3.4.5 Problemas de caixeiro-viajante 186

Problema do Caixeiro Viajante

The Travelling Salesman Problem-TSP - Problema Del viajante

Suponhamos que a qualquer momento em que realizamos uma entrega a n clientes podemos usar apenas um veículo, ou seja, que a capacidade do veículo não é problema. Nesse caso, iremos despachar um único veículo do depósito para os n clientes, com o veículo retornando ao depósito após a entrega final. Este é o Problema do Caixeiro Viajante-TSP.

O TSP pode ser assim formulado: existem n pontos e caminhos entre todos eles com comprimentos conhecidos. O objetivo é encontrar o caminho mais curto, que passa por todos os pontos uma única vez, começando e terminando no mesmo ponto. Isso significa que o objetivo é encontrar a viagem de ida e volta mais curta possível.

Dado, portanto, um conjunto de cidades e uma matriz de distâncias entre elas, o TSP consiste, então, em encontrar uma rota que:

- parta da cidade origem;
- passe por todas as demais cidades uma única vez;
- retorne à cidade origem ao final do percurso;
- percorra uma rota que dá a menor distância possível.

Esta rota é conhecida como ciclo fechado Hamiltoniano[1].

Não parece ser tão óbvio, mas o problema do TSP é um problema de otimização de difícil resolução.

1 Problemas do tipo hamiltoniano requerem que cada nó seja visitado uma única vez. Problemas do tipo euleriano requerem que cada aresta seja percorrida uma única vez. Um grafo euleriano é um grafo conexo onde cada nó tem um grau par.

I - Formulação matemática

Formulação de Miller–Tucker–Zemlin (1960) do problema clássico do TSP

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

$$\begin{array}{lll} \text{sa} & \sum_{i=1}^{n} x_{ij} = 1, & j = 1, 2, \ldots, n \\ & \sum_{j=1}^{n} x_{ij} = 1, & i = 1, 2, \ldots, n \\ & u_i - u_j + nx_{ij} \le n - 1 & \forall 2 \le i \ne j \le n \\ & x_{ij} \in \{0,1\} & \forall i, j \end{array}$$

Para dimensões elevadas, a resolução por métodos de programação matemática é proibitiva em termos de tempos computacionais. Portanto, a abordagem de solução heurística e de aproximação é mais útil para as aplicações que preferem o tempo de execução do algoritmo em relação à precisão do resultado.

II - Métodos heurísticos

Métodos heruísticos são procedimentos que seguem uma intuição para resolver o problema (forma humana de resolver o problema, fenômenos naturais, processos biológicos, etc.). Não garantem a otimalidade da solução final mas em geral produzem rapidamente soluções finais de boa qualidade.

Heurísticas para resolver o problema do caixeiro viajante são métodos por meio dos quais a solução ótima para o TSP é procurada. Embora as heurísticas não possam garantir que a solução ótima seja encontrada, ou não ao menos em tempo real, um grande número delas encontrará uma solução próxima à ótima, ou mesmo encontrará uma solução ótima para certos casos do problema do caixeiro viajante.

Quando se trata de solucionar o TSP, as variações desta classe de algoritmo, que se apresentam, são classificadas em <u>construtivas</u> e <u>de melhoria</u>.

i) <u>Heurística construtiva</u> - cria uma solução a partir do zero (começando do nada). Usando heurística construtiva constrói-se uma solução passo a passo segundo um conjunto de regras pré-estabelecido. Estas regras estão relacionadas com:

- a escolha do ciclo ou nó inicial da solução – **inicialização**;
- um critério de escolha do próximo elemento a juntar à solução – **seleção**;
- a escolha da posição onde esse novo elemento será inserido – **inserção**.

As heurísticas construtivas apresentadas neste texto são:

- a inserção do vizinho mais próximo;
- a inserção mais barata;
- algoritmo de Christofides;
- algoritmo de Clarke e Wright (método das economias/*savings*);
- metodologia Convex Hull.

ii) <u>Heurística de melhoria</u> - inicia com uma solução e, em seguida, tenta melhorar a solução, geralmente fazendo pequenas alterações na solução atual:

- 2-opt;
- 3-opt;
- . . .

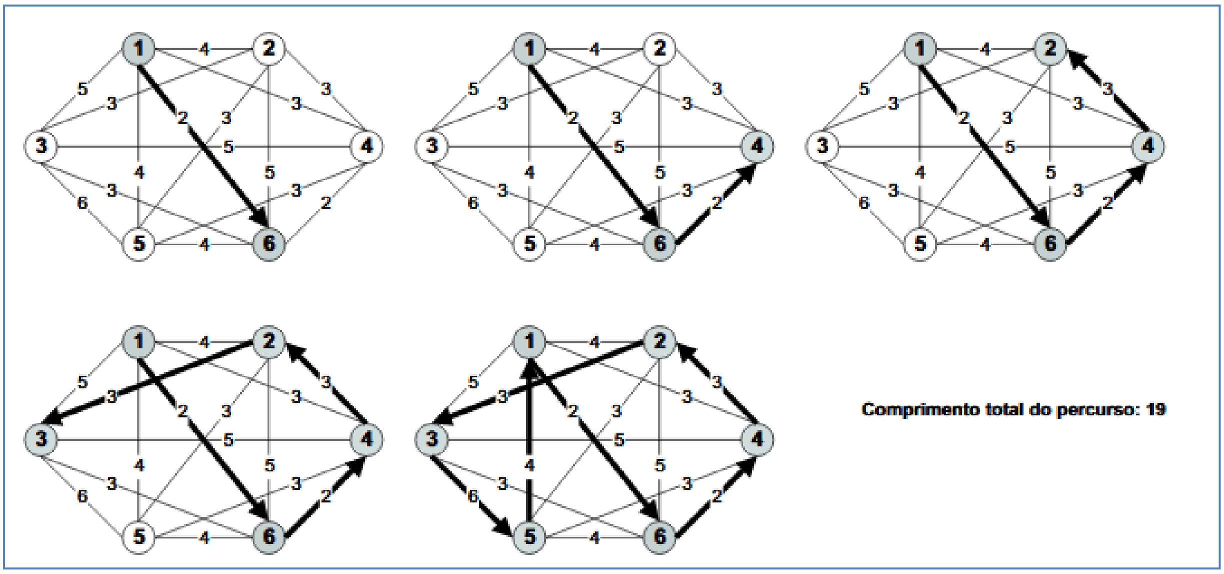
As heurísticas de melhoria começam com uma solução viável e buscam uma solução melhorada que possa ser encontrada executando um número pequeno de alterações.

a) Heurística construtiva <u>Inserção do Vizinho mais Próximo</u>

Construir uma rota adicionando, a cada passo, o nó mais próximo do último nó inserido e que ainda não tenha sido visitado.[2]

1. Inicializar a rota com apenas um nó escolhido aleatoriamente.
2. Encontrar o nó k fora da rota atual cuja aresta que o liga ao nó atual possui custo mínimo.
3. Adicionar o nó k no final da lista de nós visitados.
4. Se todos os nós foram adicionados parar, senão voltar ao passo 2.

Exemplo 1 https://paginas.fe.up.pt/~mac/ensino/docs/OR/CombinatorialOptimizationHeuristicsLocalSearch.pdf



[2] Construir a rota escolhendo a aresta de menor custo saindo do nó em que se está.

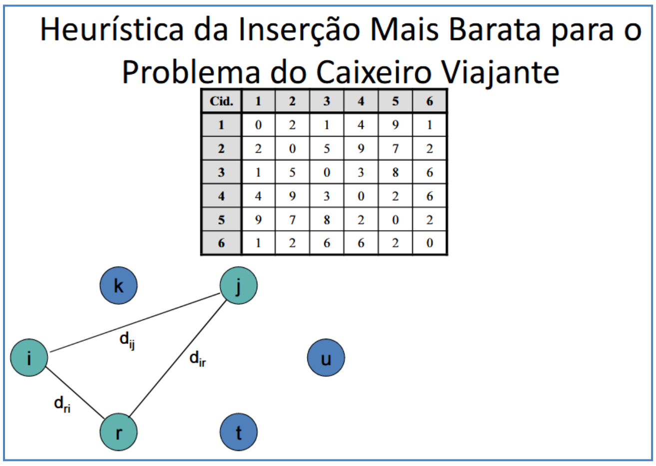
b) Heurística construtiva Inserção da Aresta mais Barata

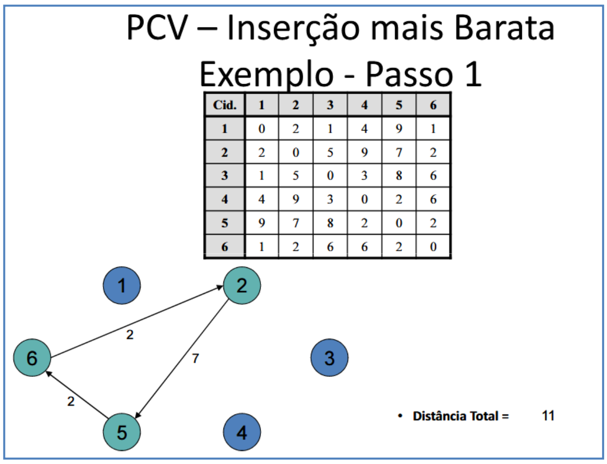
Construir uma rota, partindo de um rota inicial, formada por 3 nós. Adicionar a cada passo o nó k ainda não visitado entre a aresta (i, j) de nós já visitados, cujo custo de inserção seja o mais barato.

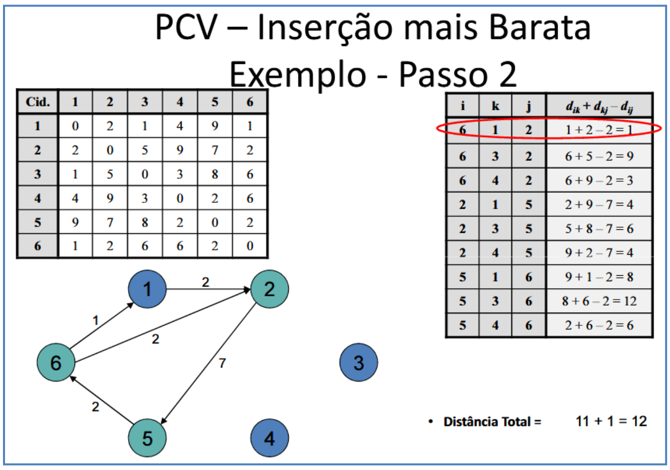
1. Inicializar um ciclo formado por 3 nós.
2. Encontrar o nó k fora do ciclo atual e o par de arestas (i, k) e (k, j) que ligam o nó k ao ciclo minimizando $c_{ik} + c_{kj} - c_{ij}$.
3. Inserir as arestas (i, k) e (k, j) e retirar a aresta (i, j).
4. Se todos os nós estão inseridos parar, senão voltar ao passo 2.

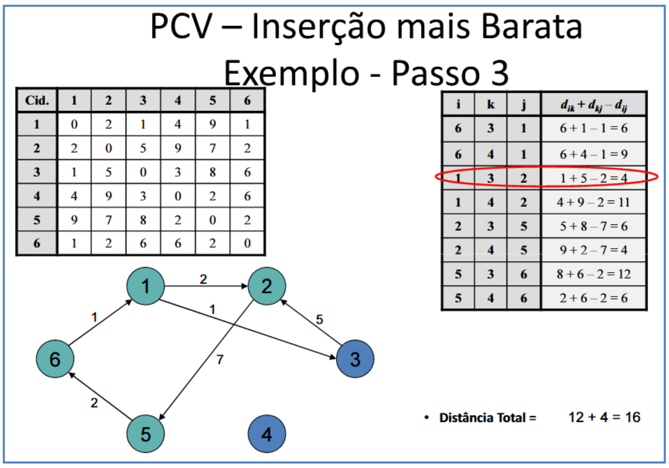
A etapa 1. pode assim ser executada: i) Comece com apenas um nó i, escolhido aleatoriamente. ii) Encontre os nós r e j para o qual o $c_{ir} + c_{rj} - c_{ij}$ é minimizado. iii) Insira r entre i e j. Assim, teremos um ciclo formado por 3 nos e 3 arestas.

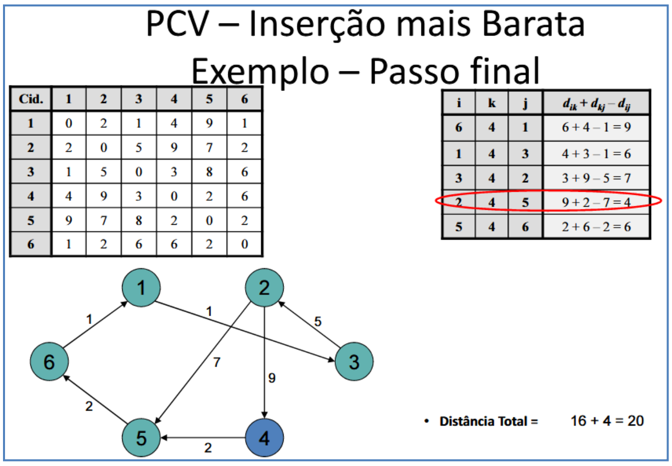
Exemplo





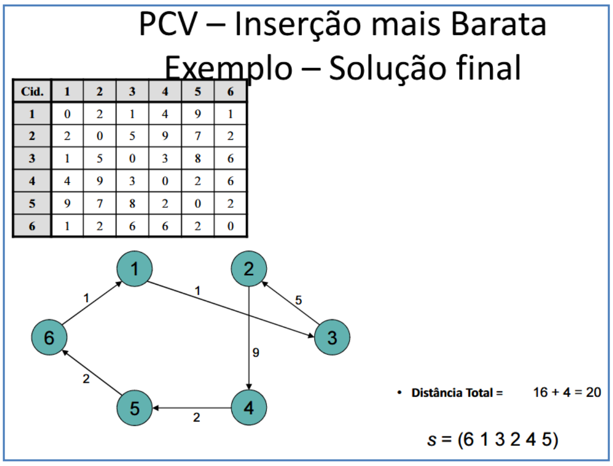






Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

i	k	j	$d_{ik} + d_{kj} - d_{ij}$
6	4	1	6 + 4 – 1 = 9
1	4	3	4 + 3 – 1 = 6
3	4	2	3 + 9 – 5 = 7
2	4	5	9 + 2 – 7 = 4
5	4	6	2 + 6 – 2 = 6



Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

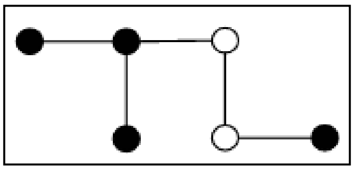
c) Heurística construtiva Algoritmo de Christofides

https://personal.vu.nl/r.a.sitters/AdvancedAlgorithms/2016/SlidesChapter2-2016.pdf

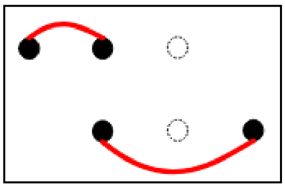
Em 1976 Nicos Christofides (Imperial College, Londres) desenvolveu um algoritmo eficiente que produz um caminho cujo custo excede o ótimo em menos de 50%. Portanto, a heurística de Christofides permite determinar uma solução aceitável para o Problema do Caixeiro Viajante, ainda que possa não ser a solução ótima.

Considere-se um grafo $G = (N, E)$ completo, onde o custo associado a cada aresta é não negativo.

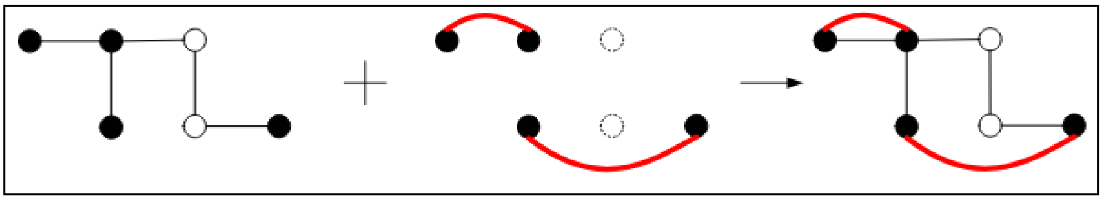
1. Encontrar, em $G = (N, E)$ a árvore geradora de custo mínimo T.



2. Seja W o conjunto dos nós de T com grau ímpar e seja M o emparelhamento/*matching* perfeito[3] de G[W] (subgrafo de G induzido por W).



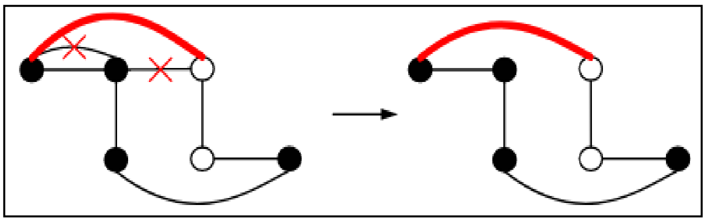
3. Seja $J = T \cup M$, conjunto das arestas de um grafo conexo onde cada nó tem grau par.



[3] Dado um grafo G com um número par de nós; um *matching* perfeito é um subconjunto de arcos, de modo que cada nó de G é um ponto final de exatamente uma aresta. *Matching* num grafo é um conjunto de arestas sem nós comuns, ou seja, sem arestas adjacentes. Pode ser feito usando o Algoritmo de Edmond Blossom (1965).

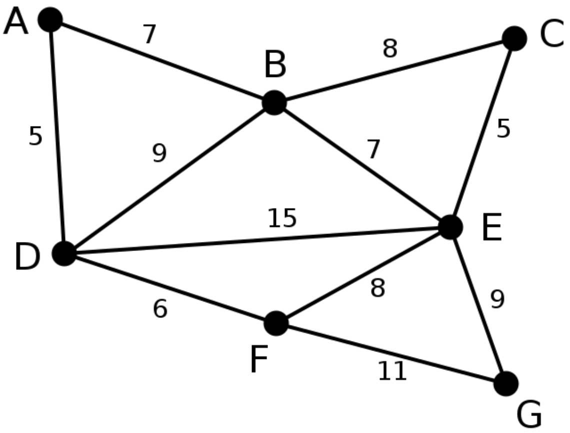
4. Se todos os nós de J apresentam grau 2, o algoritmo termina e a solução é dada pelo grafo cujas arestas são J.

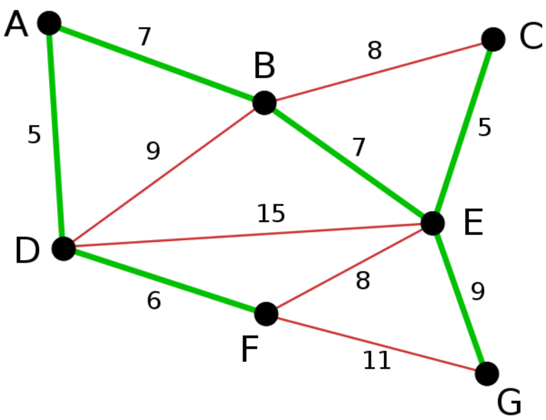
5. Caso contrário[4], seja v um nó qualquer com grau no mínimo 4 em (V,J). Então existem arestas (u,v) e (v,w) que se forem excluídas de J e adicionando a J a aresta (u,w), o subgrafo permanece conexo e ainda com grau par em todos os nós. Fazendo este "*shortcut*" e repetindo o processo até que todos os nós tenham apenas 2 arestas, obtemos a solução.



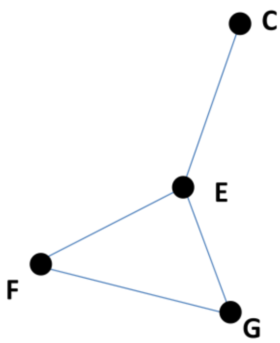
Exemplo 1

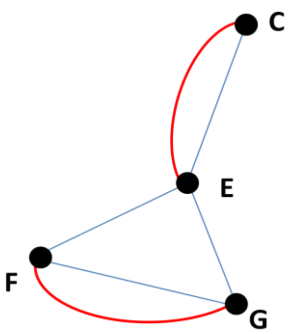
Consideremos o grafo G = (N, Ar), onde N = {A, B, C, D, E, F, G} são 7 cidades e Ar os respectivos custos entre cada par. Utilizando o algoritmo de Kruskal obtem-se a árvore geradora mínima T.





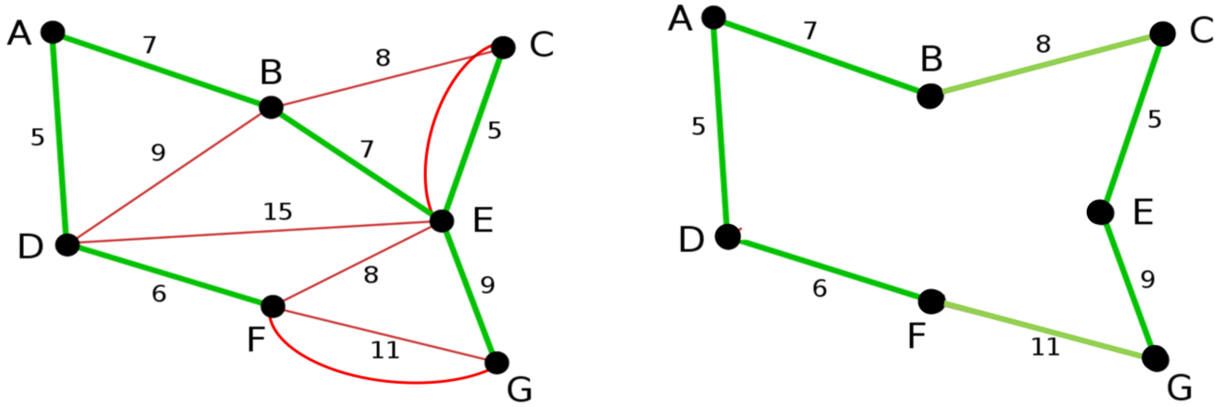
Abaixo, temos, W={C,E,F,G}. G[W] é dado na primeira figura (abaixo). O *matching* perfeito é dado pelo conjunto M={(C,E),(F,G)} (segunda figura).





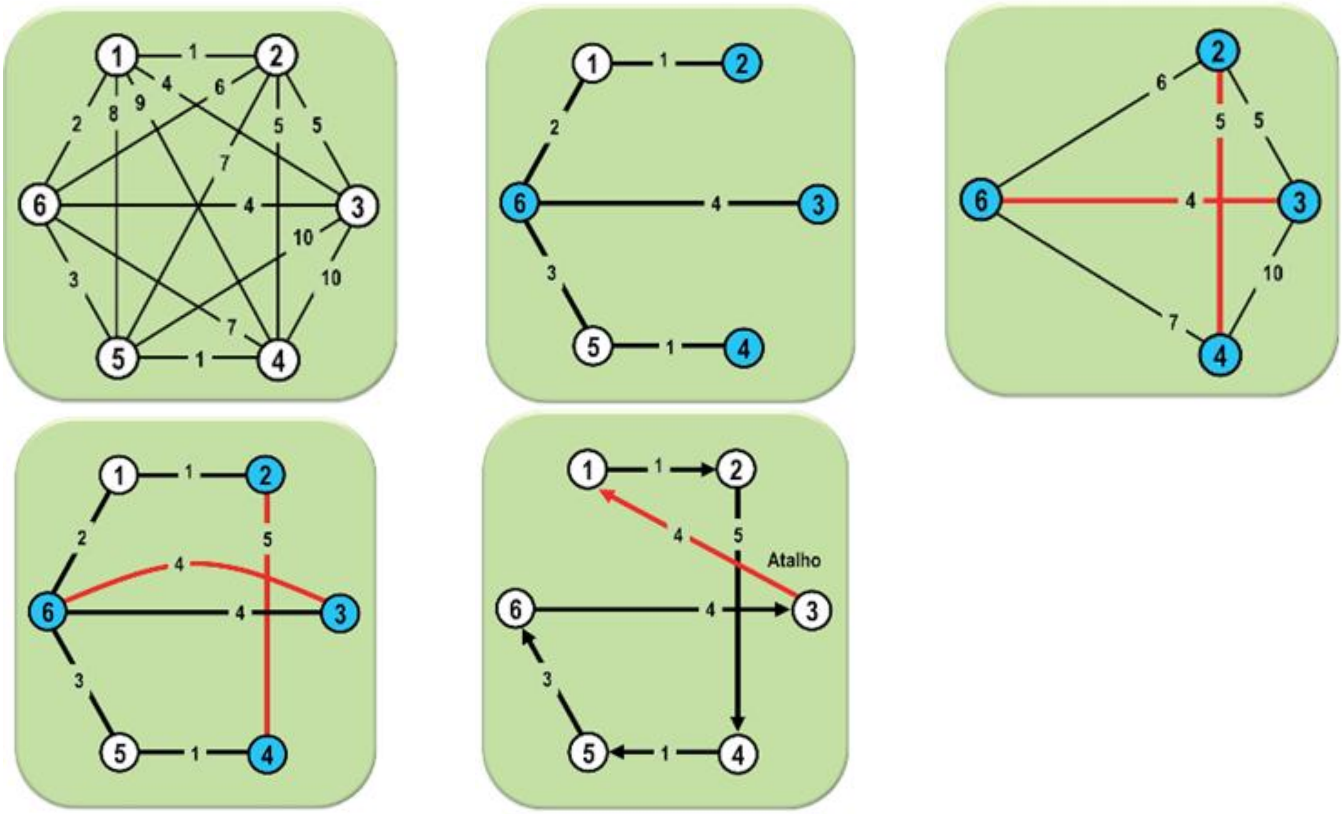
[4] Crie um ciclo hamiltoniano em J e reduza-o a uma solução plausível C usando a desigualdade triangular. Desigualdade triangular: se d(a,b) mede o comprimento do caminho mais curto entre os pontos a e b, então $d(a,c) \leq d(a,b) + d(b,c)$, porque um caminho ótimo de a para b, juntamente com um caminho ótimo de b para c, não pode ser melhor do que o caminho ótimo de a para c.

Para $J = E(T) \cup M$,tem-se o grafo

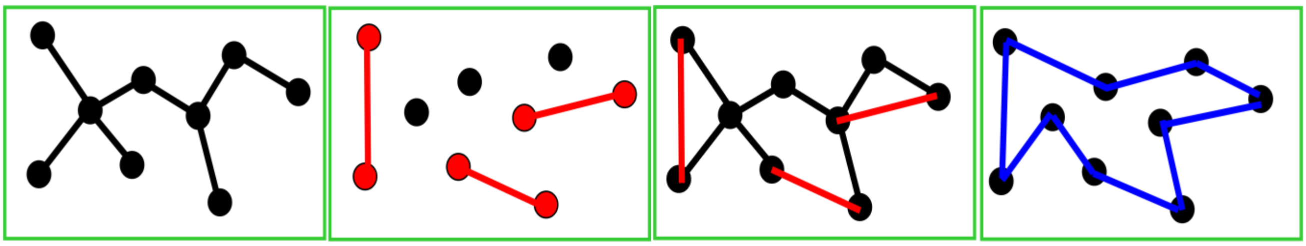


Procedendo o "*shortcut*", retirando (C,E) e (E,B), inserindo (C,B) e (F,G), obtemos o grafo onde todos os nós têm grau 2. Concluindo assim o método de Christofides, onde a solução é dada pela segunda figura acima. O custo é: 5+6+11+9+5+8+7.

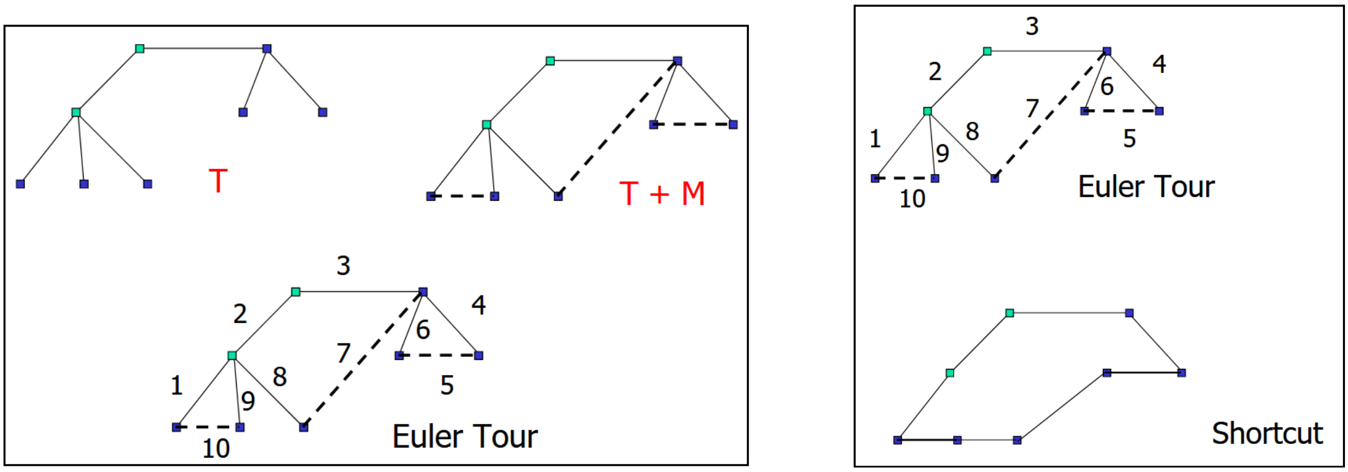
Exemplo 2



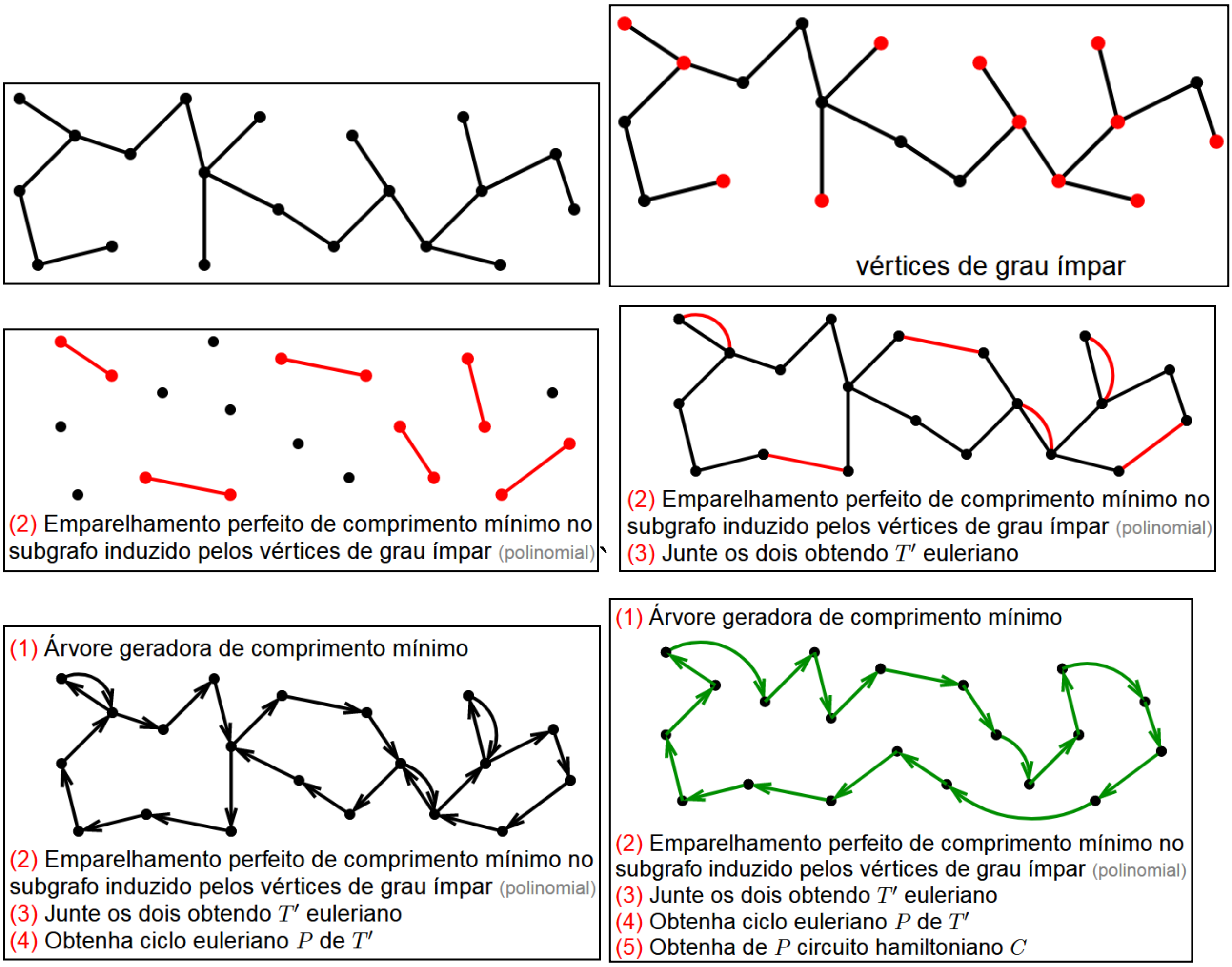
Exemplo 3



Exemplo 4



Exemplo 5 https://www.ime.usp.br/~cris/aulas/12_2_5727/slides/aula6.pdf

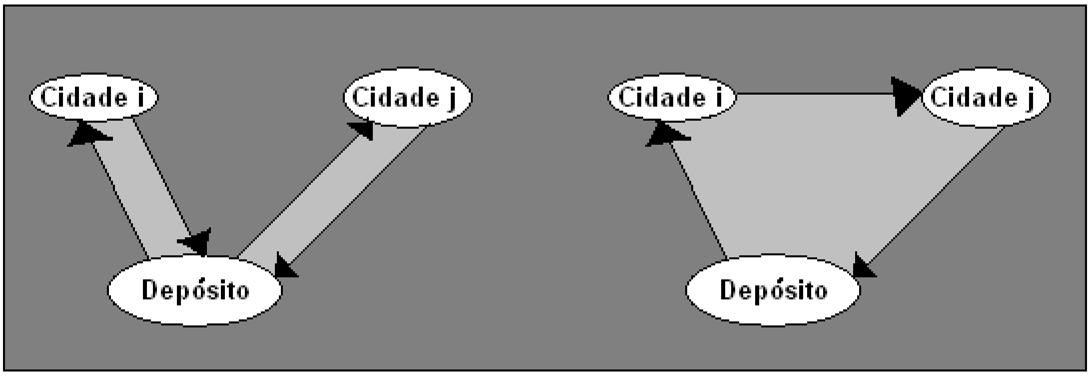


d) Heurística construtiva Algoritmo Clarke e Wright (método das economias/*savings*)

A heurística das economias de Clarke e Wright (CW, 1964) foi originalmente desenvolvida para resolver o problema de roteamento de veículos. Baseia-se na noção de economias/*savings*, que pode ser definido como o custo da combinação, ou união, de duas sub-rotas existentes. Trata-se de uma heurística iterativa de construção baseada numa função gulosa de inserção.

Inicialmente, cada nó é servido por um veículo, constituindo rotas entre o depósito e cada cliente. Seja c_{ij} o custo de viagem partindo de um nó i a um nó j, podendo ser dado em distância percorrida ou tempo de deslocamento. Duas rotas contendo os nós i e j podem ser combinadas, desde que i e j estejam ou na primeira ou na última posição de suas respectivas rotas.

Em cada iteração, todas as combinações de rotas possíveis são analisadas através da fórmula $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, onde 0 representa a origem. As duas rotas que renderem a maior economia de combinação são unidas. Por ser sempre escolhida a maior economia dentre as possíveis, a função de escolha é dita gulosa. Como a cada nova combinação de rotas, nova rota é gerada, o método é dito iterativo,

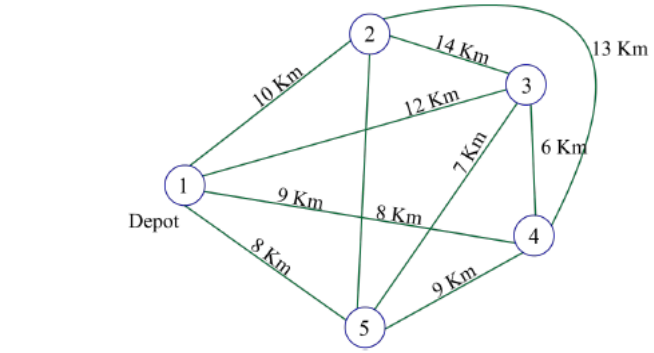


<u>Algoritmo de Clarke e Wright – Economias/*Savings* – para o TSP</u>

1. Selecione o nó origem, e denomine-o de nó 0
2. Calcule as economias $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ para $i, j = 1, \dots, n$, $i \neq j$. Crie n rotas de veículos 0–i–0 para i = 1,…, n
3. Ordene as economias, s_{ij} em ordem decrescente
4. Começando no topo (restante) da lista de economias, mescle as duas rotas associadas com a maior (restante) economia. Forme uma nova rota conectando (i, j) e removendo os arcos (i, 0) e (0, j) desde que:
   a) os nós i e j não estejam na mesma rota;
   b) os nós i e j são diretamente acessíveis a partir do nó 0, ou seja, nenhum dos 2 nós é interior a sua rota.
5. Repita o passo 3. até que nenhuma economia adicional possa ser alcançada.

Obs.: Ao conectar (i, j), substituímos duas arestas que incidem na origem 0, ou seja substituímos (i, 0) e (0, j) por (i, j).

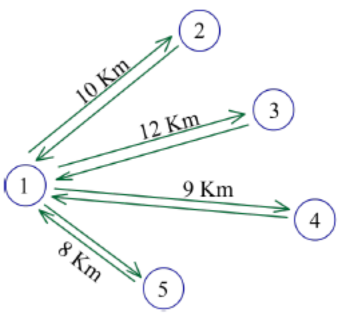
Exemplo 1



Step 1: Construct shortest distance half matrix as given below.

	2	3	4	5
1	10	12	9	8
	2	14	13	8
		3	6	7
			4	9

Step 2: Develop an initial trip allocation



Step 3: Calculate net savings for each pair of nodes as given below.

Paired Nodes	Savings sij
2,3	$S_{23} = 10 + 12 - 14 = 8$
2,4	$S_{24} = 10 + 9 - 13 = 6$
2,5	$S_{25} = 10 + 8 - 8 = 10$
3,4	$S_{34} = 12 + 9 - 6 = 15$
3,5	$S_{35} = 12 + 8 - 7 = 13$
4,5	$S_{45} = 9 + 8 - 9 = 8$

Construct net savings half matrix

	2	3	4	5
1	-	-	-	-
	2	8	6	10
		3	15	13
			4	8

Step 4: Introduce indicator I for initial round tripe allocation

	2	3	4	5
1	$I_{12} = 2$ -	$I_{13} = 2$ -	$I_{14} = 2$ -	$I_{15} = 2$ -
	2	8	6	10
		3	15	13
			4	8

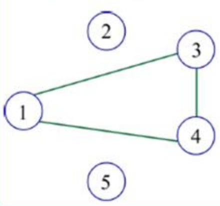
Step 5: Select the cell with maximum net savings

	2	3	4	5
1	$I_{12} = 2$ -	$I_{13} = 2$ -	$I_{14} = 2$ -	$I_{15} = 2$ -
	2	8	6	10
		3	15	13
			4	8

We can see that cell (3,4) has maximum savings so we will link (3,4) after checking conditions 5.1, 5.2 and 5.3 given in step 5 of C-W algorithm.

— I_{13} and I_{14} are greater than 0 and both have value equal to 2.
— Node 3 and node 4 are not already on the same node.
— No constraints are mentioned for this problem.

Hence, we will link nodes 3 and 4 which will change the indicator values in the net savings matrix as given below.



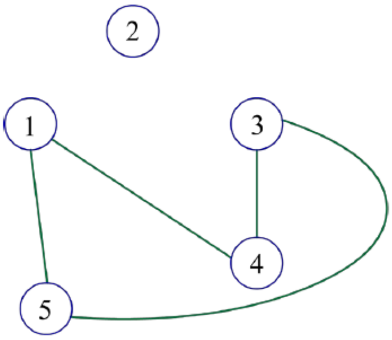
	2	3	4	5
1	$I_{12} = 2$	$I_{13} = 1$	$I_{14} = 1$	$I_{15} = 2$
	2	8	6	10
		3	$I_{34}=1$ 15	13
			4	8

The value of indicator in the cell (3,4) will become 1. The values for I_{13} and I_{14} will also change from 2 to 1.

Now look for next highest savings in the matrix, which is cell (3, 5) that is with the value of 13. Check the conditions of step 5.

— I_{13} and I_{15} are greater than 0
— Node 3 and node 5 are not already on the route

So, we will link nodes 3 and 5 with following route and update in net savings matrix.



	2	3	4	5
1	$I_{12}=2$ -	$I_{13}=0$ -	$I_{14}=1$ -	$I_{15}=1$ -
2		8	6	9
3			$I_{34}=1$ 15	$I_{35}=1$ 13
4				8

The indicator I_{15} will take value of 1 and $I_{35}=1$ will be introduced in cell (3,5). Look now $I_{13}=0$, hence we cannot link node 3 with any other node except 4 and 5.

Look for next highest savings, which are in cell (2,5).

	2	3	4	5
1	$I_{12}=2$ -	$I_{13}=0$ -	$I_{14}=1$ -	$I_{15}=1$ -
2		8	6	9
3			$I_{34}=1$ 15	$I_{35}=1$ 13
4				8

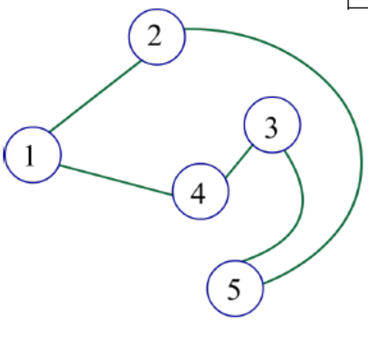
Check for the conditions in step 5.

-I_{12} and I_{15} are greater than 0

-Node 2 and node 5 are not on the route.

So, pair up node 2 and node 5 and update the network and the matrix as given below.

	2	3	4	5
1	$I_{12}=1$	$I_{13}=0$	$I_{14}=1$	$I_{15}=0$
2		8	6	$I_{25}=1$ 9
3			$I_{34}=1$ 15	$I_{35}=1$ 13
4				$I_{45}=1$ 8



Selected path or route is

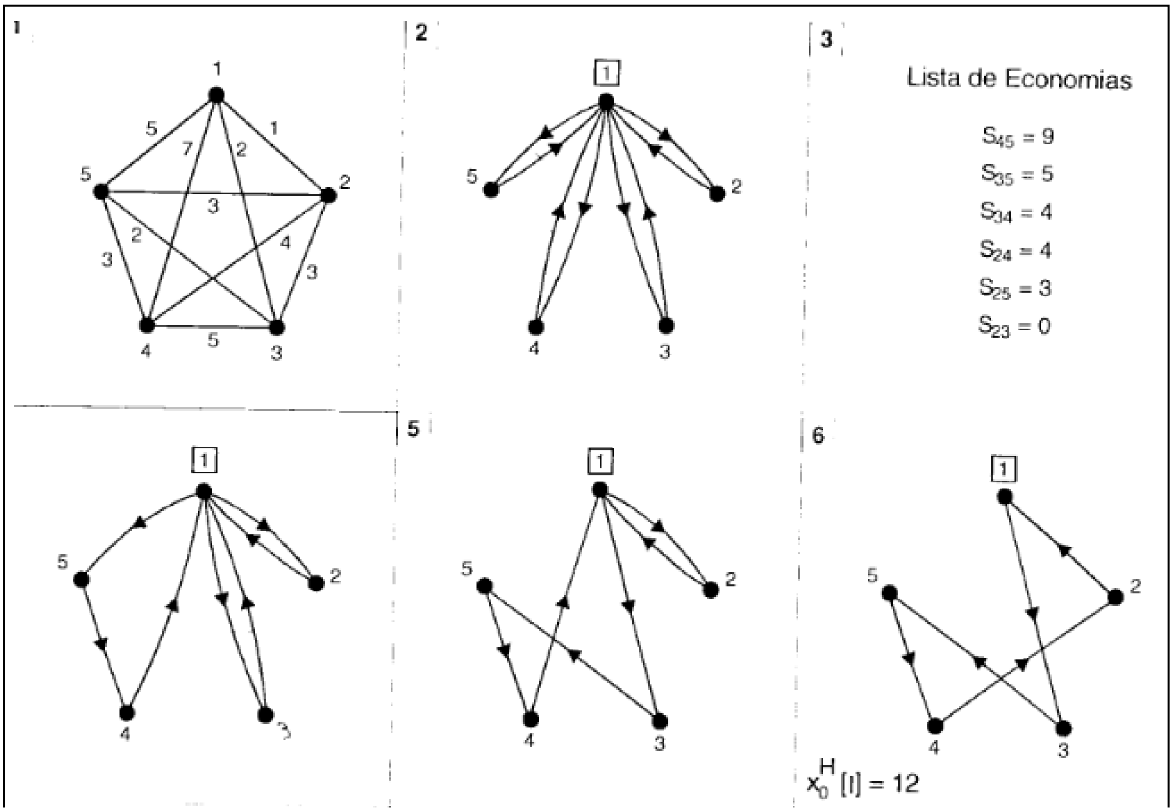
$1 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 1$

or

$1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 1$

Com distância $10 + 8 + 7 + 6 + 9 = 40$km

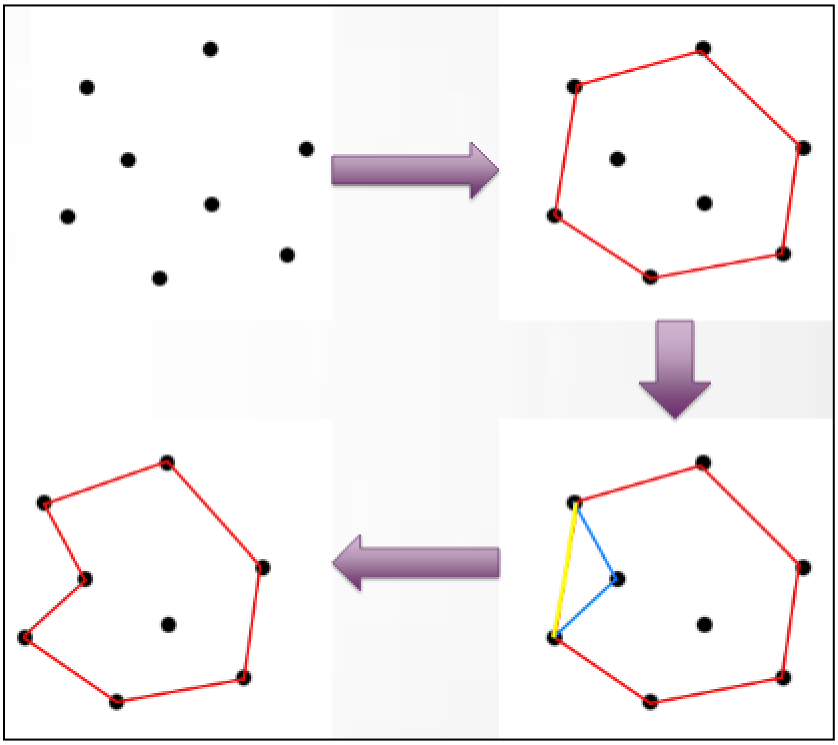
Exemplo 2



e) Heurística construtiva *Convex Hull*

Para qualquer subconjunto do plano, a envoltória convexa (*convex hull*) é o menor conjunto convexo que contém esse subconjunto. Dado um conjunto de N pontos, a envoltória convexa é a região limitada de um conjunto de N pontos, criando o menor polígono convexo.

Também conhecido como método "*Shrink Wrap*", essa heurística constrói uma rota inicial encontrando a envoltória convexa para o conjunto de nós. Então, para cada nó que ainda não está no roteiro, o algoritmo encontra o local para inserir o nó que adiciona a menor distância ao roteiro e cria uma lista contendo a melhor inserção para cada nó. Por fim, o algoritmo pega essa lista e usa uma métrica secundária para decidir qual inserção é a "melhor das melhores" e executa essa inserção. A métrica secundária pode ser qualquer coisa, incluindo menor distância.

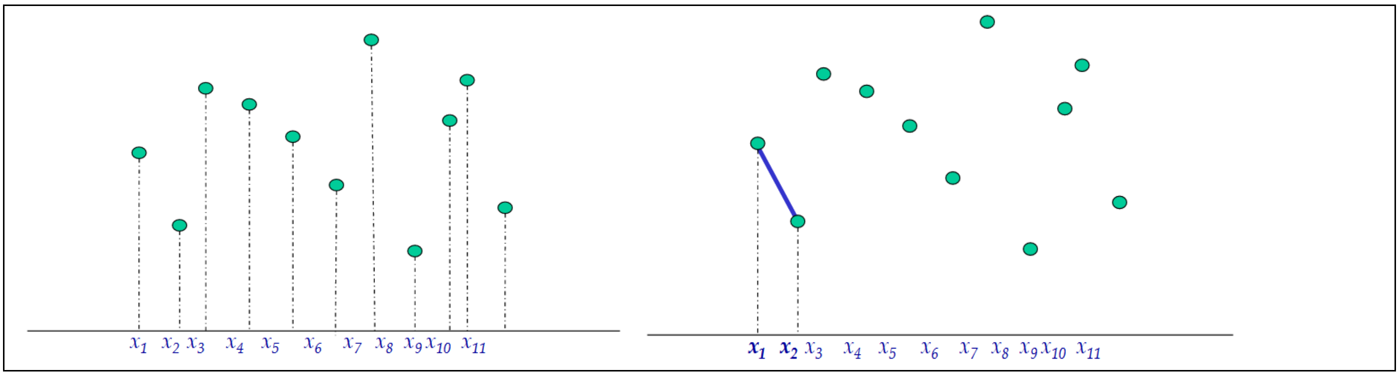


http://www.cs.utsa.edu/~korkmaz/grant/dod/posters/TourPlanning.pdf

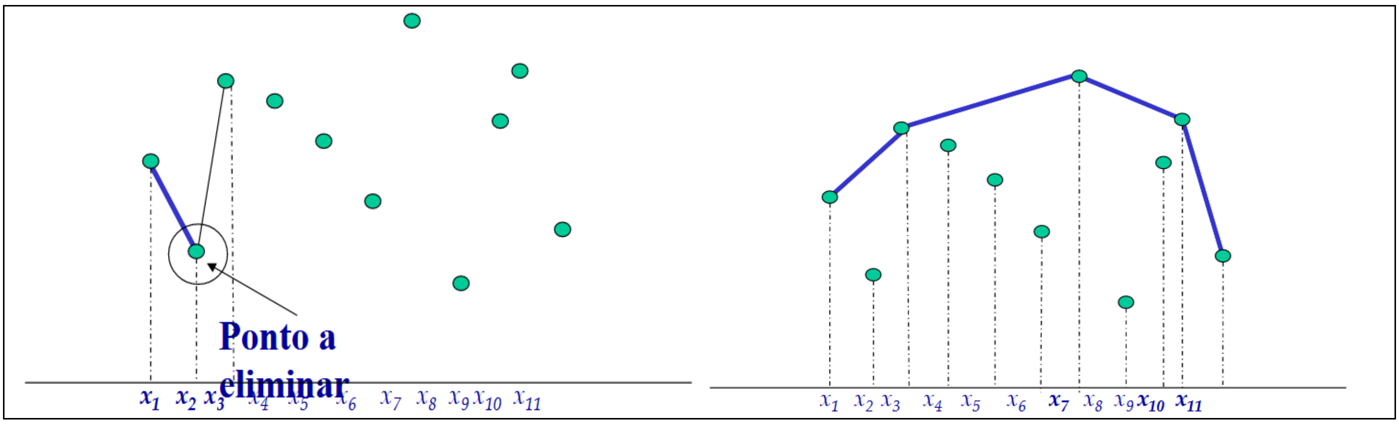
Existem diferentes métodos para construir a envoltória convexa. Abaixo, segue uma sugestão retirada do site https://inf.ufes.br/~mberger/Disciplinas/2015_2/EDII/03.pdf.

1º Ordene os pontos pelas coordenadas x

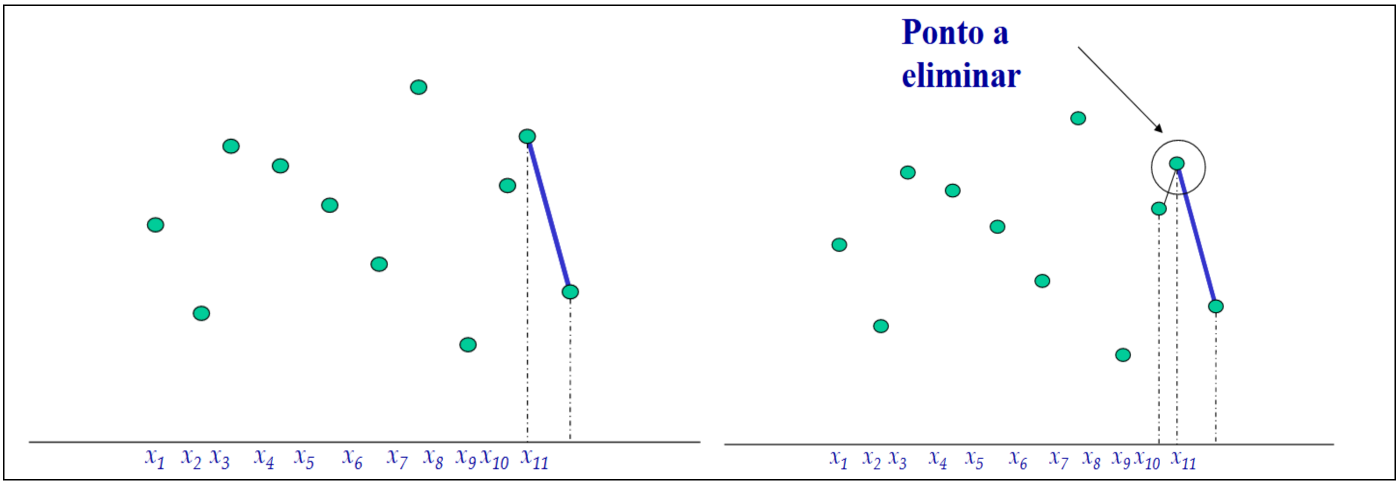
2º Construa a porção superior da envoltória convexa



Iniciamos com os 2 primeiros pontos. Adicionamos p_3 e verificamos se $p_1p_2p_3$ giram para a direita



3º. Construa a porção inferior superior da envoltória convexa



Observações:

- Para cada ponto, é determinado, se ao mover-se dos dois pontos anteriores para este ponto se forma uma "curva para esquerda" ou uma "curva para direita".
- Dado três pontos (x_1,y_1), (x_2,y_2) e (x_3,y_3), simplesmente calculando o produto vetorial dos dois vetores definidos pelos pontos (x_1,y_1), (x_2,y_2) e (x_3,y_3). Se o resultado for zero, os três pontos são colineares, se for positivo, os três pontos constituem uma "curva para esquerda", caso contrário uma "curva para direita".

O algoritmo considera cada um dos pontos na matriz ordenada em sequência. Para cada ponto, é determinado se a movimentação dos dois pontos anteriormente considerados para este ponto é uma "curva à esquerda" ou uma "curva à direita". Se for um "giro à esquerda", isso significa que o penúltimo ponto não faz parte da envoltória convexa e deve ser removido da lista. Esse processo continua enquanto o conjunto dos três últimos pontos for um "giro para a direita". Assim que uma "curva à esquerda" é encontrada, o algoritmo passa para o próximo ponto na matriz ordenada. Se em algum momento os três pontos forem colineares, pode-se optar por descartar ou incluir na lista, já que em algumas aplicações é necessário encontrar todos os pontos da fronteira (ou seja, da envoltória convexa.)

III - Métodos de melhoria das rotas

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

Os algoritmos mais utilizados são do tipo k-opt: k arcos são removidos de um roteiro e substituídos por outros k arcos. O objetivo é diminuir a distância total percorrida. Na prática são considerados 2-opt e 3-opt.

a) Heurística de melhoria Algoritmo 2-opt

No algoritmo 2-opt, elimina-se 2 arestas não adjacentes, reconecta-as usando duas outras arestas (formando um ciclo) e verifica-se se houve melhora. Este processo é repetido para todos os pares de arestas. A melhor troca (o novo ciclo com menor custo) é então realizada.

a) Remover 2 arestas da solução H obtendo uma solução H'.

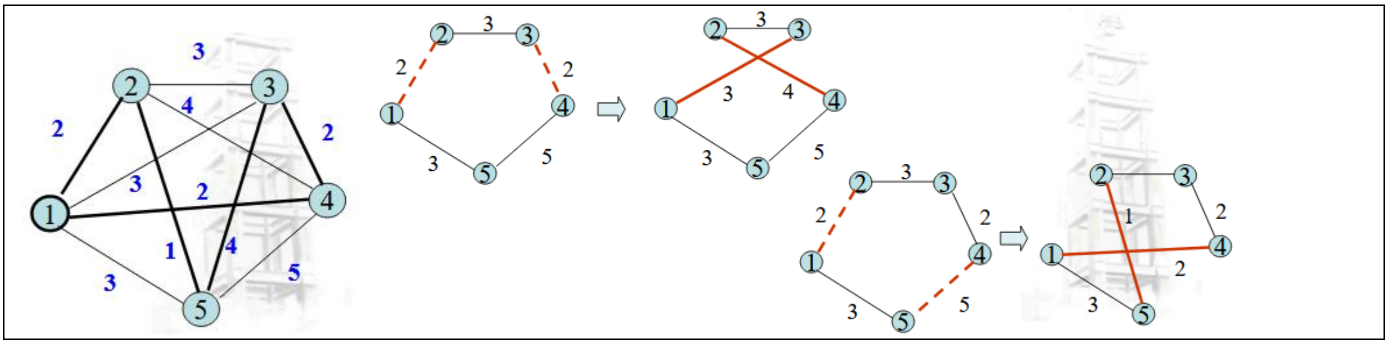
b) Construir todas as soluções viáveis contendo H'.

c) Escolher a melhor soluções dentre as encontradas e guardar.

d) Escolher outro conjunto de 2 arestas ainda não selecionado e retornar ao passo "a", caso contrário, pare.

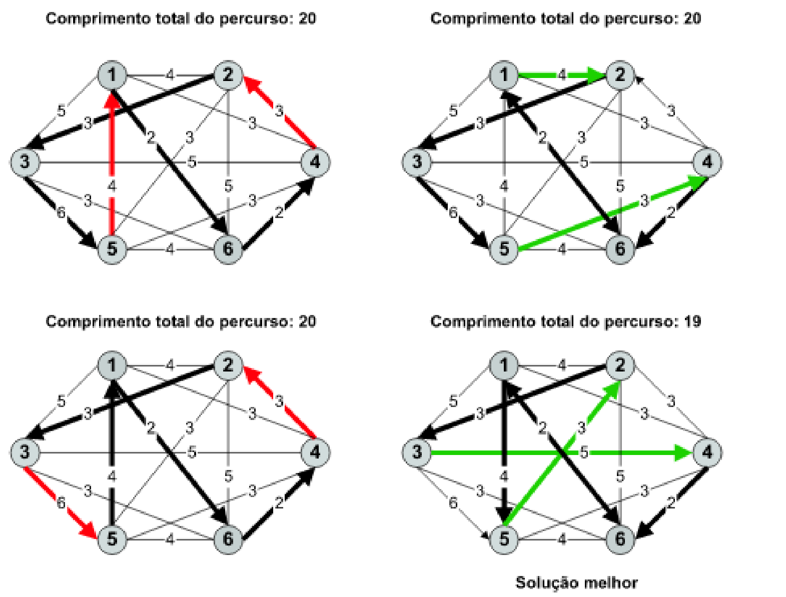
Então,entre todos os pares de arestas cuja troca 2-opt diminui o comprimento, escolhemos o par que dá o menor ciclo. Este procedimento é então iterado até que nenhum par de arestas seja encontrado.

Exemplo 1



Exemplo 2

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.)

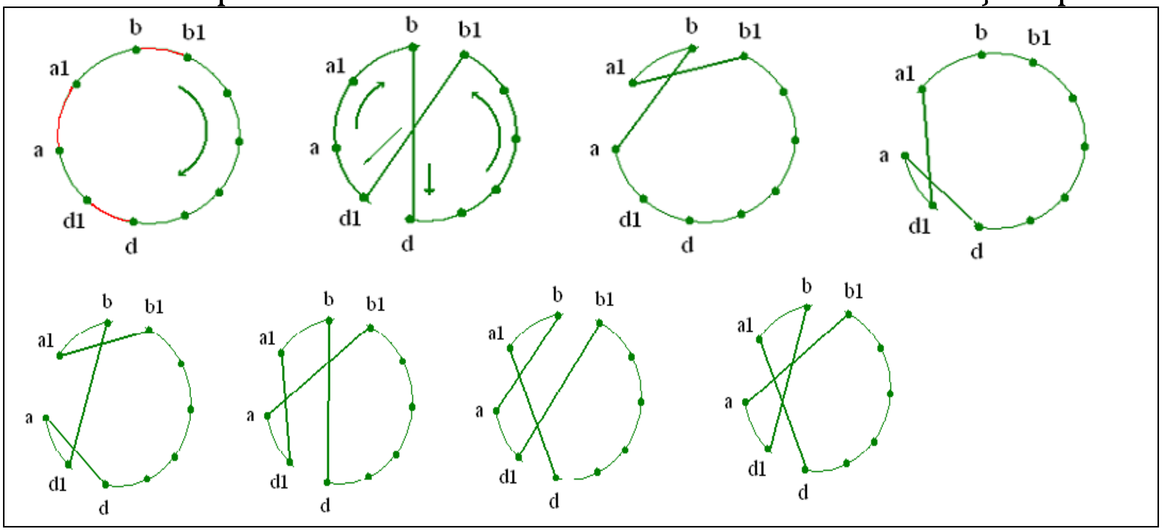
O algoritmo 2-opt basicamente remove duas arestas do ciclo e reconecta os dois caminhos criados. Isto é frequentemente referido como um movimento 2-opt. Há apenas uma maneira de reconectar os dois caminhos para que ainda tenhamos um ciclo válido. Fazemos isso apenas se o novo ciclo for mais curto. Continue removendo e reconectando o ciclo até que nenhuma melhoria 2-opt seja encontrada. O ciclo agora é 2-otimal.

Esse algoritmo tenta todas as possibilidades de combinações de arestas, mas não necessariamente retorna a melhor solução (ótima). Se olharmos para o ciclo como uma permutação de todos os nós, um movimento 2-opt resultará na reversão de um segmento da permutação. Executar a heurística 2-opt normalmente resultará em um ciclo com um comprimento menor que 5% acima do limite de Held-Karp. (Johnson D.S. & McGeoch L.A. (1995). The Traveling Salesman Problem: A Case Study in Local Optimization, November 20, 1995.)

https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/lecture-notes/MIT15_053S13_lec17.pdf

c) Heurística de melhoria Algoritmo 3-opt

Testa as trocas possíveis entre 3 arestas: Resulta em 7 combinações possíveis



Executar a heurística 3-opt normalmente resultará em um ciclo com um comprimento menor que 3% acima do limite de Held-Karp. (Johnson D.S. & McGeoch L.A. (1995). The Traveling Salesman Problem: A Case Study in Local Optimization, November 20, 1995.)

d) k-opt http://www.mafy.lut.fi/study/DiscreteOpt/DOSLID5.pdf

Valores tipicamente pequenos de parâmetros são usados: k = 2 ou k = 3. Então a mudança é relativamente fácil de executar e a vizinhança é pequena. O procedimento 3-opt demora n vezes a demora do procedimento 2-opt.

O algoritmo pode ser repetido a partir de diferentes soluções iniciais geradas aleatoriamente para melhorar a solução. Quando k é aumentado, o número de iterações aumenta, mas a solução melhora.

Mostrou-se experimentalmente que valores de k > 3 não dão vantagem adicional à qualidade da solução. Os melhores resultados são obtidos com métodos em que o valor de k é mudado dinamicamente.

Some test results http://www.cs.uu.nl/docs/vakken/an/an-travellingsalesman.pdf

In an overview paper, Junger et al report on tests on set of instances (105 – 2392 vertices; city-generated TSP benchmarks)

- *Nearest neighbor:* 24% *away from optimal in average*
- *Closest insertion:* 20%
- *Farthest insertion:* 10%
- *Cheapest insertion:* 17%
- *Random Insertion:* 11%
- *Preorder of min spanning tress:* 38%
- *Christofides:* 19% *with improvement* 11% / 10%
- *Savings method: 10% (and fast)*