloso – "Greedy Procedure")

1. Seleccionar um nó arbitrariamente, e ligá-lo ao nó mais próximo;
2. Identificar o nó ainda isolado que esteja mais próximo de um nó já ligado, e ligar estes dois nós;
3. Se todos os nós estiverem ligados entre si, PARAR, senão voltar a 2.

Slide 80

Algoritmo guloso – Exemplo

Slide 81



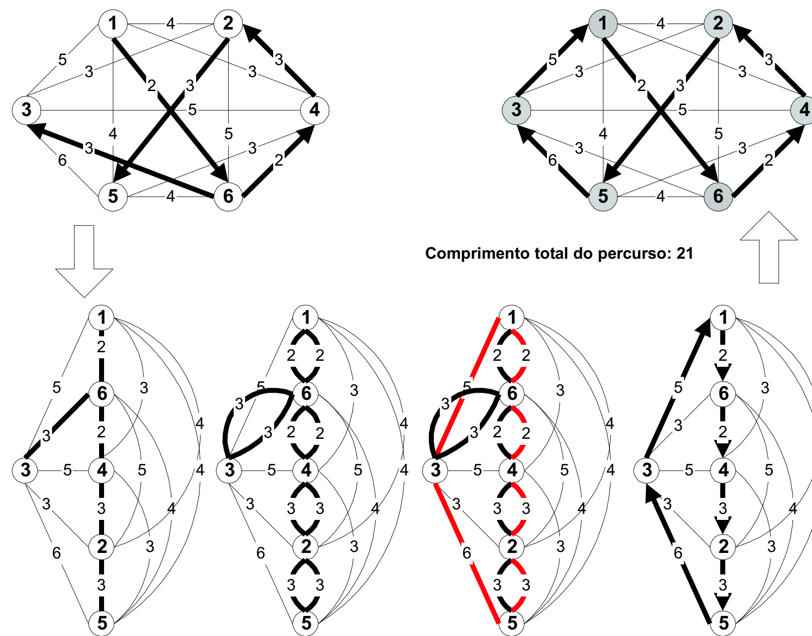
TSP – Árvore suporte de comprimento mínimo

Slide 82

1. Determinar a árvore suporte de comprimento mínimo para o conjunto de todas as cidades.
2. Fazer uma visita em profundidade à árvore.
3. Introduzir atalhos (substituição de sequências de 2 ou mais ramos por um único ramo) no percurso gerado na visita em profundidade, de forma a obter um circuito.

Árvore suporte de comprimento mínimo – exemplo

Slide 83



Algoritmos (heurísticos) de melhoramentos

Slide 84

Partem de uma **solução admissível qualquer** e procuram melhorá-la através de sucessivas pequenas alterações.

⟶ como obter?
↓

(1) aleatoriamente;

(2) heurística construtiva.

Neste segundo caso estamos de facto a desenhar uma heurística composta, em que o algoritmo de melhoramentos corresponde à segunda fase desta heurística composta.

TSP – Algoritmo (heurístico) *r-opt*[a]

Slide 85

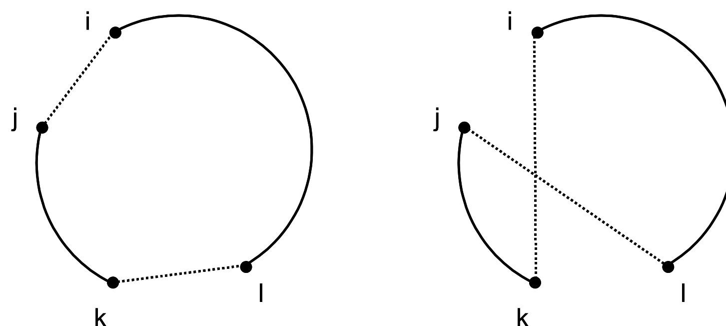
1. Obter um circuito inicial *completo* admissível $\rightarrow C^0$.
   Fazer o circuito corrente $C^k = C^0$.
2. Remover r arestas do circuito corrente C^k, tornando-o incompleto $\rightarrow C_i^k$.
3. Construir todas as soluções admissíveis (circuitos completos) que contenham C_i^k (o circuito incompleto).
4. Escolher o melhor destes circuitos $\rightarrow C^\star$.
5. Se **comprimento**$(C^\star)$ < **comprimento**(C^k) então $C^k = C^\star$ e voltar para 2. Senão PARAR.

[a] ou 2-exchange

Algoritmo 2-opt

Slide 86

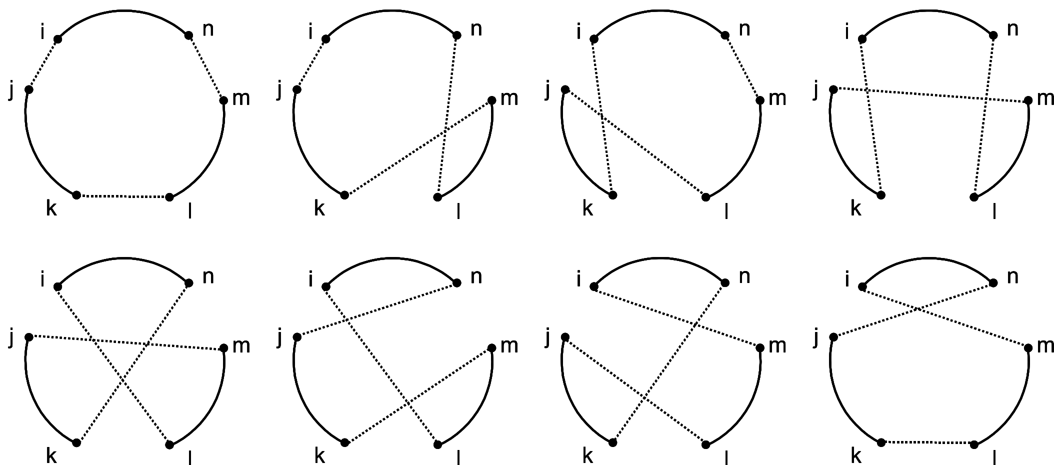
Num algoritmo 2-opt, ao retirar 2 ramos, apenas há uma solução admissível alternativa:



Algoritmo 3-opt

Slide 87

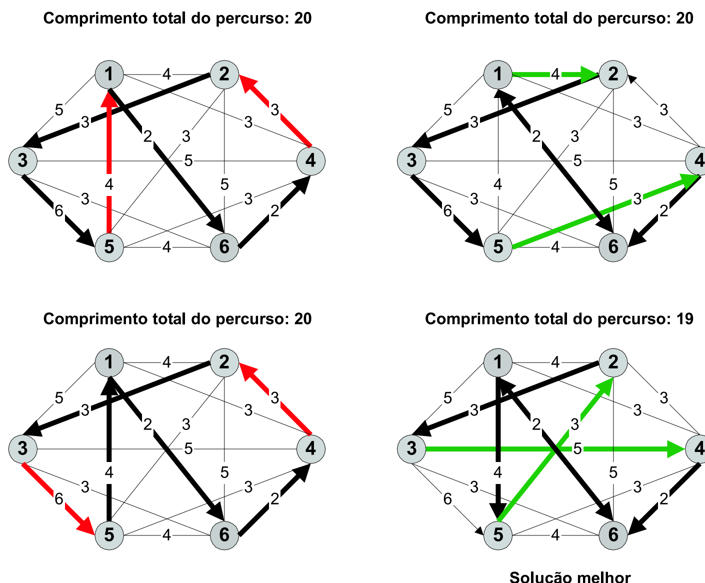
Num algoritmo 3-opt, ao retirar-se 3 ramos, há $2^3 - 1$ soluções admissíveis alternativas:



Algoritmo 2-opt – exemplo

Slide 88

Dois exemplos de trocas de 2 ramos, um conduzindo a uma solução de igual valor, e outra conduzindo a uma solução de menor valor.



O algoritmo prosseguiria a partir desta solução melhor, até que algum critério de paragem fosse atingido (e.g. número máximo de trocas, número de trocas sem melhoria, etc.)

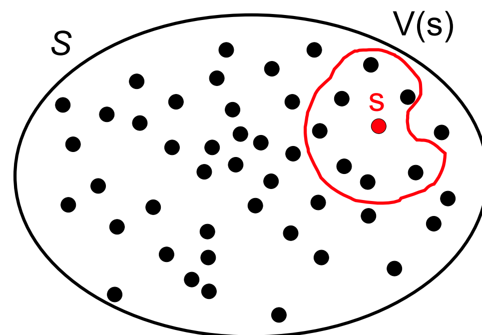
Pesquisa local e vizinhanças

A pesquisa local baseia-se no método de optimização mais antigo: a tentativa e erro. Mas de uma forma sistemática...

Para sistematizar a pesquisa constrói-se uma **estrutura de vizinhança**.

Slide 89

A vizinhança de uma solução é um conjunto de soluções admissíveis, de algum modo "parecidas" com a solução em causa, isto é, com elementos semelhantes e valores de função objectivo não muito diferentes.



Exemplo: No TSP pode-se definir como vizinhança de um percurso todos os percursos que se obtêm a partir deste através de uma iteração 2-opt.

Pesquisa local e vizinhanças – continuação

Slide 90

Algoritmo genérico de pesquisa local:

1. Gerar uma solução inicial $\rightarrow s_0$.
2. Solução corrente $s_i = s_0$.
3. Considerar um $s_j \in V(s_i)$.
4. Se $f(s_j) < f(s_i)$, então $s_i = s_j$.
5. Senão, $V(s_i) = V(s_i) - s_j$.
6. Se $V(s_i) \neq \emptyset$, ir para 3.
7. Senão, FIM.
   Solução óptima local $= s_i$.

Chama-se movimento a cada aceitação de uma nova solução como solução corrente (também designada "centro da vizinhança") – passo 4.

Algoritmo de pesquisa local para o TSP baseado em movimentos 2-opt

Slide 91

1. Construir um circuito inicial.
2. Seleccionar aleatoriamente um ramo desse circuito.
3. Fazer um movimento 2-opt com todos os outros ramos do circuito e seleccionar o melhor dos circuitos assim obtidos.
4. Se for melhor do que o circuito actual, torná-lo o circuito actual e ir para 2.
5. Senão, PARAR. Foi atingido o óptimo local.

Diferentes estruturas de vizinhança dão origem a diferentes algoritmos de pesquisa local.

Pesquisa local e vizinhanças – conclusão

Slide 92

Construir uma boa estrutura de vizinhança para um problema de optimização combinatória e determinar um método para a sua pesquisa: uma ciência e uma arte → trabalho de investigação nobre.

Bibliografia

Slide 93

- Goldbarg, Marco Cesar e Luna, Henrique Pacca (2000). *Otimização Combinatória e Programação Linear*, Editora CAMPUS.
- Golden, B.L. and Stewart, W.R. (1985). *Empirical analysis of heuristics* in The Traveling Salesman Problem, John Wiley & Sons, Inc..
- Sousa, Jorge Pinho (1991). *Apontamentos de Optimização Combinatória.*