Programação Inteira

Conteúdos da Seção

- Programação Inteira
  - Problema Relaxado
  - Solução Gráfica
  - Solução por Enumeração
  - Algoritmo de *Branch-And-Bound*
  - Solução Excel
  - Solução no Lindo
- Caso LCL Tecnologia S.A.
- Variáveis Binárias e Condições Lógicas
- Caso LCL Equipamentos S.A.

Programação Inteira

- São problemas de programação matemática em que a função-objetivo, bem como as restrições, são lineares, porém uma ou mais variáveis de decisão podem apenas assumir valores inteiros
- É uma das ferramentas comumente usada para resolver problemas de otimização combinatória.
- Tipos básicos:
  - **Programação Inteira Total -** onde todas as variáveis de decisão são do tipo inteiro.
  - **Programação Inteira Mista -** onde apenas uma parte das variáveis são do tipo inteiro, enquanto outras são do tipo real

Solução óbvia e ERRADA

- Resolver o problema como se fosse um problema de programação linear e arredondar os valores ótimos encontrados para cada uma das variáveis de decisão inteiras.
- Para problemas de grande porte, isto geralmente gerará uma solução aceitável (próxima do ótimo real) sem a violação de nenhuma das restrições.
- Para problemas menores, esse tipo de procedimento poderá nos levar a soluções inviáveis ou não ótimas.

Problema Relaxado

- A todo problema de programação inteira está associado um **Problema Relaxado:**
  - Mesma função objetivo e as mesmas restrições, com exceção da condição de variáveis inteiras.

$$Max\ 18x_1 + 6x_2$$

$$s.t. \quad x_1 + x_2 \geq 5 \quad (1)$$

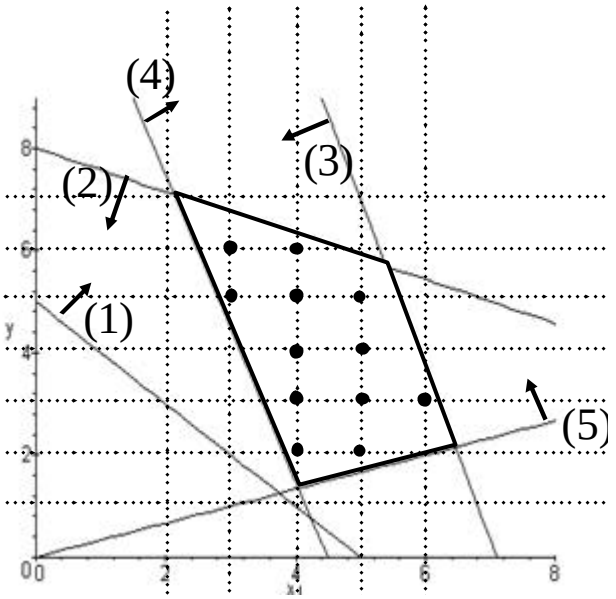
$$42.8x_1 + 100x_2 \leq 800 \quad (2)$$

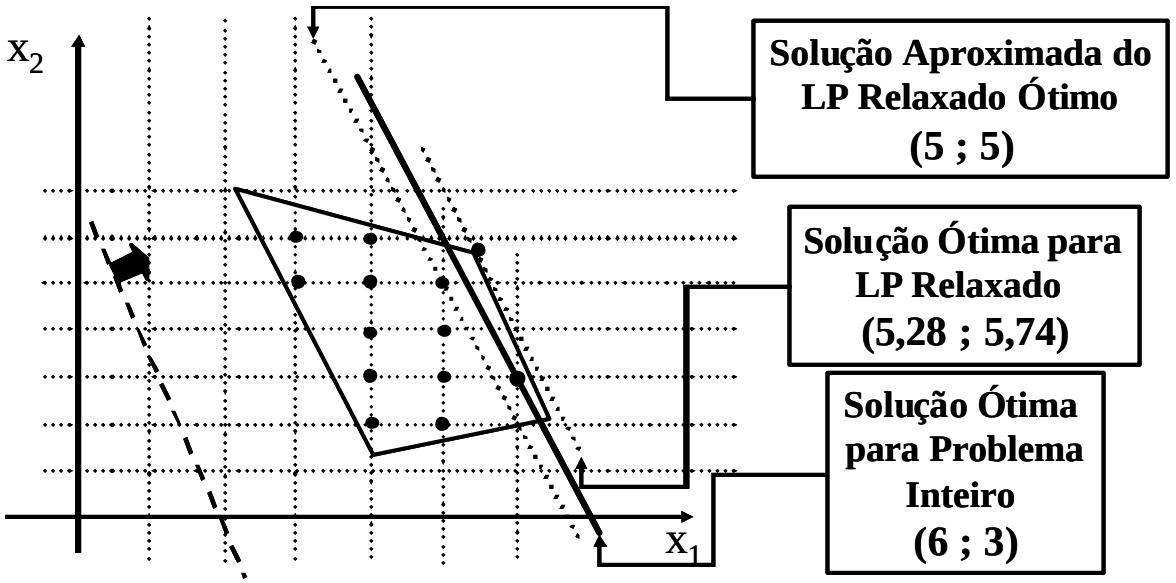
$$20x_1 + 6x_2 \leq 142 \quad (3)$$

$$30x_1 + 10x_2 \geq 135 \quad (4)$$

$$x_1 - 3x_2 \leq 0 \quad (5)$$

$$x_1, x_2 \geq 0 \quad \text{e} \quad \text{inteiros}$$





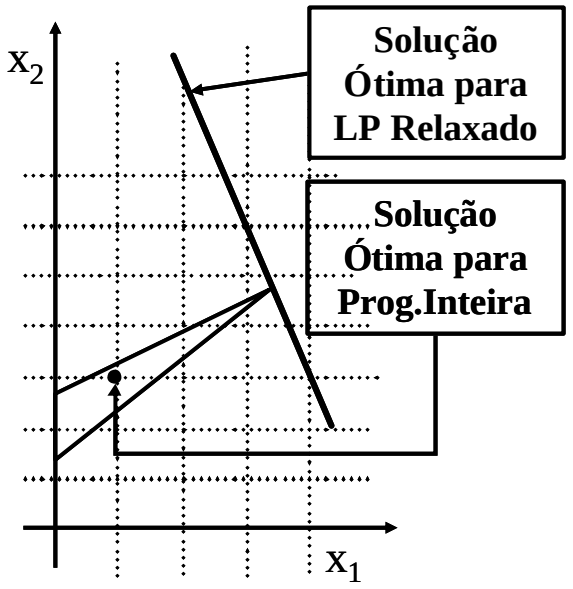
Limitantes

- Em um problema de **MAXIMIZAÇÃO,** o valor ótimo da função objetivo, do **Problema Relaxado,** sempre representa um limite superior ao respectivo **Problema Inteiro**.

- Em um problema de **MINIMIZAÇÃO,** o valor ótimo da função objetivo, do **Problema Relaxado,** sempre representa um limite inferior ao respectivo **Problema Inteiro**.

- Nenhum ponto inteiro vizinho ao ponto ótimo do problema relaxado é necessariamente viável.

- Mesmo que um dos vizinhos seja viável.
  - Não é necessariamente o ponto ótimo inteiro.
  - Não é obrigatoriamente uma solução aceitável.



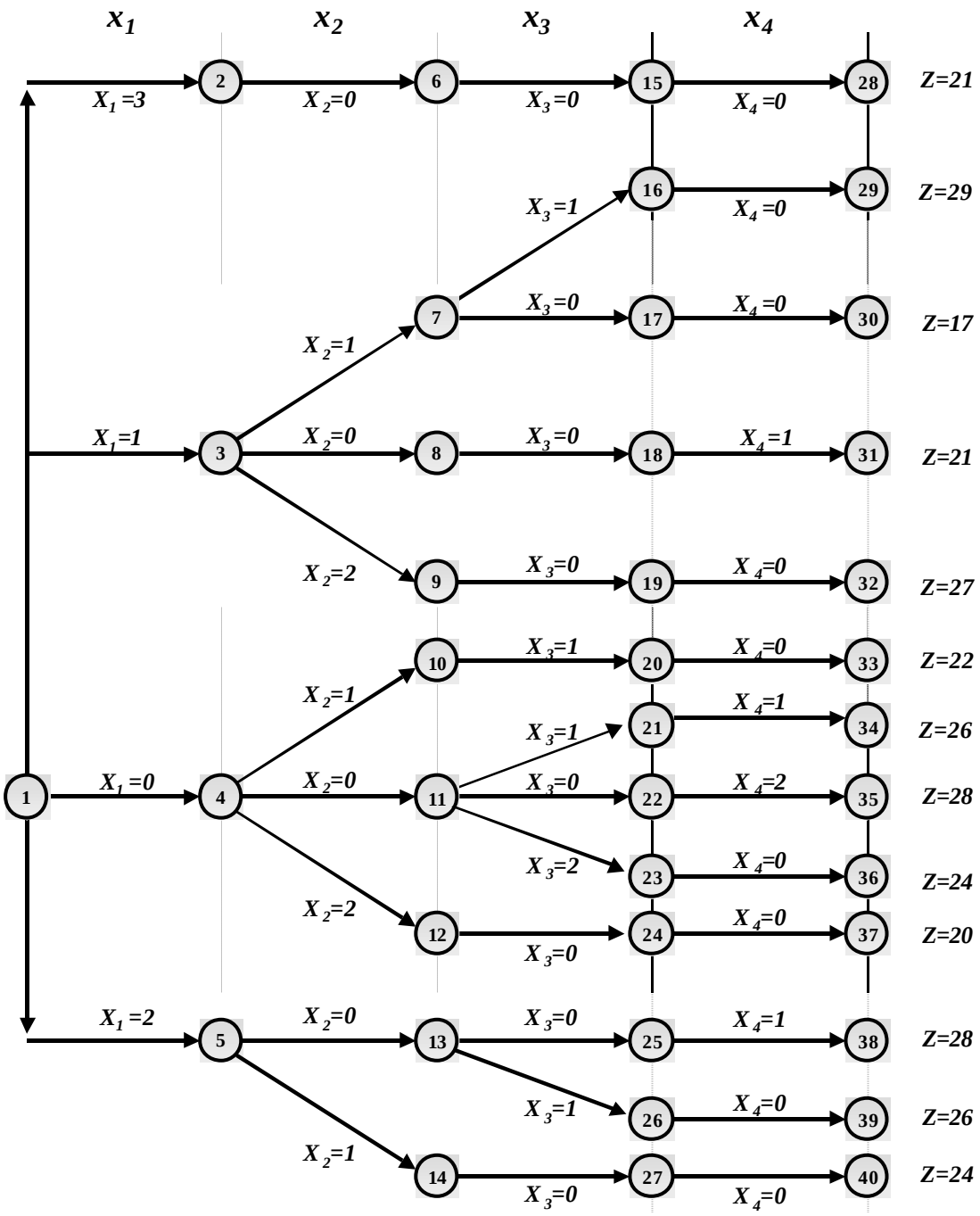
Solução por Enumeração

- Uma idéia que pode resultar em uma solução para um problema de programação inteira é a de se enumerar todas as possíveis soluções.
- De forma exaustiva, todos os valores possíveis para a função objetivo são calculados e é escolhido aquele que apresente o maior ou o menor valor (maximização ou minimização).

Pontos Examinados	*Coordenadas* (x_1, x_2)	*Valor da função* $z = x_1 + 3x_2$
A	*(40,10)*	*70*
B	*(40,30)*	*130*
C	*(20,60)*	*200*
D	*(0,60)*	*180*
E	*(0,20)*	*60*
F	*(10,10)*	*40*

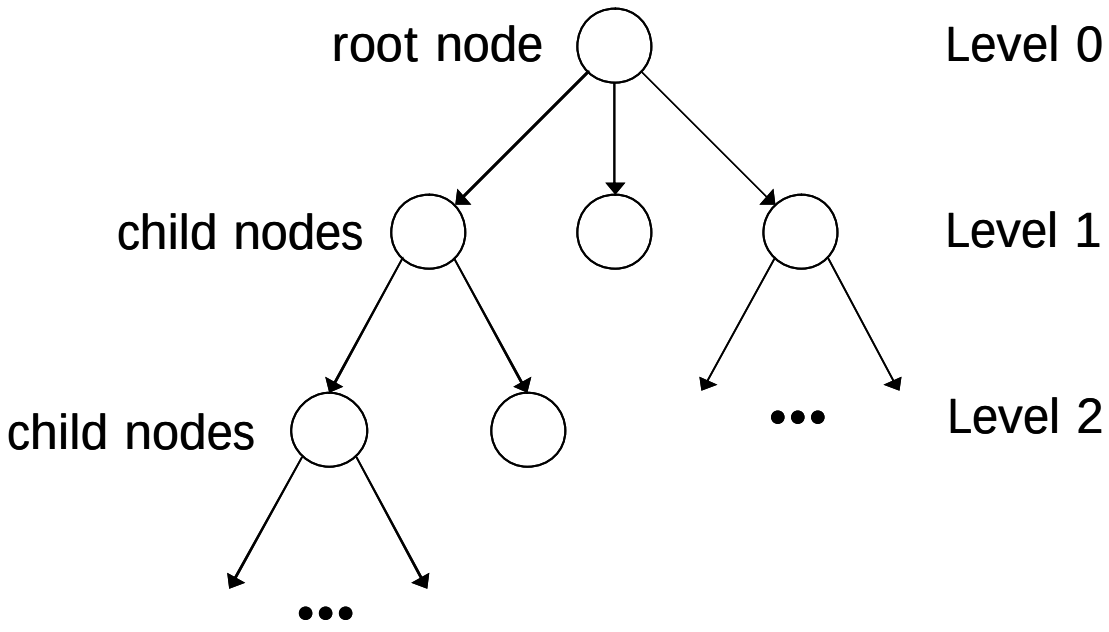
- Aplicável a problemas pequenos.
- O número de **combinações** possíveis de soluções cresce de forma exponencial.
- **Problemas intratáveis:** Ex.: Um ILP com 100 variáveis de decisão do tipo binárias (assumem 0/1) terá até 2^{100} soluções viáveis, isto é, 1,27 x 10^{30} soluções possíveis.
  - **Exponencia**l: complexidade é $O(c^n)$, c>1.
    - **problema do caixeiro viajante:** (O(n!)).
  - Mesmo problemas de tamanho pequeno a moderado não podem ser resolvidos por algoritmos não-polinomiais.

Solução por Enumeração



Algoritmo de *Branch-And-Bound*

- Algoritmo bastante popular em P.I.
- É uma metodologia geral para solução de ILP e MILP, e diversas variantes existem para tratar diversos tipos de problemas específicos.
- A idéia geral é a de se particionar o conjunto de soluções viáveis em subconjuntos sem interseções entre si;
- Calcula-se os limites superior e inferior para cada subconjunto;
- Elimina-se certos subconjuntos de acordo com regras pré-estabelecidas;

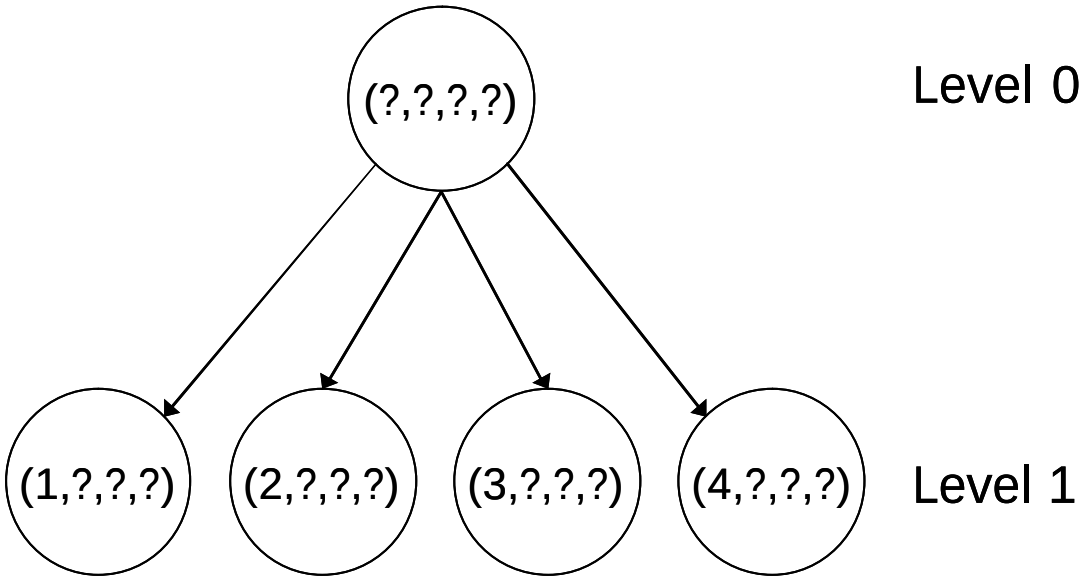


Cada nó é uma solução parcial

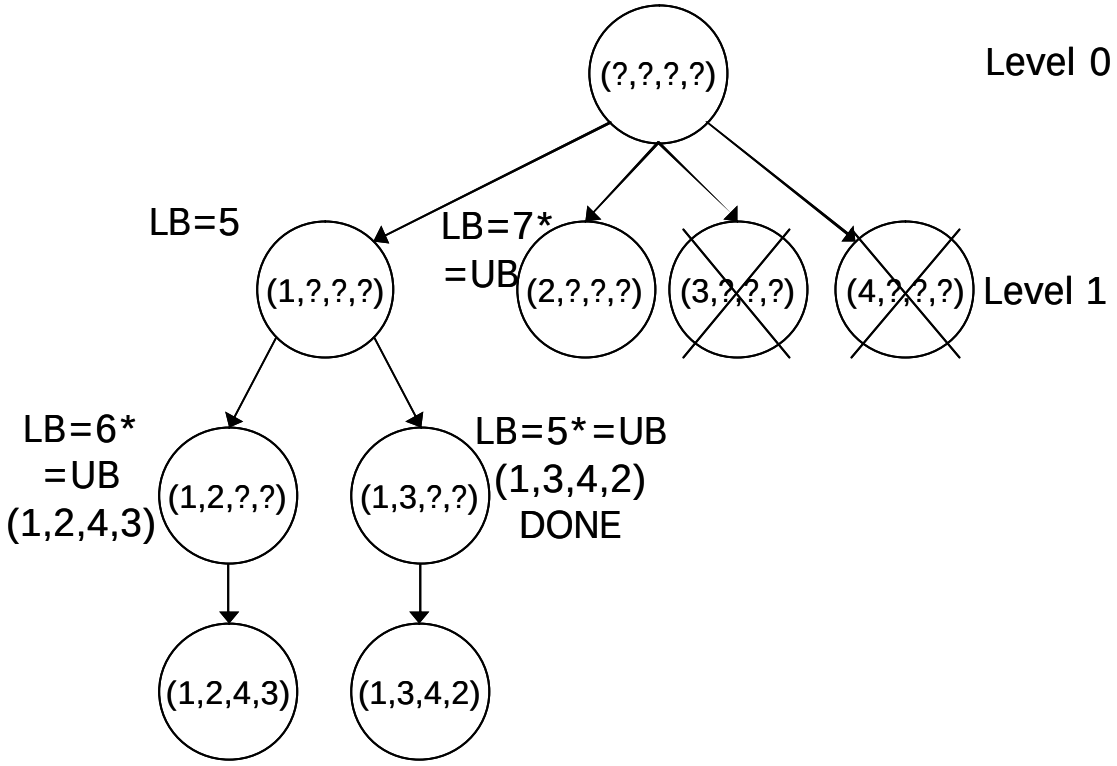
Algoritmo de *Branch-and-Bound*

- Cada nó possui limitantes inferior e superior
  - *Upper bound* (limite superior)
    - Obtido através de uma solução viável
  - *Lower bound* (limite inferior)
    - Estimado
- Eliminação de nós:
  - quando LB >= UB

- Estratégia de ramificação:
  - Como particionar o espaço de busca
- Seleção de nós: seqüência de exploração de nós
  - Profundidade: tenta obter logo uma solução
  - Largura: tenta obter a melhor solução



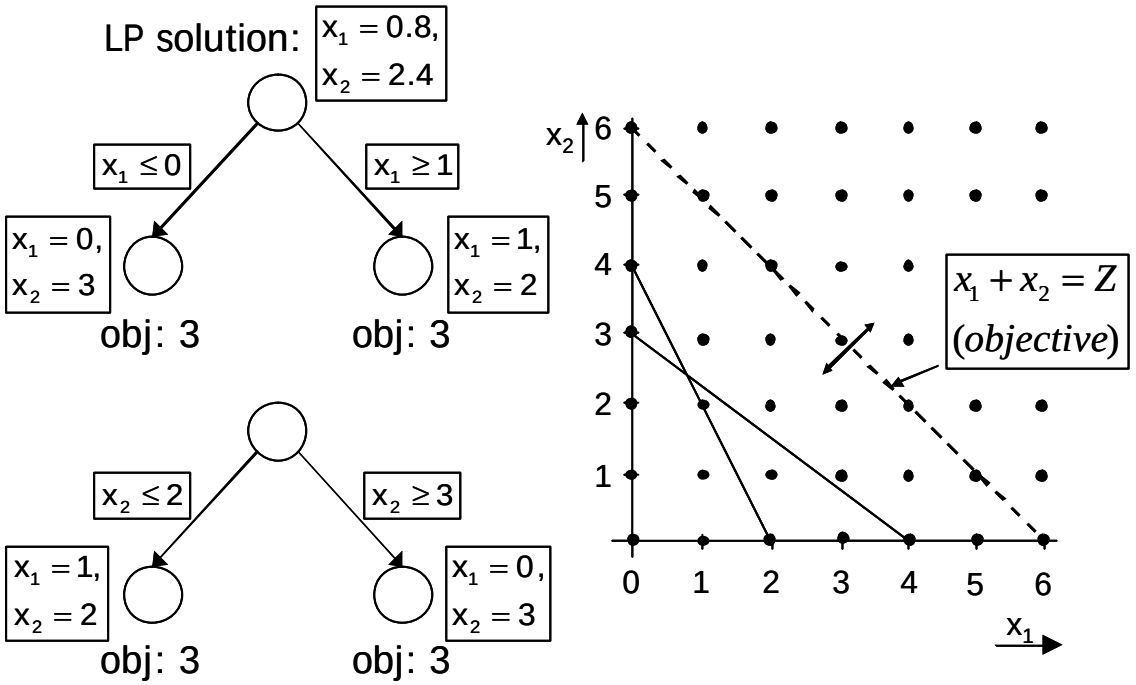
Algoritmo de *Branch-And-Bound*



- Profundidade x Largura
- Heurísticas para selecionar as melhores subárvores
- LB pode considerar heurísticas específicas
- Não determinar o LB não limita a eficiência do método
- O maior problema do BB é que ele ainda explora uma árvore onde as ramificações podem explodir exponencialmente
- Dependendo da heurística usada ou solução inicial encontrada, o algoritmo ainda pode ser muito lento para problemas grandes.

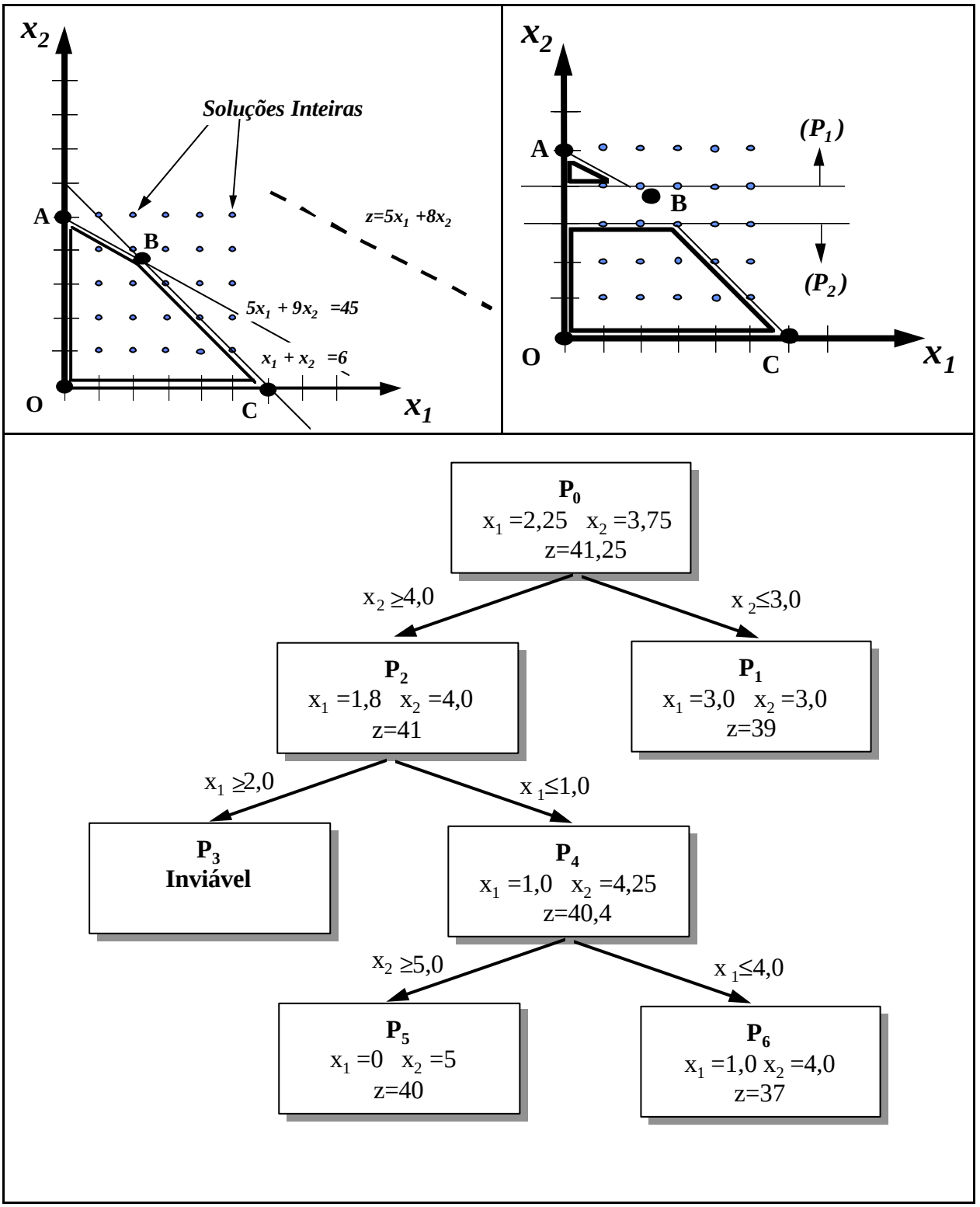
Branch-and-Bound na Programação Inteira

- Pode ser usado para resolver sucessivos problemas relaxados;
  - Diversos problemas de LP são resolvidos sucessivamente para se obter a solução de um IP.
  - Se o problema for interrompido no meio do processo, uma solução aproximada do problema inteiro pode ser gerada



Branch-and-Bound em Programação Inteira

Exemplo (1): BB reduzindo o número de estados



Branch-and-Bound em Programação Inteira

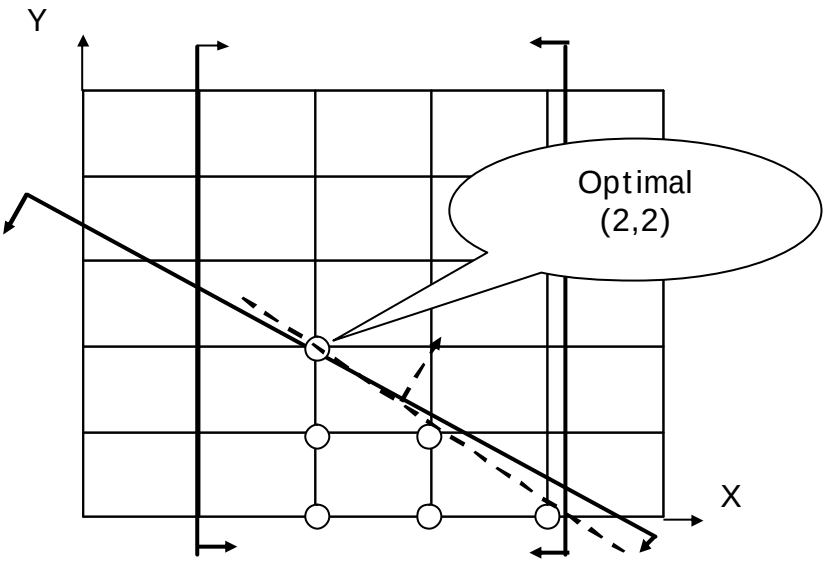
Exemplo (2)

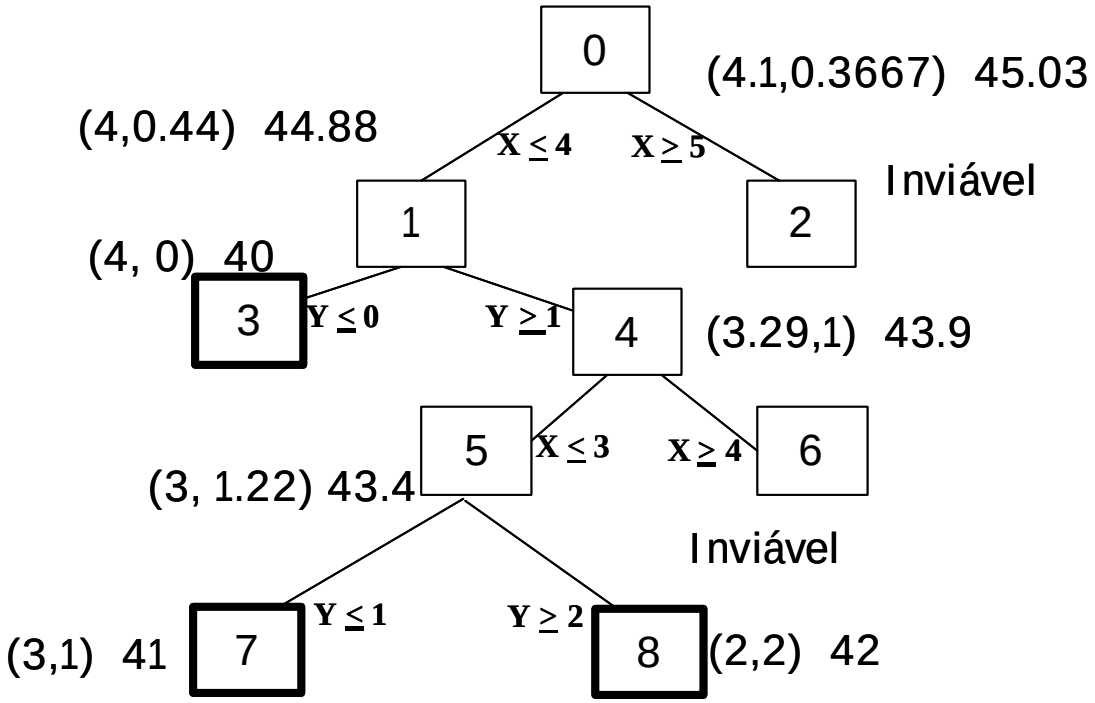
Max 10x + 11y

st 35x + 45y < 160

1 < x < 4.1

y > 0





Problema de Orçamento de Capital

- A LCL Tecnologia S/A tem que planejar seus gastos em P&D. A empresa pré-selecionou 4 projetos e deve escolher dentre esses quais deve priorizar em função de restrições orçamentárias. Os dados relevantes encontram-se na tabela abaixo.

		Capital Requerido em mil R$				
Proj.	(mil R$)	Ano 1	Ano 2	Ano 3	Ano 4	Ano 5
1	$105.99	70	15	0	20	20
2	$128.90	80	20	25	15	10
3	$136.14	90	20	0	30	20
4	$117.38	50	30	40	0	20
Capital Disponível		200	70	70	70	70

- Variáveis de Decisão

$$X_i = \left\{ \begin{array}{l} 1, \text{ se o projeto i for selecionado} \\ 0, \text{ se o projeto i não for selecionado} \end{array} \right\} \text{ i} = 1,2,3,4$$

- Função Objetivo = Maximizar o lucro

$$Max\ 105.99X_1 + 128.90X_2 + 136.14X_3 + 117.38X_4$$

- Restrições Orçamentárias

$$70X_1 + 80X_2 + 90X_3 + 50X_4 \leq 200 \quad \text{- Ano 1}$$

$$15X_1 + 20X_2 + 20X_3 + 30X_4 \leq 70 \quad \text{- Ano 2}$$

$$25X_2 + 40X_4 \leq 70 \quad \text{- Ano 3}$$

$$20X_1 + 15X_2 + 30X_3 \leq 70 \quad \text{- Ano 4}$$

$$20X_1 + 10X_2 + 20X_3 + 20X_4 \leq 70 \quad \text{- Ano 5}$$

Variáveis Binárias e Condições Lógicas

- As variáveis binárias também se prestam a selecionar alternativas que sejam condicionais.
- No exemplo anterior, imagine que não mais de um dos projetos 1, 3 e 4 pudesse ser selecionado. Deve-se então adicionar: $X_1 + X_3 + X_4 \leq 1$

- Se apenas um dos projetos 1, 2 ou 4 pudesse ser escolhido obrigatoriamente, inclui-se:

$$X_1 + X_2 + X_4 = 1$$

- Se o projeto 1 depender de uma tecnologia que deve ser desenvolvida pelo projeto 2, isto é, o projeto 1 só pode ser aprovado se e somente se o projeto 2 for aceito. Inclui-se:

$$X_1 - X_2 \leq 0 \begin{cases} X_1 = 0, X_2 = 0 \Rightarrow \text{nenhum dos projetos aceitos} \\ X_1 = 1, X_2 = 1 \Rightarrow \text{ambos os projetos aceitos} \\ X_1 = 0, X_2 = 1 \Rightarrow \text{apenas o projeto 2 foi aceito} \\ X_1 = 1, X_2 = 0 \Rightarrow \text{inviável} \end{cases}$$

- A LCL Equipamentos S.A. produz três tipos de furadeiras que necessitam de tempos diferentes na linha de montagem. Para que cada tipo de furadeira seja fabricada, um custo de preparação da fabrica é incorrido. Suponha que todas as furadeiras do mesmo tipo serão produzidas de uma só vez (apenas uma preparação por tipo). Abaixo os dados relevantes à análise do problema.

	Tipo 1	Tipo 2	Tipo 3	Total Disponível
Montagem	2h/unid	3h/unid	2,5h/unid	600h
Pintura	3h/unid	2h/unid	2,5h/unid	500h
Lucro Unitário	R$50	R$60	R$65	
Preparação	R$5.000	R$4.000	R$3.000	

- **Variáveis de Decisão**

$X_i = \text{Quantidade a ser produzida do produto i (i} = 1,2,3)$

$$Y_i = \begin{cases} 1, \text{ se } X_i > 0 \\ 0, \text{ se } X_i = 0 \end{cases} \quad \text{i} = 1,2,3$$

- **Função Objetivo**

$$Max\, 50X_1 + 60X_2 + 65X_3 - 5000Y_1 - 4000Y_2 - 3000Y_3$$

- **Restrições:**

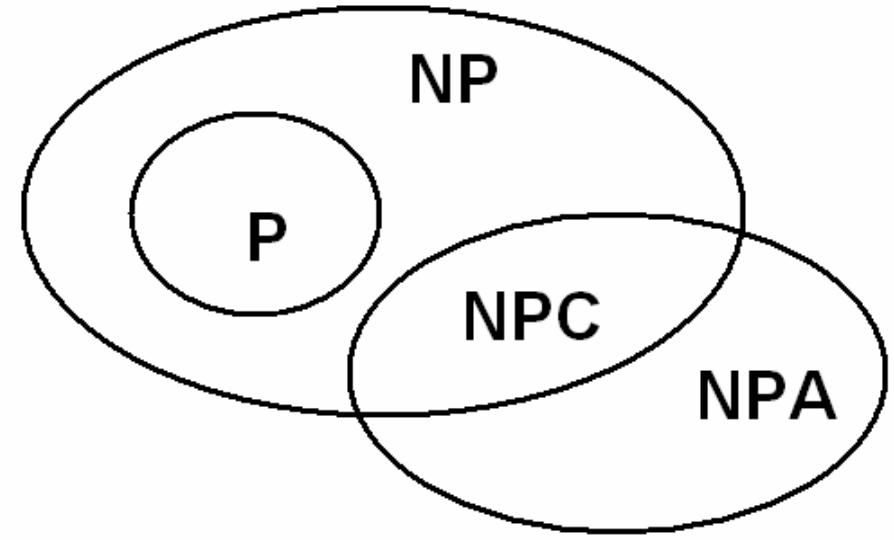
$$2X_1 + 3X_2 + 2{,}5X_3 \leq 600$$
$$3X_1 + 2X_2 + 2{,}5X_3 \leq 500$$
$$X_1 \leq 600Y_1$$
$$X_2 \leq 600Y_2$$
$$X_3 \leq 600Y_3$$

Obs : 600 é um nº que é grande o suficiente.

Teoria da computação

- Um algoritmo A é dito *polinomial* se existe algum polinômio p tal que A sempre termina (corretamente) depois de no máximo p(n) operações elementares, para quaisquer dados com tamanho total n.
  - ordenação ruim $p(n) = cn^2 + d$
  - algoritmo que conta de 1 em 1, n bits, até um limite: $q(n) = 2^n$
  - não existe nenhum polinômio p(n) que seja maior que q(n) para todo n.
- **A classe de problemas P** representa o conjunto de todas as funções ("problemas") para as quais existem (foram descobertos) algoritmos polinomiais.
- **A classe NP** representa o conjunto de todas as funções que podem ser verificadas por algoritmos polinomiais.
- **Classe de problemas NP-Completos** subconjunto de NP onde um problema pode ser “transformado” (reduzido) em outro através de um algoritmo polinomial.

- **Redução de problemas**
  - $P(x) = ax + b \leq_p Q(x) = 0x^2 + ax + b$
  - $P \leq_p Q$ => P não é mais que um fator polinomial mais difícil que Q
- Um problema NP-completo admite um algoritmo de verificação polinomial e pode ser reduzido a qualquer outro problema NP-completo em tempo polinomial.
- **P=NP:** Se um problema NP completo pudesse ser resolvido por um problema polinomial (não existe tal garantia) então todos os demais também poderão.
- **Os problemas NP-Árduos (*NP-hard*)** também podem ser reduzidos, mas não é conhecido algoritmo polinomial que verifique a solução.
- Problemas de otimização, em geral, são NP-Árduos, pois não é fácil verificar se a melhor solução foi obtida.

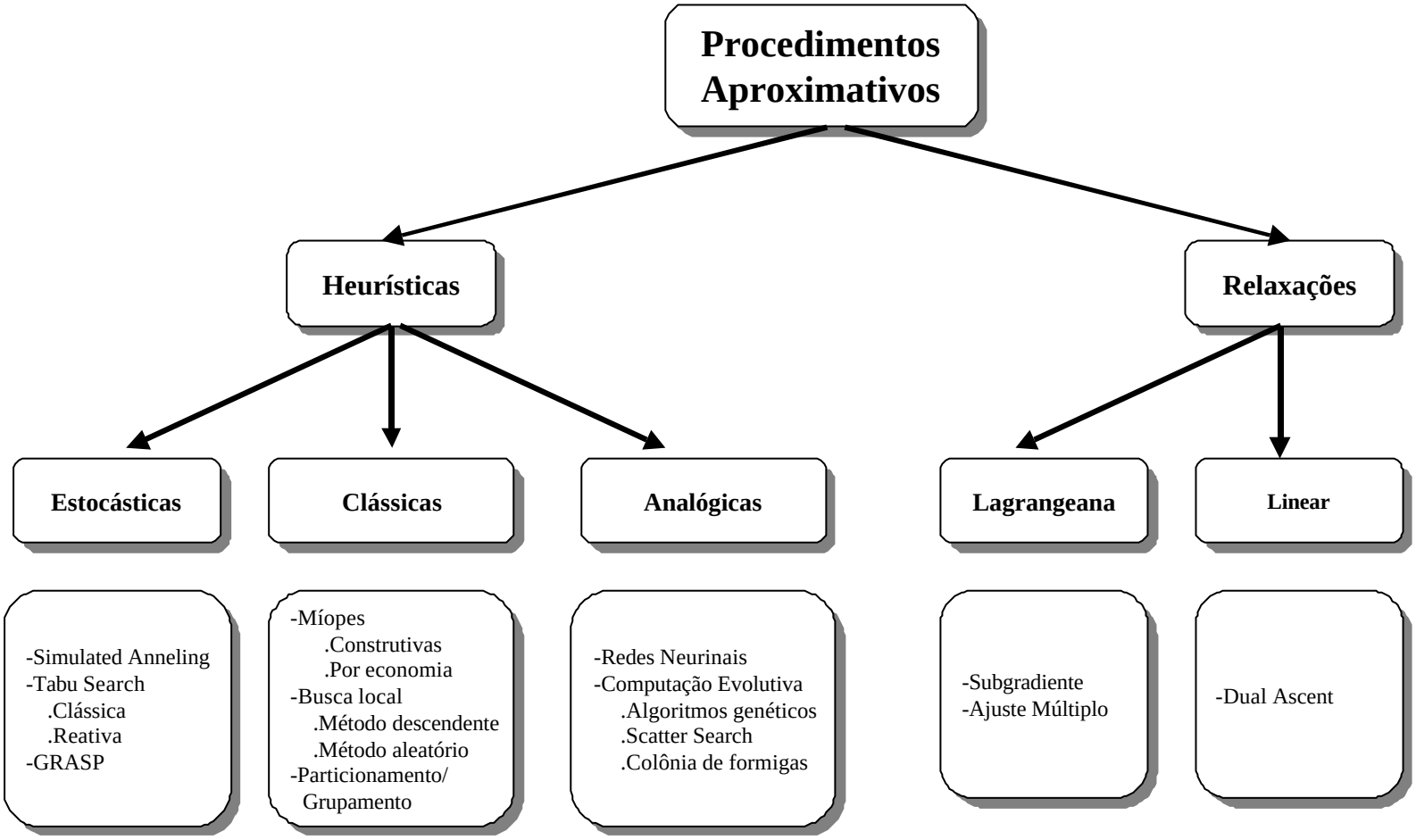


Otimização combinatória

- Ramo de Ciência da Computação que estuda problemas de otimização em conjuntos.
- A solução do problema (ótimo global) será o menor (ou maior) valor possível para a função objetivo para o qual o valor atribuído às variáveis não viole nenhuma restrição.
- O conjunto de possíveis soluções forma o espaço de busca que é discreto.
- Ótimo Local é a melhor solução dentro de uma localidade do espaço de busca:
  - Ocorre quando não é possível encontrar outra solução melhor
  - Motivos:
    - a estratégia do algoritmo;
    - relação de vizinhança entre soluções
- Campo onde se disseminam métodos exatos e métodos aproximativos (heurísticas)

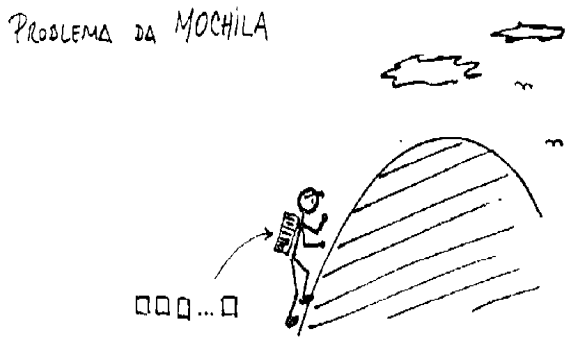
- **Algoritmos exatos (algoritmos polinomiais)**
  - Algoritmos de grafos
  - Enumeração implícita
  - Algoritmos gulosos
  - Algoritmo simplex
  - *Branch-and-Bound*
  - Programação dinâmica
  - Relaxações
  - Planos-de-Corte;
  - Decomposição Dantzig-Wolfe;
  - *Branch-and-Cut e Branch-and-Price*.
- **Algoritmos aproximados:**
  - Algoritmos genéticos (Genetic Algorithms)
  - Busca tabú (Taboo Search)
  - Colônia de formigas (Ant Colony)
  - Greedy Randomized Adaptive Search Procedures (GRASP)
  - Redes neurais (Neural Networks)
  - Recozimento Simulado (Simulated annealing)

Heurísticas



Problema da Mochila (Knapsack)

- Maximizar a satisfação (lucro) com limite de capacidade
- Muitos problemas podem se enquadrar nessa definição



$$\text{Maximizar} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{Sujeito a} \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$x_j \geq 0$ e inteiro, $j = 1, \ldots, n$.

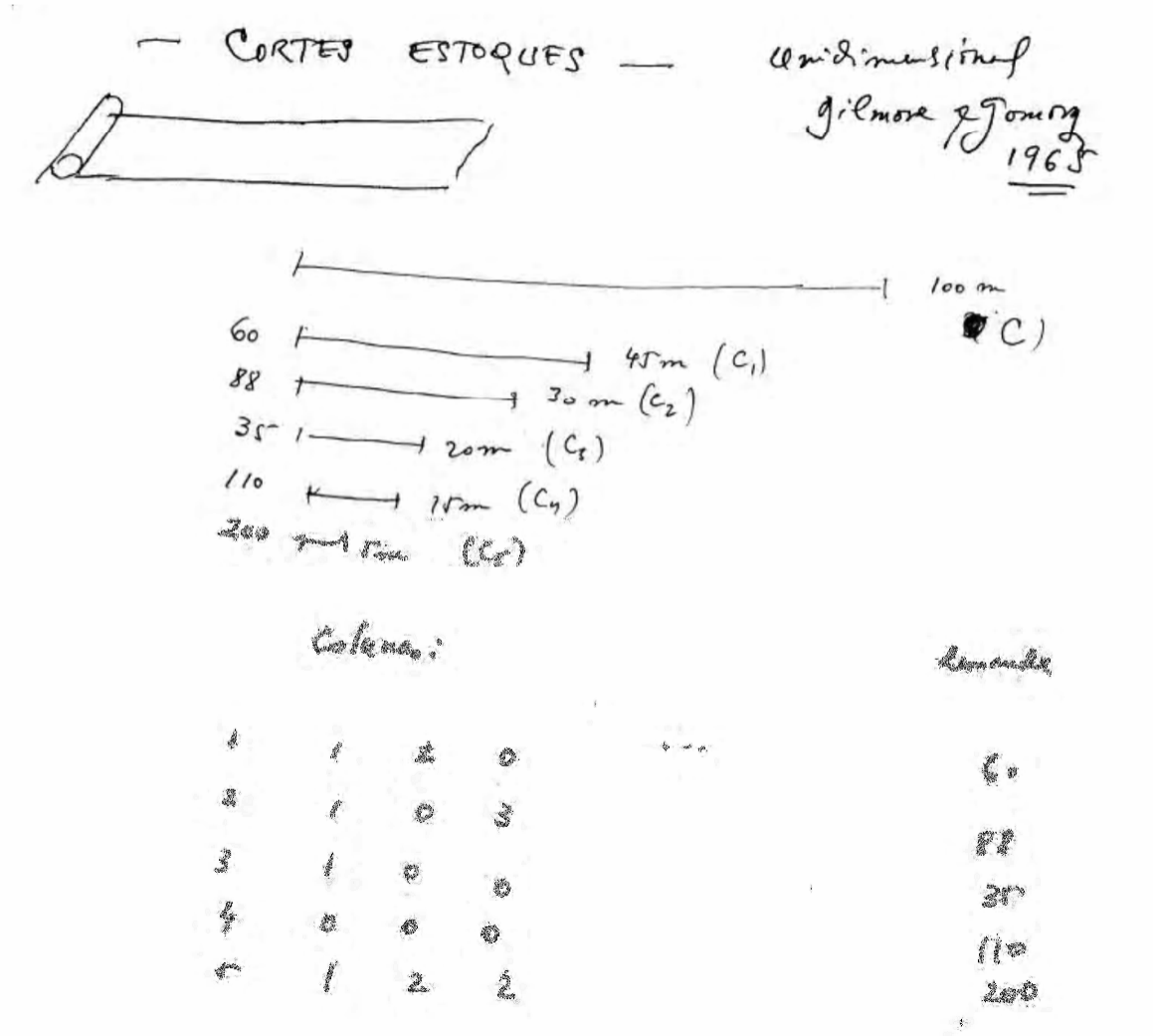
sendo n, b, c_j e a_j inteiros positivos satisf. $a_j \leq b$, $\forall j$.

onde:

- b = peso máximo
- c_i = utilidade do ítem i
- a_i = peso do ítem i
- x_i = 1 se o ítem i é selecionado, ou 0 caso contrário

Este problema é NP-hard.

Problema de cortes



"Uma indústria de aço possui em estoque N barras de comprimento *L* e custo unitário *C.* As barras devem ser cortadas para produzir peças menores considerando-se *n* diferentes itens de comprimento *1* e demanda *d (1=1 n).* Sabendo-se que o número de barras em estoque é suficientemente grande para atender toda a demanda, a empresa deseja encontrar uma solução que minimize o custo de produção."

Problema de cortes

O problema de corte de estoque apresentado pode ser matematicamente formulado considerando-se as *m* possíveis colunas A_1 , A_2 , A_3 ,..., A_m correspondentes aos padrões de corte, onde:

$$A_1 = \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix}, A_2 = \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \\ \vdots \\ a_{n2} \end{pmatrix}, A_3 = \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \\ \vdots \\ a_{n3} \end{pmatrix}, \ldots, A_m = \begin{pmatrix} a_{1m} \\ a_{2m} \\ a_{3m} \\ \vdots \\ a_{nm} \end{pmatrix}$$

e a_{ij} é a quantidade de itens de comprimento l_i existentes no padrão *j*. Portanto, os a_{ij} ´*s* devem respeitar à seguinte restrição:

$$l_1 a_{1j} + l_2 a_{2j} + l_3 a_{3j} + \ldots + l_n a_{nj} \leq L \text{ , } j=1,\ldots,m \text{ e } a_{1j}, a_{2j}, \ldots, a_{nj}$$

Com isto, o problema pode ser modelado como um problema de programação linear inteira da seguinte forma:

$$\text{Minimizar } z = Cx_1 + Cx_2 + Cx_3 + \ldots + Cx_m$$

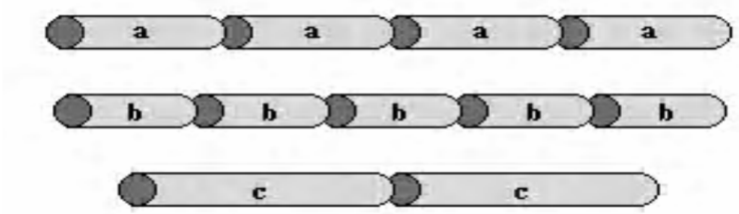
$$\text{Sujeito a: } A_1 x_1 + A_2 x_2 + A_3 x_3 + \ldots + A_m x_m \geq d \text{ ,}$$

$$x_i \geq 0 \text{ e inteiro } (i=1,\ldots,m),$$

onde x_i é a quantidade de vezes que o padrão A_i deve ser cortado e $d = (d_1 ,\ldots, d_n)^T$

Problema de cortes

Solução trivial (padrões de cortes homogêneos)



- Problema unidimensional

$$B = \begin{pmatrix} \left\lfloor \frac{L}{l_1} \right\rfloor & 0 & \cdots & 0 \\ 0 & \left\lfloor \frac{L}{l_2} \right\rfloor & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \left\lfloor \frac{L}{l_m} \right\rfloor \end{pmatrix}.$$

- Problema bidimensional

$$B = \begin{pmatrix} \left\lfloor \frac{L}{l_1} \right\rfloor \times \left\lfloor \frac{W}{w_1} \right\rfloor & 0 & \cdots & 0 \\ 0 & \left\lfloor \frac{L}{l_2} \right\rfloor \times \left\lfloor \frac{W}{w_2} \right\rfloor & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \left\lfloor \frac{L}{l_m} \right\rfloor \times \left\lfloor \frac{W}{w_m} \right\rfloor \end{pmatrix}.$$

Problema de cortes

Geração de colunas (Gilmore and Gomory)

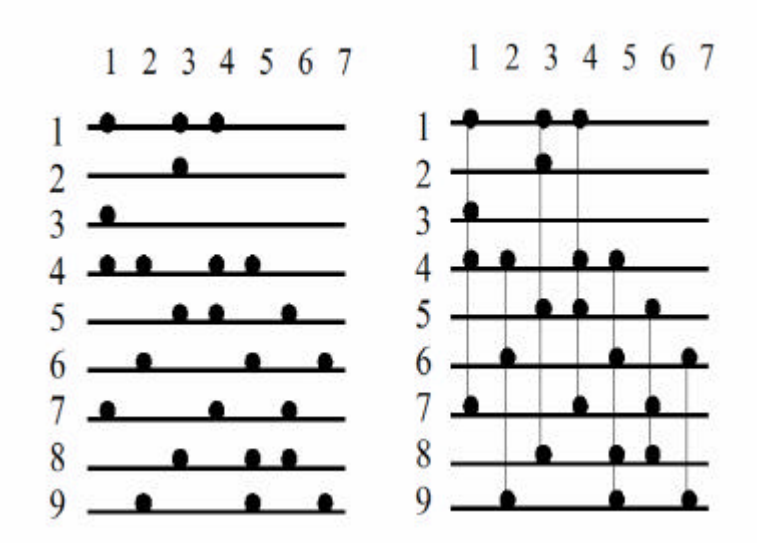
- Iniciar um conjunto trivial de padrões de corte e
- Gerar padrões de corte subseqüentes

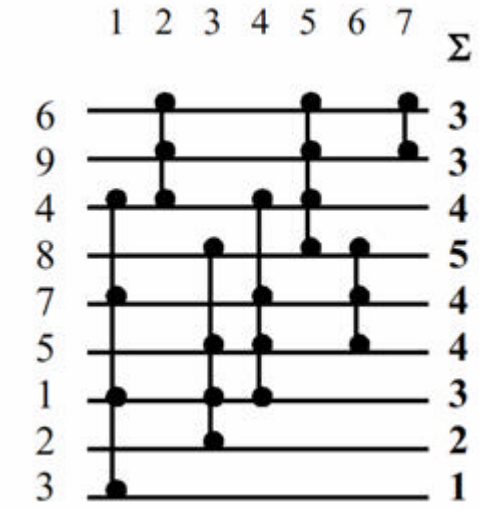
$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{n} x_j \\ \text{Subject to} \quad & \sum_{j=1}^{n} A_{ij} x_j \geq b_i \\ & x_j \geq 0 \end{aligned}$$

$$\begin{aligned} \text{Maximize} \quad & Z = \sum_{i=1}^{m} y_i z_i \\ \text{Subject to} \quad & \sum_{i=1}^{m} W_i z_i \leq L \\ & z_i \text{ integer} \end{aligned}$$

Problemas de seqüenciamento de padrões

Deseja-se fazer a programação de uma máquina (ou o desenho de um circuito VLSI) de forma a reduzir um certo custo associado com a disposição de padrões.



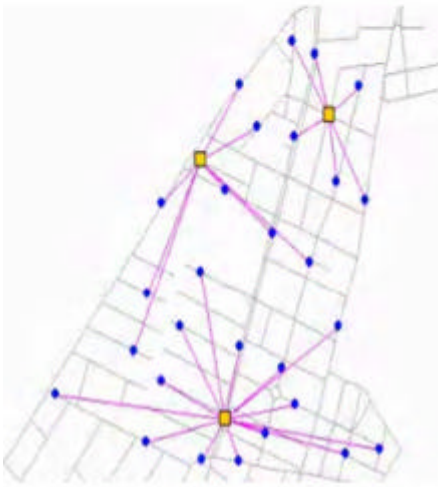


$$P = \begin{matrix} 1\,0\,1\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0 \\ 1\,1\,0\,1\,1\,0\,0 \\ 0\,0\,1\,1\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0\,1\,0 \\ 0\,0\,1\,0\,1\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1 \end{matrix}$$

$$Q = \begin{matrix} 1\,0\,1\,1\,0\,0\,0 & 3 \\ 1\,0\,1\,1\,0\,0\,0 & 3 \\ 1\,0\,1\,1\,0\,0\,0 & 3 \\ 1\,1\,1\,1\,1\,0\,0 & 5 \\ 1\,1\,1\,1\,1\,1\,0 & 6 \\ 1\,1\,1\,1\,1\,1\,1 & 7 \\ 1\,1\,1\,1\,1\,1\,1 & 7 \\ 0\,1\,1\,0\,1\,1\,1 & 5 \\ 0\,1\,0\,0\,1\,0\,1 & 3 \end{matrix}$$

Problema de Localização de p-medianas

Minimizar as distâncias entre os nós (quadras, clientes) e *p* centros (medianas, prestadores de serviços)



$$\text{Min} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \, x_{ij}$$

$$\text{Suj. a} \quad \sum_{i=1}^{n} x_{ij} = 1 \quad ; \quad j = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ii} = p \quad ;$$

$$x_{ij} \leq x_{ii} \quad ; \quad i, j = 1, 2, \ldots, n$$

$$x_{ij} \in \{0, 1\} \quad ; \quad i, j = 1, 2, \ldots, n$$

Onde:

d_{ij} = distâncias

x_{ij} = alocações

p = número de medianas

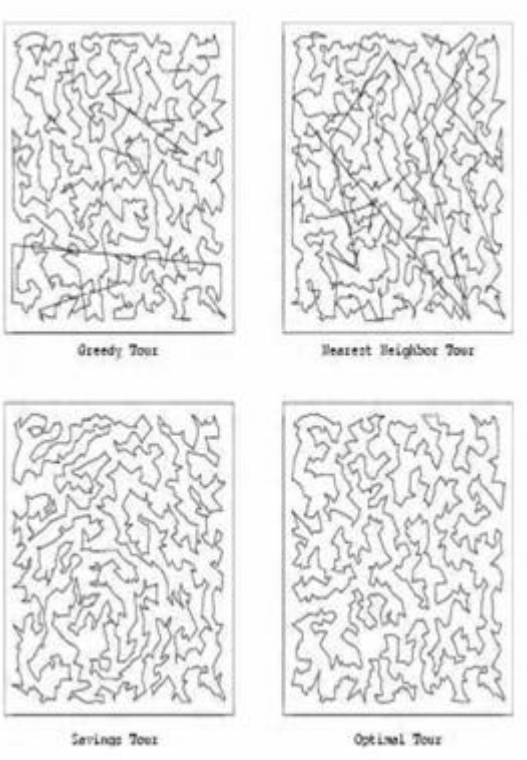
n = número total de nós na rede

Casos especiais:

- P-medianas capacitado
- Cobertura

Problema do Caixeiro Viajante

Minimizar o custo de um ciclo hamiltoniano em um grafo completo.



$$\text{min} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$$

$$\sum_{j \in N} x_{ij} = 1, \quad \forall i \in N$$

$$\sum_{i \in N} x_{ij} = 1, \quad \forall j \in N$$

$$\sum_{i \in S} \sum_{j \in N/S} x_{ij} \geq 1, \quad \forall S \subset N, \; S \neq \emptyset \quad 1 < |S| < n$$

$$x_{ij} \in \{0,1\}, \quad \forall \{i,j\} \in M$$

Onde:

c_{ij} = distância entre as cidades i e j

x_{ij} = 1 se o *tour* passa por i e depois j

N = {1,...,n} é o conjunto de vértices

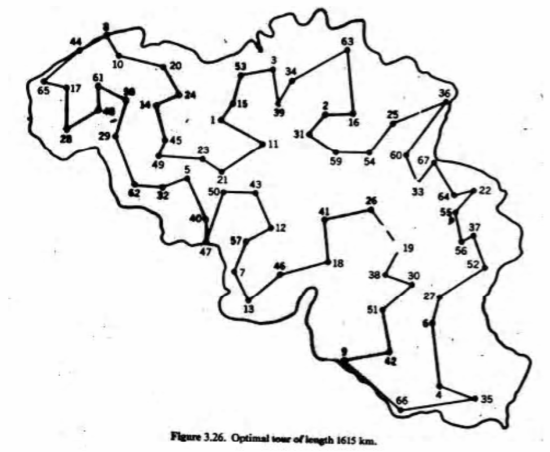


Figure 3.26. Optimal tour of length 1615 km.

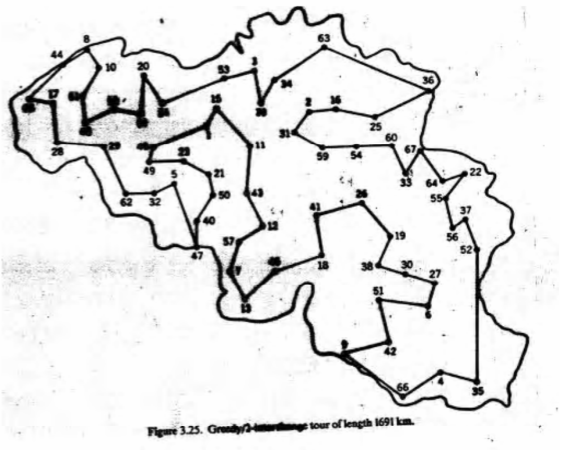


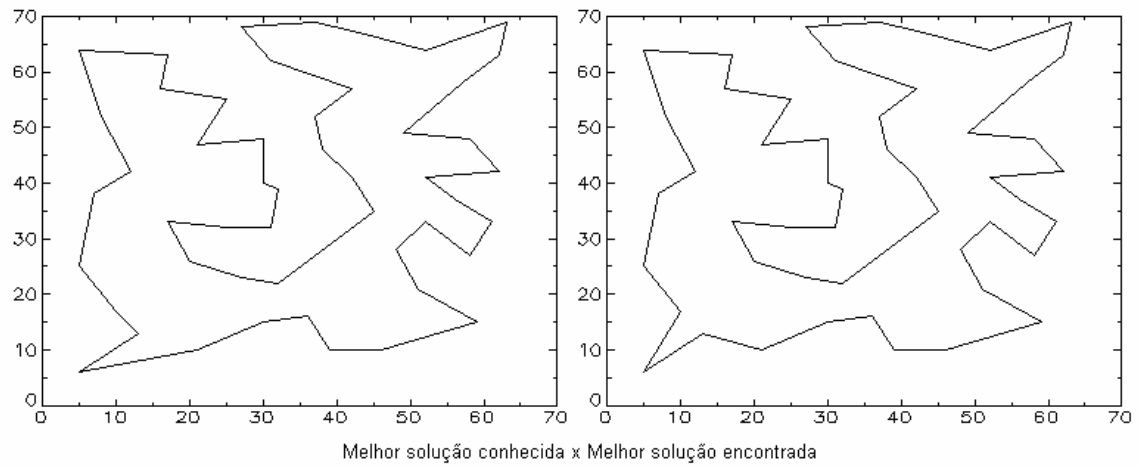
Figure 3.25. Greedy/2-interchange tour of length 1691 km.

Problema do Caixeiro Viajante

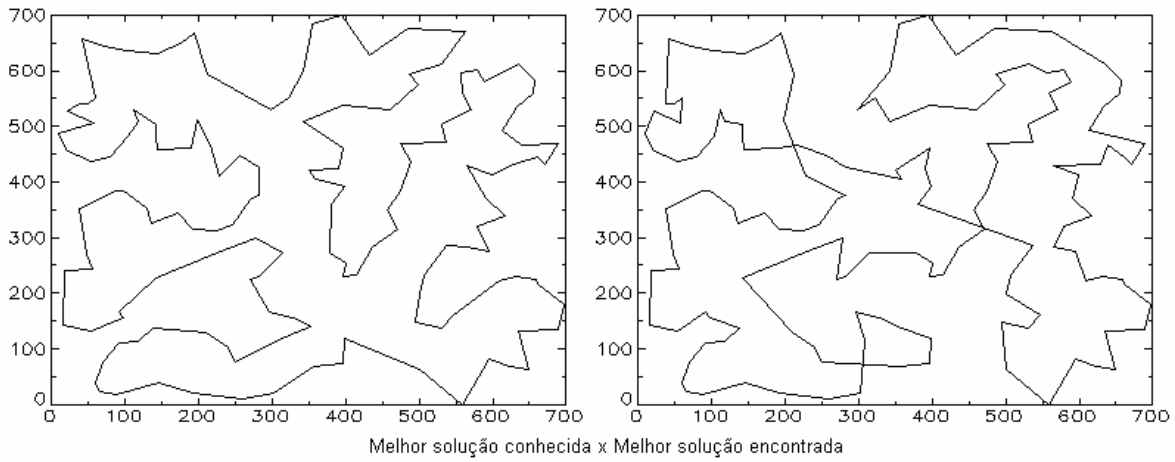
Solução através de algoritmos genéticos

Fuzzy Genetic Algorithm, 1998

Problema	Melhor solução encontrada real / inteira (%)		Melhor solução conhecida
eil51	**428,87**	**426(0.00)**	**426**
eil76	**551,26**	**546(1,48)**	**538**
eil101	**658,79**	**648(3,02)**	**629**
KroA100	**21857,15**	**21854(2,68)**	**21282**
ch150	**6730,27**	**6727(3,04)**	**6528**



Melhor solução conhecida x Melhor solução encontrada



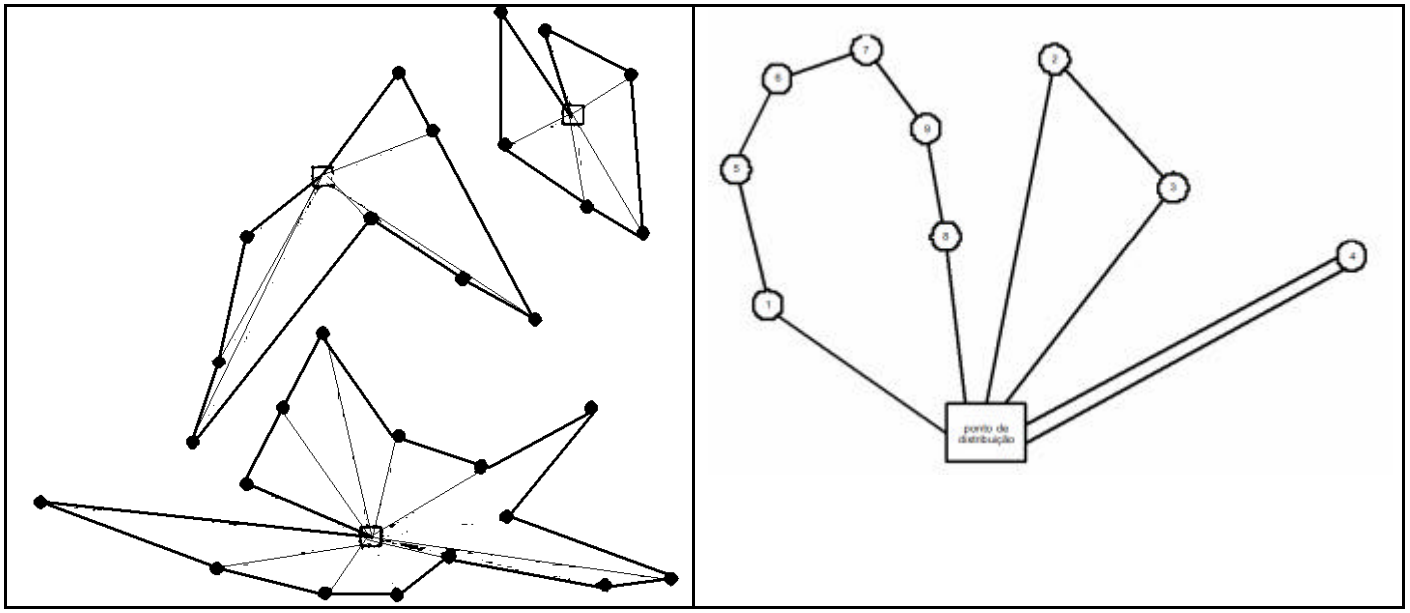
Melhor solução conhecida x Melhor solução encontrada

Problema do Caixeiro Viajante

- *A Hamiltonian Cycle exists in G iff there exists a TSP tour in G' with cost < n. (=)*
  - *If there is a Hamiltonian cycle in G, then the same cycle exists in G' and its price is n. Hence, there exists a TSP tour in G' with cost < n. (=)*
  - *If there is a TSP tour in G' with cost < n, its length is necessarily n and therefore the price of every edge in this tour is 1.*
  - *According to the reduction, these edges exist also in G, and therefore this Hamiltonian cycle exists also in G.*
  - *Building G' and defining the cost function C can be done in polynomial time, and therefore the reduction is polynomial.*
- ***We conclude:***
  - *The Traveling Salesman Problem is NP-complete.*
  - *As the TSP is a subproblem of many vehicle routing problems, it follows that most vehicle routing problem are at least NP-hard.*

PCV+p-Medianas

- Deseja-se encontrar p-rotas de caixeiro viajante, mutuamente excludentes, com custo total < C_{max}



- **Múltiplos caixeiros viajantes**

$$\text{Minimizar} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

s.a. :

$$\sum_{i=1}^{n} x_{ij} = \begin{cases} M, & \text{se } j = 1 \\ 1, & \text{se } j = 2, 3, \ldots, n \end{cases}$$

$$\sum_{j=1}^{n} x_{ij} = \begin{cases} M, & \text{se } i = 1 \\ 1, & \text{se } i = 2, 3, \ldots, n \end{cases}$$