Aula do Curso de Projeto e Análise de Algoritmos (PAA)

Lidando com Problemas NP-Difíceis (Parte 1)

Mário César San Felice

(com materiais da Profa. Carla Negri Lintzmayer do CMCC-UFABC
e do Prof. Flávio Keidi Miyazawa do IC-Unicamp)

Departamento de Computação
Universidade Federal de São Carlos

14 de Fevereiro de 2025

Classes de Complexidade

Classe P: problemas de decisão que possuem algoritmos eficientes.

Classe NP: problemas de decisão cuja solução pode ser verificada em tempo polinomial.

Classe NP-Completo: problemas em NP para os quais todo problema em NP é redutível.

Questão P vs. NP: se houver um algoritmo que resolve qualquer problema NP-Completo em tempo polinomial, então todos os problemas em NP também são resolvidos.

Classe NP-Difícil: problemas para os quais todo problema em NP é redutível.

Para onde ir a partir daqui?

Áreas de aprofundamento sobre algoritmos para problemas em P:

- fluxo em redes
- emparelhamento
- programação linear

Áreas de aprofundamento sobre algoritmos para problemas NP-difíceis:

- métodos exatos
- heurísticas e metaheurísticas
- algoritmos de aproximação
- parametrização

Nestas últimas aulas o foco será no tratamento de problemas NP-difíceis.

Problemas de otimização

Um problema de otimização é caracterizado por

entrada: dados que são recebidos.

soluções viáveis: atribuições de valores às variáveis do problema que respeitam as restrições do mesmo.

função objetivo: função que associa um valor a cada solução viável.

objetivo: encontrar uma solução viável com melhor valor.

Mas, o que significa melhor?

Problemas de otimização combinatória

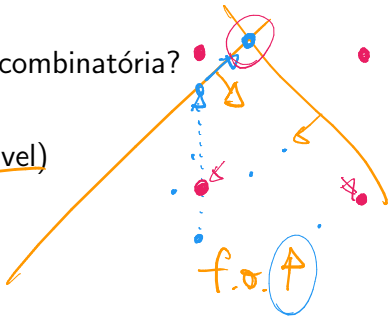
Melhor pode ser

maior: em um problema de maximização queremos uma solução que maximiza o "lucro".

menor: em um problema de minimização queremos uma solução que minimiza o "custo".

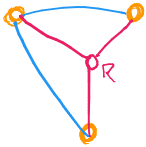
E o que tem de especial os problemas de otimização combinatória?

- As variáveis possuem domínios discretos
- O conjunto de soluções viáveis é finito (enumerável)
- Mas, em geral, este conjunto é muito grande



Projeto de redes

Suponha que vai projetar uma rede conectando capitais brasileiras.



Projeto de redes

Árvore de Steiner

Entrada: $G = (V, E)$ com $V = R \cup S$, sendo R terminais e S vértices de Steiner, e função w de peso nas arestas.

Soluções viáveis: árvores que conectam todos os vértices em R.

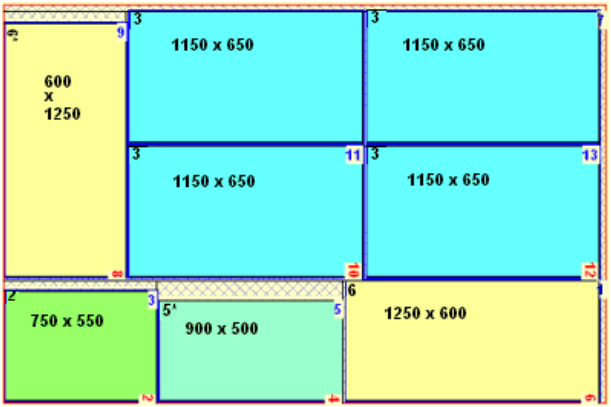
Função objetivo: soma dos pesos das arestas na árvore.

Objetivo: encontrar uma árvore de peso mínimo.

Problemas de corte

Suponha que está construindo um prédio

- as janelas de vidro têm quantidades e tamanhos diferentes
- a vidraçaria vai cortar os pedaços em placas de vidro de tamanhos fixos



Problemas de corte

Bin Packing

Entrada: conjunto $L = \{1, \ldots, n\}$ de itens retangulares, item i com largura w_i e altura h_i, largura W e altura H do recipiente retangular.

Soluções viáveis: partição L_1, L_2, $\ldots$, L_q de L tal que os itens em L_k cabem num recipiente $W \times H$.

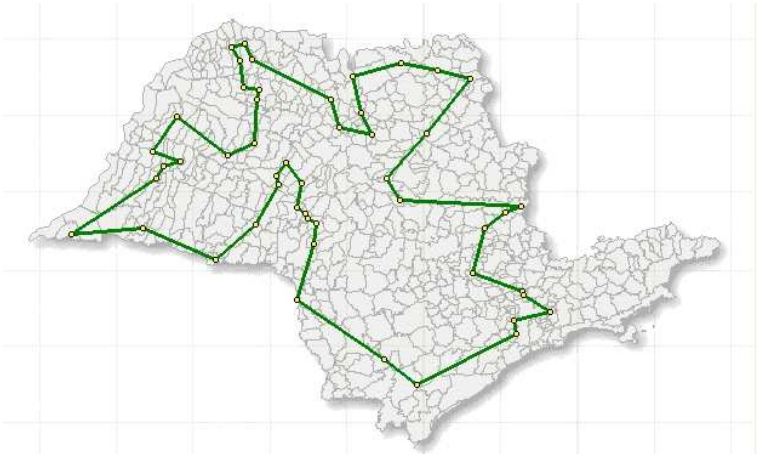
Função objetivo: número q de recipientes (bins) utilizados.

Objetivo: encontrar solução de custo mínimo.

Problemas de percursos

Suponha que está planejando uma viagem pelo estado de São Paulo

- você tem uma lista de cidades que deseja visitar
- e não quer visitar a mesma cidade duas vezes



Problemas de percursos

Caixeiro Viajante (TSP)

Entrada: $G = (V, E)$ e função w de peso nas arestas.

Soluções viáveis: circuitos hamiltonianos de G.

Função objetivo: soma dos pesos das arestas do circuito.

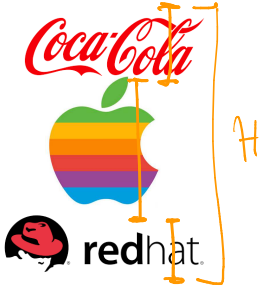
Objetivo: encontrar um circuito de menor custo.

Escolha de propagandas

Suponha que está projetando um site

- e vai manter um retângulo vertical para propagandas
- as propagandas têm largura fixa e alturas variáveis
- cada propaganda fornece um lucro diferente

Lorem ipsum per vulputate malesuada ullamcorper viverra sollicitudin risus sapien, torquent turpis metus ultricies rutrum pretium sollicitudin per aliquam, elit dapibus euismod quis mollis odio platea morbi. nam molestie tortor accumsan eros viverra nulla, nibh sodales maecenas sapien felis, mattis libero id fusce porta. consequat nisi semper neque nam tincidunt congue dolor elementum malesuada, netus orci nulla vulputate aenean ante iaculis imperdiet conubia lorem, hendrerit habitasse aliquam quis scelerisque pharetra mauris aliquet. erat a tincidunt vitae tristique hendrerit lacus etiam elit habitasse, pharetra elementum cubilia sapien facilisis egestas curabitur semper, placerat ut rhoncus metus donec inceptos potenti sed. Donec conubia phasellus cubilia luctus curae dictum est quisque at proin nam justo, tempus condimentum morbi mi rutrum pellentesque non pharetra habitant turpis. feugiat sem faucibus donec fringilla maecenas molestie ultricies aenean, imperdiet lacus iaculis vel mi scelerisque venenatis vivamus, porttitor integer etiam molestie porttitor condimentum rhoncus. facilisis placerat nam hendrerit convallis curabitur aenean aliquam aenean, at



Escolha de propagandas

Problema da Mochila

Entrada: conjunto de n itens, cada item i tem peso w_i e valor v_i, e tamanho W da mochila.

Soluções viáveis: conjuntos de itens $S \subseteq \{1, \ldots, n\}$ com $\sum_{i \in S} w_i \leq W$.

Função objetivo: soma dos valores dos itens em S.

Objetivo: encontrar uma solução de valor máximo.

Abordagem: busca por força-bruta

Algoritmos de busca por força-bruta enumeram todas as soluções

- guardando a melhor encontrada

Dado tempo suficiente, eles resolvem o problema,

- mas sem explorar as estruturas combinatórias do problema
- acabam usando muito esforço computacional

Por conta do tamanho do espaço de busca por soluções,

- costumam ser inviáveis na prática

Força-bruta e o espaço de busca do TSP

No TSP qualquer sequência dos n vértices é candidata à solução.

Algoritmo de busca por força-bruta:

- gere as $n!$ sequências de vértices
- cada uma delas é um circuito Hamiltoniano
- calcule seu custo e compare com o melhor já encontrado

$n!$

Curiosidade: olhando atentamente, são $(n-1)!$ sequências no espaço de busca. Por que?

Força-bruta e o espaço de busca da Mochila

Na mochila qualquer subconjunto dos n elementos é candidato à solução.

Algoritmo de busca por força-bruta:

2^n

- gere os 2^n possíveis subconjuntos
- para cada um, teste se os itens cabem na mochila
- se couberem, calcule o custo da solução e compare com o melhor já encontrado

Comparando tempos

Suponha um computador que executa 10 bilhões de instruções por segundo, i.e., 10GHz.

Seja n o tamanho da entrada e $f(n)$ o # de instruções do algoritmo.

$f(n)$	$n = 10$	$n = 20$	$n = 50$	$n = 100$
n	0.0000000001s	0.000000002s	0.000000005s	0.00000001s
n^2	0.00000001s	0.00000004s	0.00000025s	0.000001s
n^3	0.0000001s	0.0000008s	0.0000125s	0.0001s
n^5	0.00001s	0.00032s	0.03125s	1 segundo
2^n	0.0000001024s	0.000104858s	31.3 horas	$4 \cdot 10^9$ milênios
$n!$	0.00036288s	7.7 anos	-	-

Vale notar que, para $n = 1000$ temos n^5 levando mais de 1 dia.

Abordagens

Se P $\neq$ NP, então não podemos ter algoritmos para problemas NP-Difíceis que, simultaneamente,

1. encontrem soluções ótimas
2. em tempo polinomial
3. para qualquer entrada.

Nos resta não exigir todas essas propriedades do mesmo algoritmo!

Algumas abordagens possíveis são

- métodos exatos
- heurísticas e metaheurísticas
- algoritmos de aproximação
- parametrização e casos particulares

Métodos exatos

Procuram pela solução ótima considerando as estruturas combinatórias dos problemas.

Para problemas NP-Difíceis relaxamos a restrição do tempo polinomial.

Consideram técnicas como

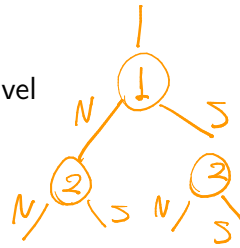
- programação dinâmica
- *branch and bound*
- programação linear / programação linear inteira
- programação por restrições

Branch and Bound

Busca exaustiva inteligente que combina enumeração (branch) das soluções com poda (bound) de soluções inviáveis ou ruins

A estratégia envolve percorrer uma árvore de enumeração e

- sempre que chegamos a um ramo que não é promissor ou viável
- este é eliminado antes de ser percorrido



É importante pensar como

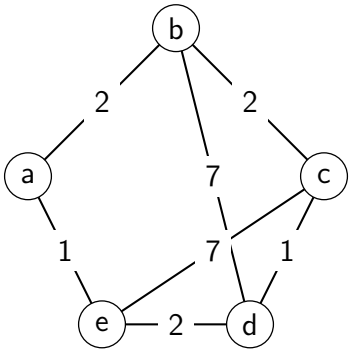
- fazer a enumeração das soluções
- percorrer a árvore (qual ordem usar?)
- podar os ramos da árvore (quais condições testar?)

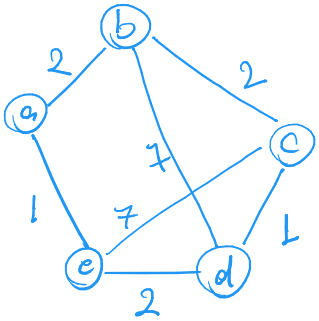
Branch and Bound para o TSP

Ideia: enumerar todas as sequências de vértices, sendo que

- cada nó da árvore de enumeração representa um vértice do grafo
- cada ramificação na árvore representa uma aresta percorrida

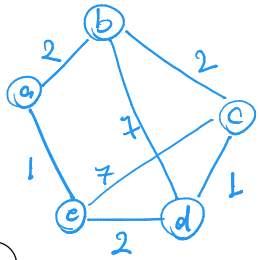
Considere o seguinte grafo

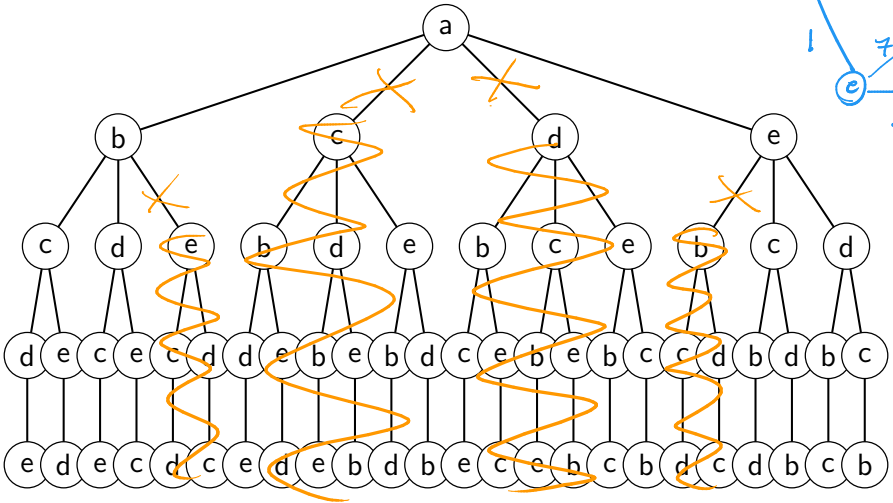




Branch and Bound para o TSP

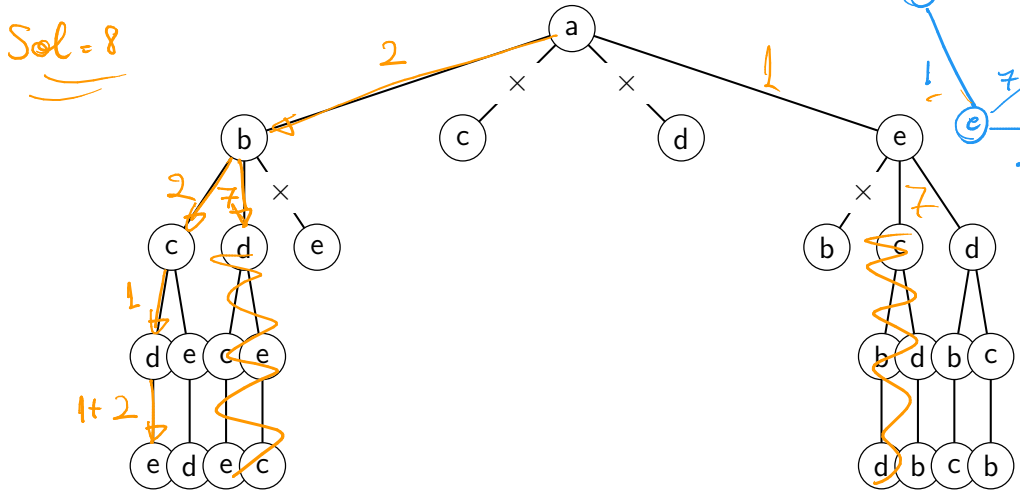
Árvore de enumeração com todas as sequências de vértices.

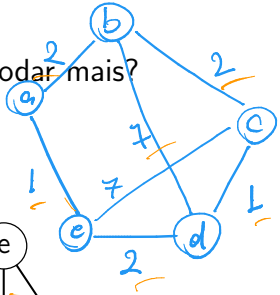




Branch and Bound para o TSP

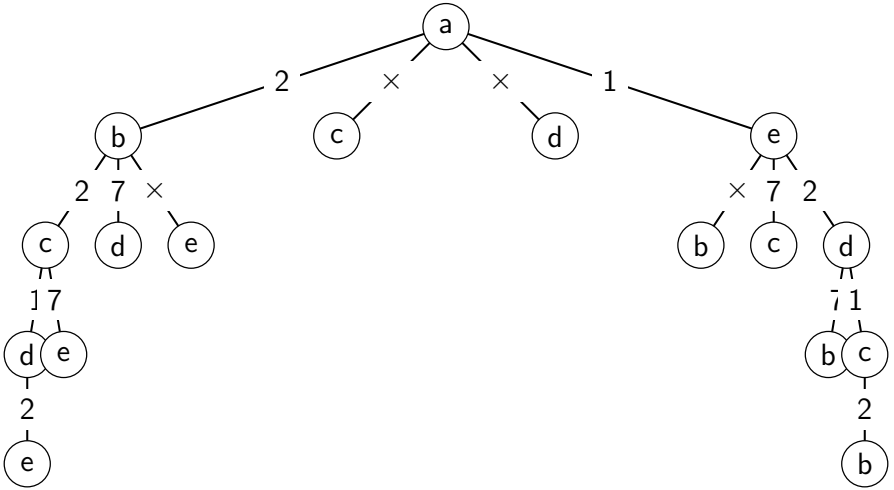
Árvore podada das soluções inviáveis. Será que conseguimos podar mais?





Branch and Bound para o TSP

Árvore podada das soluções não promissoras.



Programação Linear

Modelo matemático que descreve problemas de otimização.

Um programa linear possui

- um conjunto de variáveis reais
- um conjunto de restrições, que são inequações lineares
- uma função objetivo, que é uma expressão linear

Uma atribuição de valores para as variáveis é uma solução viável

- se todas as restrições são satisfeitas

Programação Linear

A forma padrão de um programa linear para um problema de minimização com n variáveis e m restrições é

$$\begin{aligned} \text{minimizar} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 \qquad\qquad \forall j \in \{1, \ldots, n\} \end{aligned}$$

onde a_{ij}, b_i e c_j são constantes e x_j são as variáveis.

Em um programa linear inteiro, as variáveis são inteiras,

- muitas vezes são binárias, chamadas variáveis de decisão

Formulação em PLI da Mochila

Sejam x_i variáveis binárias que indicam se o item i foi escolhido.

$$\text{maximizar} \quad \sum_{i=1}^{n} v_i x_i$$

$$\text{sujeito a} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0, 1\} \qquad \forall i \in \{1, \ldots, n\}$$

$0 \leq x_i \leq 1 \rightsquigarrow$ relaxada

Programação Linear Inteira

Programas lineares podem ser resolvidos em tempo polinomial.

Programas lineares inteiros são, em geral, problemas NP-difíceis.

Resolvedores de PLI usam o algoritmo *branch and bound* junto de PL

- para encontrar melhores limitantes inferiores
- e assim realizar mais podas nos ramos da árvore

Heurísticas

São algoritmos que geram uma solução viável

- mas não dão garantias sobre a qualidade da solução gerada
- Ou seja, relaxamos a restrição de exigir soluções ótimas

Podem ser

- **construtivas**, que produzem uma solução viável, muitas vezes usando estratégias gulosas
- **de busca local**, que partem de uma solução inicial e tentam melhorá-la através de pequenas modificações

Muitas vezes usam aleatoriedade pra explorar mais soluções do espaço de busca.

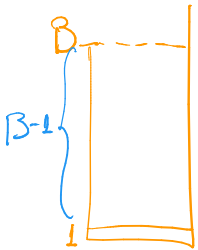
Heurística construtiva para a Mochila

1: **função** MochilaGuloso(n, w, v, W)

2: ordene e renomeie os itens para que $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$

3: seja q um inteiro tal que $\sum_{i=1}^{q} w_i \leq W$ e $\sum_{i=1}^{q+1} w_i > W$

4: **devolve** $v_1 + v_2 + \ldots + v_q$



O que acontece no seguinte cenário?

$W = B$

$v_1 = 2$ $\quad w_1 = 1$ $\quad \frac{v_1}{w_1} = 2$

$v_2 = B$ $\quad w_2 = B$ $\quad \frac{v_2}{w_2} = 1$

razão

ordem fração espaço livre = B

$1 \ll 2$

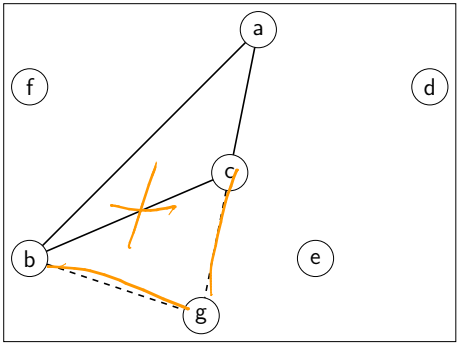
ALG = 2

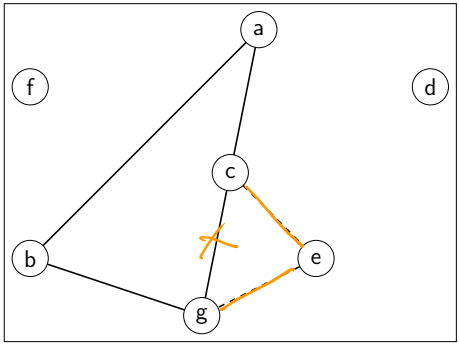
OPT = B

Heurística construtiva para o TSP

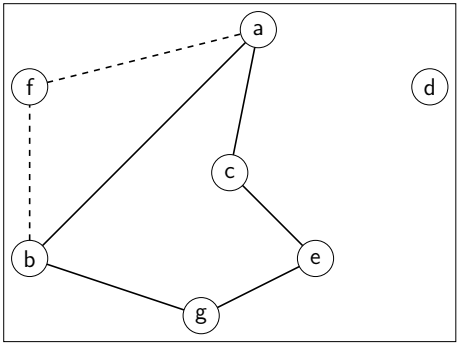
1: **função** TSP-Guloso1($G = (V, E)$, w)
2: seja C um circuito trivial
3: **enquanto** C não é hamiltoniano **faça**
4: escolha uma aresta $uv \in C$
5: seja x um vértice que não aparece em C e tal que $w(ux) + w(xv)$ é mínimo
6: $C \leftarrow C - uv + ux + xv$
7: **devolve** C

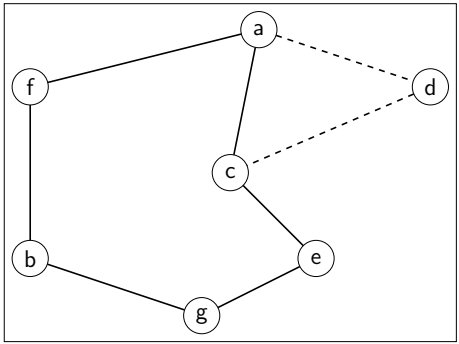
Heurística construtiva para o TSP





Heurística construtiva para o TSP

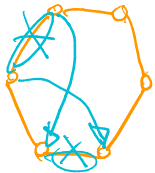




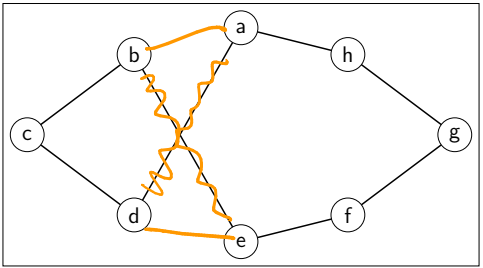
Heurística de busca local para o TSP

Vizinhança 2-OPT de um circuito hamiltoniano C:

- conjunto dos circuitos hamiltonianos que são obtidos
- removendo 2 arestas de C e inserindo outras 2 arestas



Exemplo de troca 2-OPT



Pseudo-código do algoritmo para o TSP

1: **função** $\mathrm{TSP\text{-}2\text{-}OPT}(G = (V, E),\ w)$
2: encontra um circuito hamiltoniano inicial C
3: **enquanto** houver um circuito C' na vizinhança 2-OPT de C tal que $w(C') < w(C)$ **faça**
4: $C \leftarrow C'$
5: **devolve** C

Mais sobre vizinhanças do TSP

Podemos generalizar a Vizinhança 2-OPT para Vizinhança k-OPT:

- conjunto de circuitos hamiltonianos que são obtidos
- removendo k arestas de C e inserindo outras k arestas

Exemplo de troca 3-OPT