PROBLEMINHAS E PROBLEMAS EM GRAFOS

PAULO JORGE M. TEIXEIRA / SAMUEL JURKIEWICZ

1. Motivação

Os grafos são fonte imensa e inesgotável de problemas teóricos ou aplicados que apresentam, em sua grande maioria, enunciados de simples entendimento, mas que, muitas vezes, escondem uma sofisticada estrutura matemática onde precisam ser modelados visto que, vez por outra, suas soluções (nem sempre exatas) exigem difíceis métodos de procura e obtenção. Neste trabalho abordamos apenas alguns aspectos de dois desses problemas e de dois de seus respectivos problemas-aplicação correlatos.

São inúmeros os subproblemas de caráter prático que podem ser modelados através das situações discutidas nos 2 (dois) problemas-chave que serão discutidos durante todo este trabalho. As diferentes aplicações desses conhecidos problemas-chave necessitam de estruturas bastante complexas para a obtenção de suas soluções mas que, muitos vezes não sendo as soluções ótimas que o problema pode apresentar, já atendem às necessidades imediatas para a sua utilização. Vamos nos limitar a levantar questões bastante pertinentes sobre os problemas e a descrever as diferentes visões para atacar cada um desses problemas, deixando as partes relativas à elaboração e à implementação de algoritmos, e a qualidade das soluções obtidas para trabalhos futuros.

2. Introdução

Quando dois ou mais grafos têm particularidades próprias entre si, dizemos que eles pertencem à mesma família de grafos.

Assim, podemos definir uma família de grafos como o conjunto (finito ou não) de todos os grafos que satisfazem a uma ou mais propriedades (as quais caracterizam essa família) que precisam ser claramente atendidas por todos os seus elementos. Normalmente, a definição de uma família consiste em mencionar a propriedade satisfeita pelos grafos que a constituem.

Como o conjunto de todos os grafos é infinito, o máximo que podemos afirmar é que um grafo pode pertencer a mais de uma família, de acordo com a caracterização a ela dada. Assim, o universo dos grafos tem sido largamente estudado

através de diferentes famílias com suas específicas características. Portanto, o grande desafio da comunidade de Teoria dos Grafos consiste em:

(1) estabelecer condições que caracterizam com precisão os elementos de cada família assim definida;

(2) encontrar condições que permitam determinar o total de elementos de cada família; e

(3) definir a maneira de ser possível construir grafos que atendam às propriedades estabelecidas para cada uma das famílias.

2.1. **Famílias de Grafos.**

Entre essas famílias podemos destacar:

- **a família dos grafos planares** (aqueles em que sua representação no plano seja de tal modo que quaisquer duas arestas não se cruzem, ou seja, possam até ter um vértice em comum).
- **a família das árvores** (aqueles grafos que são conexos e acíclicos com número mínimo de arestas);
- **a família dos grafos isomorfos** (aqueles grafos em que há uma bijeção entre os conjuntos de vértices de cada um deles e tais que as adjacências entre as arestas de ambos são preservadas);
- **a família dos grafos eulerianos** (aqueles grafos em que é possível verificar a existência de um circuito que inclua exatamente uma vez cada aresta);

  Problema-aplicação correlato: O Problema do Carteiro Chinês.

  Como tratar os circuitos eulerianos? Veremos no parágrafo 5.
- a família dos grafos hamiltonianos (aqueles grafos em que é possível verificar a existência de um circuito que inclua exatamente uma vez cada vértice);

Problema-aplicação correlato: O Problema do Caixeiro Viajante.

Como tratar os circuitos hamiltonianos? Veremos no parágrafo 6.

Há grafos que podem pertencer a mais de uma família. Por exemplo, os grafos da família de ciclos são eulerianos e hamiltonianos. Há outros grafos que são eulerianos e não são hamiltonianos. Há outros que são hamiltonianos mas não são eulerianos, e há outros que não são eulerianos e também não são hamiltonianos.

Embora os grafos pertencentes às duas famílias anteriores sejam aparentemente semelhantes, há algumas diferenças fundamentais entre eles.

Ou seja, se o problema é: **dado um grafo, verificar se existe um circuito euleriano ou um circuito hamiltoniano**,estamos diante de dois problemas cujas soluções são de **graus de dificuldade bastante diferentes**.

Essas duas últimas famílias fazem parte do objeto fundamental de discussão dos problemas que iremos abordar neste trabalho.

3. Conceitos Iniciais da Teoria de Grafos

3.1. **Representação de Grafos.**

Um esquema de representação é uma maneira alternativa de exibir os grafos de uma determinada família.

O resultado obtido pela aplicação de um esquema a um grafo $G = (V, E)$, onde $|V| = n$ e $|E| = m = \{(u, v)/u \in V, v \in V\}$ é chamado representação para o grafo. É preciso explicitar n vértices e m pares de vértices. A partir dela, os conjuntos de vértices e arestas do grafo representado devem poder ser deduzidos sem ambigüidades.

3.1.1. ***Esquema 1.***

O esquema de representação mais usual para grafos arbitrários é o **gráfico**, onde:

- todo grafo pode ser representado sobre uma superfície por um conjunto de pontos e segmentos de curvas de tal modo que: a cada vértice corresponde um ponto e a cada aresta corresponde um segmento de curva interligando os pontos associados às suas extremidades.
- essa representação gráfica não é única.

3.1.2. ***Esquema 2.***

Outro esquema consiste em mencionar, para todo vértice $v \in V$, seu conjunto de vizinhos $N(v)$, construindo um conjunto de pares da forma $\{(v, \{N(v)\})/v \in V\}$. Numa representação obtida por esse esquema são mencionados $\sum_{v \in V} [1 + |N(v)|] = n + 2m$ termos.

Exemplo 3.1.2.1 - Subfamília dos grafos conexos cujos vértices possuem grau 2. Uma caracterização equivalente seria dizer que os grafos integrantes desta família são ciclos simples sem arestas interiores (cordas). Logo, grafos

com n vértices desta família possuem exatamente n arestas. O esquema de representação por vizinhos exige a nomeação de $n + 2n = 3n$ elementos. Por exemplo, considere o ciclo e seu esquema de representação apresentados abaixo:

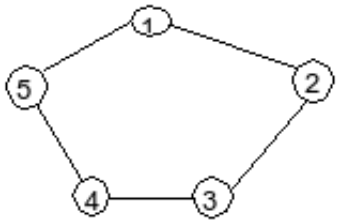


Figura 2: Um ciclo simples com 5 vértices e seu esquema de representação por vizinhos

3.1.3. ***Esquema 3.***

Um esquema alternativo mais compacto para representar tais grafos é utilizar uma seqüência de n vértices, dispostos na mesma ordem em que figuram no ciclo: $<< 1, 2, 3, 4, 5 >>$. Esta representação não é única: o mesmo grafo poderia ser representado por $<< 1, 5, 4, 3, 2 >>$ ou $<< 2, 3, 4, 5, 1 >>$, por exemplo.

3.2. **Considerações teóricas.**

Um grafo não orientado, ou simplesmente grafo G é um par (V, E), onde V é o conjunto de vértices e E é o conjunto de arestas. Indicaremos por $|V| = n$ e $|E| = m$ a quantidade de elementos de cada conjunto. Cada aresta é um subconjunto de V com 1(um) ou 2 (dois) vértices. Indicaremos uma aresta e por $\{e\}$ ou, quando for necessário nomear os vértices extremidades da aresta indicamos por $\{u, v\} \in E$ ou $e = \{u, v\} \in E$. Diz-se também que a aresta incide sobre os vértices v e w.

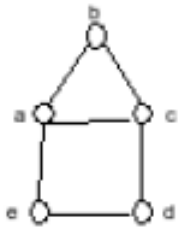
Uma aresta unitária, da forma $\{v\}$, é dito um laço. O grau de um vértice é o número de arestas que nele incidem. No caso de um laço, a aresta unitária é ?contada? duas vezes quando da determinação do grau do vértice e, portanto, contribui com 2(duas) unidades para o seu grau. Um grafo é dito regular se todos os seus vértices possuem o mesmo grau. Em particular, se todos possuem grau k ele é dito k-regular.

Um multigrafo é um grafo em que há pelo menos um par de vértices que tem pelo menos duas arestas adjacentes entre si. Neste trabalho, os grafos considerados são finitos (possuem um número finito de vértices) e não possuem laços. Dois vértices são adjacentes ou vizinhos quando são extremidades de uma aresta. Um grafo é dito completo quando todo vértice é vizinho dos demais

vértices. Uma clique em um grafo G com n vértices é um subgrafo completo de G com k vértices ($k < n$). O conjunto de vizinhos de um vértice v é o conjunto $N(v) = \{w \in V/\{v, w\} \in E\}$. A cardinalidade $|N(v)|$ é dita o grau de v, e indicado por $d(v)$. O grau mínimo de um grafo G é indicado por δ é o menor dos graus dentre todos os vértices de G. O grau máximo de um grafo G é indicado por Δ.

Um grafo H é dito um subgrafo de um grafo G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Se H é um subgrafo de G e $V(H) \subseteq V(G)$ e $V(H)?V(G)$ então H é dito um subgrafo próprio de G. Um subgrafo H de G é dito um subgrafo gerador ou de espalhamento de G se $V(H) = V(G)$.

Seja $G = (V, E)$, onde $V = \{a, b, c, d, e\}$ e $E = \{(a, b), (b, c), (a, c), (c, d), (d, e), (a,$



No exemplo acima, seja H um subgrafo de G tal que $V(H) = \{a, b, c, d\}$ e $E(H) = \{\{a, b\}, \{c, d\}\}$

Em particular, um subgrafo H de G é dito induzido se, para qualquer par de vértices v e w de $V(H)$, se existe $\{v, w\} \in E(G)$, então existe $\{v, w\} \in E(H)$. Seja H um subgrafo induzido de G tal que $V(H) = \{a, b, c, d\}$ e $E(H) = \{\{v, w\} \in E(G)/v \in V(H) e w \in V(H)\}$. Podemos dizer que H é um subgrafo induzido por $V(H)$ em G e denota-se por $H = G[V(H)]$.

Exemplo: Seja $A \subseteq V(G)$ tal que $A = \{b, e, d\}$ e considere $G[A]$ (subgrafo de G induzido por A).

Se H é um subgrafo de G e $V(H) \subseteq V(G)$ e $V(H)?V(G)$ então H é dito um subgrafo próprio de G. Um subgrafo H de G é dito um subgrafo gerador ou de espalhamento de G se $V(H) = V(G)$.

A excentricidade de um vértice v de um grafo G, denotada por $e(v)$, é a maior das distâncias do vértice v aos demais vértices do grafo G, isto é: $e(v) = \max\{d(v, w)/w \in V(G), w \neq v\}$.

O diâmetro de um grafo G, denotado por *diâmetro(G)*, é o maior valor dentre todas as excentricidades dos vértices de G, isto é: *diâmetro(G)* = $\max\{e(v)/v \in V(G)\}$.

O raio de um grafo G, denotado por *raio(G)*, é o menor valor dentre todas as excentricidades dos vértices de G, isto é: *raio(G)* = $\min\{e(v)/v \in V(G)\}$.

O centro de um grafo G, denotado por $C(G)$, é o conjunto de vértices do grafo G que têm a menor excentricidade, isto é: $C(G) = \{v \in V(G)/e(v)$é mínimo$\}$. Portanto, em vista da definição, tem-se que o centro de um grafo não é formado, necessariamente, por um único vértice.

Dado um grafo $G = (V, E)$, um passeio em G é uma seqüência de vértices de G: $v_1, v_2, \ldots, v_k$ tais que toda aresta $(v_i, v_{i+1}) \in E, \forall i, 1 \leq i \leq k-1$. O comprimento do passeio é dado pelo número de arestas que o compõem, isto é, $k-1$ como indicado anteriormente. O vértice $v1$ é chamado de origem (ou início) do passeio e o vértice v_k seu término (ou fim). Um passeio em que não há repetição de arestas, isto é: todas as arestas do passeio são distintas, é dito uma trilha ou trajeto. Em particular, se não há repetição de vértices, o passeio é dito um caminho. Um passeio $v_1, v_2, \ldots, v_k$ é dito fechado quando o vértice de seu início coincide com o vértice de seu término, isto é: $v_1 = v_k$. Analogamente, uma trilha ou um caminho são ditos fechados quando o vértice de início coincide com o vértice final. Um caminho é dito hamiltoniano se é composto por uma seqüência de arestas que percorre os vértices do grafo uma e somente uma única vez.

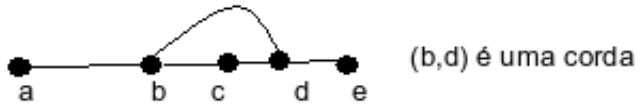
Uma trilha é dita hamiltoniana se é um caminho hamiltoniano que, partindo de um qualquer vértice de um grafo G conexo, retorna ao vértice inicial de tal modo que, exceto para o vértice inicial, foi percorrido um caminho hamiltoniano. Assim, fica formado um circuito com todos os vértices do grafo G e, portanto, todos os vértices desse circuito têm grau igual a 2(dois). Um ciclo é um passeio fechado $v_1, v_2, \ldots, v_{k-1}, v_k, v_1$ tal que $v_1, v_2, \ldots, v_k$ é um caminho. Ou seja, um ciclo é um caminho de comprimento maior do que ou igual a 3, em que o primeiro e o último vértices coincidem. Um ciclo simples é um caminho de comprimento maior do que ou igual a 3 em que somente o primeiro e o último vértices coincidem. O tamanho de um passeio fechado é dado pelo número de arestas que o compõem. Em particular, o comprimento de um ciclo é dado pelo seu número de vértices distintos ou, equivalentemente, de seu número de arestas. Um ciclo com um número par de arestas é dito um ciclo par, e com um número ímpar de arestas é dito um ciclo ímpar. Um grafo que não possui ciclos

é dito **acíclico**. Um grafo é **conexo** quando existir pelo menos um caminho entre cada par de vértices. Caso contrário é dito **desconexo**.

Uma árvore é um grafo conexo e acíclico.

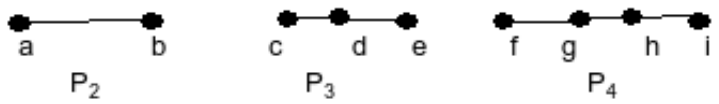
Uma corda em um ciclo é uma aresta que é adjacente a dois vértices não consecutivos desse ciclo. Uma corda em um caminho é uma aresta que é adjacente a dois vértices não consecutivos de um caminho.

Exemplo:



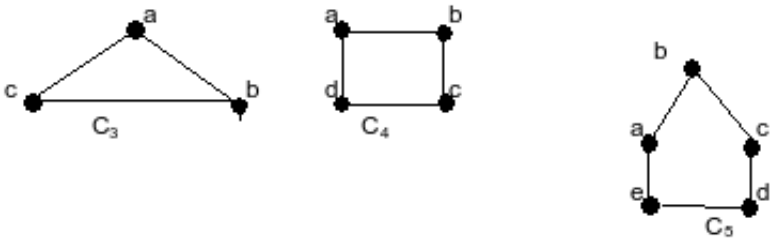
Um grafo $G = (V, E)$ é dito um caminho se for um grafo induzido por um caminho sem cordas. Um grafo caminho com n vértices será denotado por P_n , com $n \geq 2$.

Exemplo:

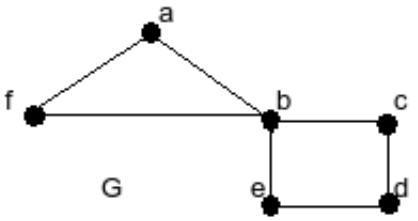


Um grafo ciclo é um grafo induzido por um ciclo sem cordas. Um grafo ciclo com n vértices será denotado por C_n.

Exemplo:



Exemplo:



Tem-se:

- $abcdebaf$ é um passeio do vértice a até o vértice f , mas não é trilha;
- $abcdebafa$ é um passeio fechado;
- $abcdebfa$ é uma trilha fechada;
- $abcdebf$ é uma trilha de vértice a ao vértice f, mas não é caminho;
- caminhos entre os vértices a e f: abf ou af;

- $abfa$ é um ciclo de tamanho 3 e $bcdeb$ é um ciclo de tamanho 4.

Um emparelhamento em um grafo G é um conjunto de arestas de G duas a duas não adjacentes. Um emparelhamento em G é dito um emparelhamento perfeito de G se cada vértice de G está presente em uma única aresta do emparelhamento.

Teorema 3.2.1 (**do aperto de mãos**). *Seja $G = (V, E)$ grafo tal que $|V| = n$ e $|E| = m$. Então $\sum_{v \in V} d(v) = 2m$.*

Demonstração. Cada aresta (v, w) contribui com 2(duas) unidades para a soma dos graus dos vértices (uma unidade para $d(v)$ e uma unidade para $d(w)$). Logo, a soma total é duas vezes o número de arestas. □

Observação Esse teorema é também válido para multigrafos.

Corolário 3.2.2. *Em qualquer grafo G, a soma dos graus de seus vértices é par.*

Demonstração. É imediato, pois $\sum_{v \in V} d(v) = 2m$, e $2m$ é par. □

Corolário 3.2.3. *Em qualquer grafo $G = (V, E)$, o número de vértices de grau ímpar é par.*

Demonstração. Sejam $V_P = \{v \in V / d(v) \text{ é par}\}$ e $V_I = \{v \in V / d(v) \text{ é ímpar}\}$. É claro que $V = V_P \cup V_I$ e $V_P \cap V_I = \varnothing$. Além disso: $2m = \sum_{v \in V} d(v) = \sum_{v \in V_P \cup V_I} d(v) = \sum_{v \in V_P} d(v) + \sum_{v \in V_I} d(v)$. Logo, o número de parcelas de grau ímpar (ou o número de vértices de grau ímpar) é par. □

4. Problemas NP-Completos

De um modo geral um problema algorítmico pode ser caracterizado por um conjunto de todos os possíveis dados do problema, denominado conjunto de dados e por uma questão solicitada, denominada objetivo do problema. Resolver o problema P consiste em desenvolver um algoritmo cuja entrada é composta pelos dados específicos do problema (retirados desse conjunto) e sua saída, denominada **solução**, resposta ao objetivo do problema. Os dados específicos que constituem uma entrada fornecem o que chamamos de uma **instância do**

problema. Ou seja: um problema possui tantas instâncias diferentes quantas são as variações possíveis de seus dados.

Por exemplo, considere as seguintes situações:

Dados: Um grafo G e um inteiro $k > 0$.

Objetivo: Elaborar um algoritmo para encontrar em G, se existir, uma clique de tamanho maior do que k.

Conjunto de dados: conjunto de todos os pontos (G, k) onde G é grafo arbitrário e k um inteiro positivo arbitrário.

Instância do problema: Um par específico (G, k).

Solução: Um subgrafo completo de G com k ou mais vértices, se existir, é uma solução do problema.

Podemos dividir os problemas algorítmicos nas seguintes classes de problemas :

- **Problemas de Decisão:**
  Existe uma estrutura S que satisfaça às propriedades do problema P?
  Objetivo: decidir pela resposta Sim ou Não, à questão acima;
- **Problemas de Localização:**
  Encontrar uma estrutura S que satisfaça às propriedades de S?
  Objetivo: localizar certa estrutura S que satisfaça à um conjunto de propriedades dadas;
- **Problemas de Otimização:**
  Encontrar uma estrutura S que satisfaça à certo(s) critério(s) de otimização.
  Objetivo: verificar se as propriedades a que S deve satisfazer envolvem critérios de otimização.

Tomemos como exemplo o PCV (Problema do Caixeiro Viajante):

Decisão: Um grafo completo G valorado em suas arestas e um inteiro $k > 0$. Verificar se o grafo G possui um circuito hamiltoniano C (de Caixeiro Viajante) de peso menor do que ou igual k: $p(C) \leq k$.

Localização: Localizar, em G, um percurso de Caixeiro Viajante, de peso menor do que ou igual a k.

Otimização: Localizar, em G, um percurso de caixeiro viajante ótimo (de peso mínimo).

Um algoritmo natural para solucionar este problema precisa calcular o peso de $\frac{(n-1)!}{2}$ circuitos hamiltonianos de G. A complexidade de tal algoritmo é de ordem fatorial e, dessa forma, é muito ineficiente. O PCV está na classe de complexidades NP, isto é, dado um grafo G valorado em suas arestas, uma

constante k e uma seqüência de n vértices $v_1, v_2, \ldots, v_n$, podemos verificar se esta seqüência de vértices é um circuito hamiltoniano em G, com peso no máximo igual a k, em tempo polinomial e o problema é pelo menos tão difícil de ser solucionado quanto qualquer outro problema da classe NP. Um problema que satisfaz a essa segunda condição também é dito ser NP-difícil.

Definimos a classe P de problemas de decisão como aquela classe que compreende aqueles problemas que admitem algoritmo polinomial para obter a solução.

Define-se a classe NP como sendo aquela que compreende todos os problemas de decisão P tais que existe uma justificativa à resposta SIM para P, cujo passo de reconhecimento pode ser realizado por um algoritmo polinomial no tamanho de entrada de P.

Exemplos de problemas da classe NP, cujos algoritmos desenvolvidos são exponenciais:

(1) Dados: Um grafo G e um inteiro $k > 0$.
Decisão: G possui um conjunto independente de vértices de tamanho igual a k?
(2) Dados: Um grafo G e um inteiro $k > 0$.
Decisão: G possui uma clique de tamanho igual a k.
(3) Dados: Um grafo G e um inteiro $k > 0$.
Decisão: G possui uma cobertura de vértices de tamanho igual a k.
(4) Dados: Um grafo G orientado.
Decisão: G possui um ciclo hamiltoniano direcionado?
(5) Dados: Um grafo G não orientado
Decisão: G possui um ciclo hamiltoniano?
(6) Dados: Um grafo G e um inteiro $k > 0$.
Decisão: G possui uma coloração com um número menor que k de cores?
(7) Dados: Dados os grafos G_1 e G_2
Decisão: O grafo G_1 contém um subgrafo isomorfo a G_2?
(8) Dados: um grafo $G(V, E)$ e um inteiro $k > 0$.
Decisão: A clique de tamanho máximo de G possui k vértices?

Um algoritmo é dito eficiente precisamente quando a sua complexidade for uma função polinomial nos tamanhos dos dados de sua entrada. Um algoritmo é dito exponencial quando sua complexidade for uma função exponencial nos tamanhos dos dados de sua entrada.

Considere a coleção de todos os algoritmos que resolvem um certo problema P. Será que nessa coleção existe algum que seja eficiente? Isto é, de complexidade polinomial?

Se existir tal algoritmo que o resolva, o problema P será denominado ***tratável*** e, caso contrário, ***intratável***. Assim, a idéia é a de que um problema tratável possa ser resolvido para entradas e saídas de tamanho razoável através de algum processo automático.

Enquanto isso, um algoritmo de complexidade não polinomial, de algum problema intratável poderia, em certos casos, levar séculos para computar dados de entrada e saída com tamanhos relativamente reduzidos.

Para verificar se um problema P é intratável há necessidade de provar que todo possível algoritmo que o resolva não possui complexidade polinomial.

A classe de problemas P é aquela em que para seus problemas são conhecidos algoritmos de complexidade polinomial capazes de resolvê-los. Será que, se todos os algoritmos conhecidos para resolver certo problema de decisão $\wp$ exibirem complexidade exponencial, é possível garantir que $\wp \notin P$? Um exemplo é uma tautologia, onde não se sabe se algum deles pertence à classe P. Então, uma tautologia está na classe NP-completo, segundo Karp (1972). Se algum dos problemas pertence à classe P, então, todos os demais em NP pertencerão à P, resultando P = NP.

P = NP ou P≠NP. Nada se sabe até agora.

Conjectura: a classe máxima não pertence a NP.

Lema: P≠NP.

P = NP? Ou seja: existe algum problema na classe NP que seja intratável? Ou, caso contrário, todo problema em NP admite necessariamente algoritmo polinomial? Até o momento não se conhece resposta a essa indagação. Todas as evidências apontam, na direção P≠NP.

Um argumento: a classe NP incorpora um grande número de problemas para os quais inúmeros pesquisadores já se debruçaram para elaborar algoritmos eficientes, mas que, apesar disso, não foi possível a formulação de algoritmos polinomiais para quaisquer deles. Os problemas pertenceriam então à classe NP?

Alguns problemas da classe NP podem ser reduzidos a outros problemas da mesma classe por uma transformação realizada em tempo polinomial. Assim, um problema P* se reduz a um problema P se existe uma função computável

em tempo polinomial f tal que $f : P^* \to P$ tal que $f(I^*) = I$. Nota-se como: $P^* \propto P$. Esses problemas estão na classe NP-árduo.

Definição: P é NP-árduo se e somente se qualquer que seja o problema P*, se $P^* \notin NP$ então $P^* \propto P$.

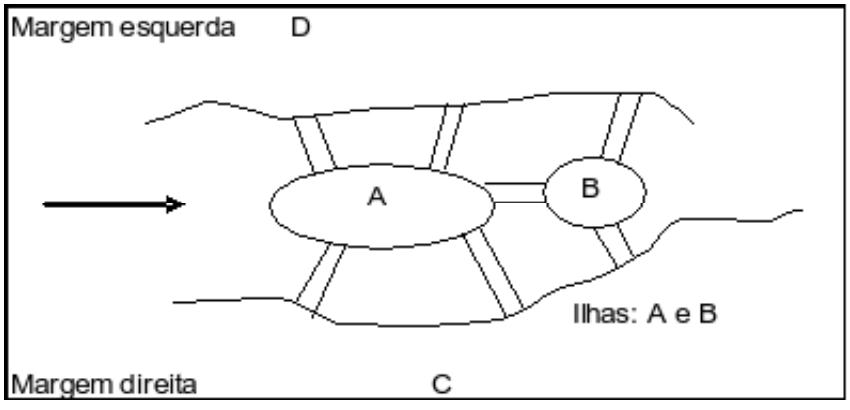
Assim, se P é NP-árduo então P é pelo menos tão difícil quanto qualquer problema P*, $P^* \notin NP$.

Um problema de decisão P é NP completo se e somente se P é NP-árduo e $P \notin NP$. O Problema da Parada é indecidível e é um problema de decisão NP-árduo que não é NP completo.

5. Grafos Eulerianos

5.1. **Histórico.**

O problema foi estudado pela primeira vez por Euler (1707-1783) e, por essa razão um circuito que percorre cada aresta de um grafo exatamente uma vez é dito um circuito euleriano e todo grafo que possui tal circuito é dito um **grafo euleriano**. A situação estudada por Euler é conhecida como o ***Problema das Pontes de Könisberg*** (hoje chamada de Kaliningrado), e está ilustrado na Figura 1, abaixo. O objetivo, na ocasião, era saber se é possível, saindo de qualquer dos lugares A, B, C ou D, percorrer, exatamente uma vez, todas as sete pontes da cidade (a que conecta as duas ilhas A e B entre si e as seis que conectam as ilhas com as margens C e D do rio), e voltar ao ponto de partida.



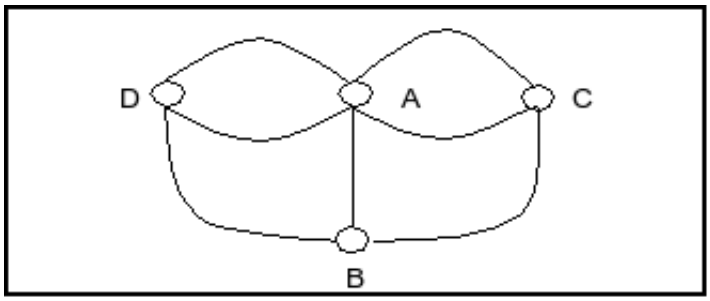


Figura 1 - O Problema das Pontes de Könisberg

Na linguagem da Teoria dos Grafos, trata-se de encontrar, se possível, um circuito euleriano no multigrafo da figura acima, no qual os vértices A, B, C, e D representam as ilhas e as margens, e as arestas são as pontes. Euler mostrou que tal circuito não existe através de um argumento extremamente simples, a saber:

Considere, por exemplo, a ilha B. Um circuito qualquer deve chegar à ilha e sair dela o mesmo número de vezes. Logo, para que exista um circuito euleriano, deve haver um número par de pontes com extremidade nesta ilha. Como na ilha B existem três pontes nessas condições, concluímos que não é possível encontrar um circuito euleriano. De modo mais geral tem-se o seguinte teorema:

Teorema 5.1.1. *Um grafo* ***conexo*** *possui um* ***circuito euleriano*** *se e somente se cada vértice tem* ***grau*** *par.*

Demonstração. ($\Rightarrow$) Seja $G = (V, E)$ um grafo conexo euleriano. Logo G possui uma trilha euleriana fechada C com origem e término em um vértice v. Toda vez que se toma um vértice u interno à trilha C duas arestas com vértice terminal u são utilizadas por C. Como C contém todas as arestas do grafo G ele também contém todas as arestas adjacentes ao vértice u. Assim, o grau do vértice u é par, qualquer que seja o vértice u diferente do vértice v. Por outro lado, a trilha C começa e termina no vértice v e, portanto, o grau do vértice v também é par. Portanto, concluímos que qualquer vértice de G tem grau par. ($\Leftarrow$) Suponha que G é grafo conexo e tal que todos os seus vértices têm grau par. Vamos mostrar que G é euleriano por indução na quantidade de suas arestas. Se $|E| = 0$, então G é grafo nulo (só possui vértices) e, portanto, é euleriano no sentido de que não há arestas a serem atravessadas. Vamos supor, então, que G possui pelo menos uma aresta. Como G é conexo e todo vértice tem grau par, o grau de cada vértice é pelo menos igual a dois, ou seja, o grafo G possui pelo menos uma trilha fechada. Seja $C = (V, E^*)$ a trilha fechada de G com maior comprimento. Vamos então mostrar que a trilha C contém todas as arestas de G $(E = E^*)$. Para isso, suponha, por absurdo, que C

não contém todas as arestas de G e seja $\{e\}$ uma aresta que pertence a G mas não está na trilha C, isto é: $\{e\} \in (E(G) - E(C) = E(G) - E^*)$. Seja G^* a componente conexa de G que possui todas as arestas de $E(G) - E^*$, ou seja, que contém a aresta $\{e\}$. Como C é uma trilha fechada, todo vértice de G^* tem grau par. Assim, todo vértice de G^* tem grau par. Por hipótese de indução, G^* é grafo euleriano. Seja C^* uma trilha euleriana em G^*. Como G é conexo, existe pelos menos um vértice v em $V(C) \cap V(C^*)$. Assim, $C \cup C^*$ é uma trilha fechada em G e de comprimento maior do que o comprimento de C, o que contraria a escolha de C como sendo de maior comprimento. Daí, a trilha C contém todas as arestas de G e, portanto, G é grafo euleriano. □

O Teorema acima mostra a necessidade de se ter grau par em cada vértice para existir um circuito euleriano. É também natural a necessidade do grafo ser conexo.

Corolário 5.1.2. *Um grafo conexo tem uma trilha euleriana se e somente se ele possui, no máximo, dois vértices de grau ímpar.*

Demonstração. ($\Rightarrow$) Seja G um grafo conexo e tal que G possua uma trilha euleriana. Assim, cada vértice desta trilha, distinto do vértice inicial e do vértice final, tem grau par. Portanto, G tem, no máximo, dois vértices de grau ímpar.
($\Leftarrow$) Suponha G um grafo que possui, no máximo, dois vértices de grau ímpar. Assim, G poderá possuir 0(zero) ou 2(dois) vértices de grau ímpar (conforme Corolário 3.2.3).

(1) Se G não possui vértices de grau ímpar, então, pelo Teorema 5.1.1, G tem uma trilha euleriana fechada;
(2) Se G possui dois vértices de grau ímpar, digamos, os vértices u e v, vamos considerar a aresta $e = \{u, v\}$ e tal que o grafo $G + \{e\}$ é obtido de G pela adição dessa aresta ligando os vértices u e v entre si. Então, todo vértice de $G + \{e\}$ tem grau par (pois G possui todos os demais vértices com grau par), e, novamente pelo Teorema 5.1.1, $G + \{e\}$ tem uma trilha euleriana fechada.

Portanto, de (1) e (2), G tem uma trilha euleriana fechada com vértice inicial em u e vértice final v. □

5.2. **O Problema do Carteiro Chinês.**

Suponha que um carteiro deva entregar correspondências e/ou entregas em domicílios localizados em parte das residências de um bairro ou de uma cidade. É aconselhável que antes de sair do setor central de distribuição da empresa, o carteiro possa estabelecer o roteiro a ser feito de modo a realizar o percurso necessário à cumprir os compromissos da forma mais eficiente possível, não só entregando corretamente as correspondências a seus destinatários como também minimizando o percurso a ser feito, de modo a retornar à empresa o mais rapidamente possível. Assim, caso haja correspondência em ambos os lados de diferentes ruas ou trechos dessas, é recomendável o planejamento do percurso que atravessa todo o trecho de uma rua entre duas esquinas consecutivas, pelo menos uma vez, minimizando a quantidade de vezes em que ele atravessa uma rua em ziguezague, caso necessário, para a entrega da correspondência.

O problema de obtenção de um percurso como este, é chamado o **"Problema do Carteiro Chinês"**, devido ao fato do matemático chinês Guan, em 1962, ter sido o primeiro que o formulou.

Em Teoria dos Grafos podemos modelar esta situação num grafo, fazendo com que os cruzamentos entre as ruas correspondam aos vértices do grafo e de tal modo que dois vértices são adjacentes se há uma seção de rua (calçadas e residências) que passa pelos dois cruzamentos correspondentes aos respectivos vértices, como a seguir:

Dado um grafo conexo G, o Problema do Carteiro Chinês consiste em determinar um passeio fechado (tour) nesse grafo, de comprimento mínimo, passando pelo menos uma vez por cada aresta e por todas as arestas de G. Ora, tal passeio fechado que contenha todas as arestas de G e de comprimento mínimo, nada mais é que a obtenção de um passeio euleriano de custo mínimo em G. Dizemos que um tal passeio é um passeio ótimo. Já sabemos que se G é grafo euleriano então um passeio euleriano de G é uma trilha fechada euleriana, ou seja, qualquer trilha fechada euleriana é um tour ótimo, já que passa exatamente uma única vez por cada aresta do grafo. Assim, O Problema do Carteiro Chinês é facilmente resolvido em grafos que, a priori, sabe-se serem grafos eulerianos. Basta, para tal, encontrar-se uma trilha euleriana fechada.

No caso do grafo G ser euleriano, já há o Algoritmo de Fleury, bastante simples, que determina uma trilha fechada euleriana (trilha fechada que atravessa cada aresta uma única vez), e, então, o Problema do Carteiro Chinês é facilmente resolvido (uma vez que qualquer trilha fechada euleriana de G é uma

trilha ótima (solução ótima sem se considerar os custos das arestas)). O Algoritmo de Fleury, devido a Fleury, 1921 (veja Lucas) constrói a trilha fechada, passo a passo, pela escolha de uma aresta sujeita à condição de que uma aresta de corte (do subgrafo aresta-induzido pelas arestas que não estão na trilha que está sendo construída) é escolhida somente quando não há outro modo de fazê-lo.

A seguir, apresentamos de modo simplificado o Algoritmo de Fleury:

Algoritmo de Fleury

1. Escolha um vértice arbitrário v_o , e faça $W_o = v_o$;
2. Suponha que a trilha $W_j = v_o e_1 v_1 ..... e_j v_j$ tenha sido construída. Então, escolha uma aresta $\{ e_{j+1} \}$ do conjunto $E - \{e_1 , e_2 , ..., e_j\}$ de tal forma que:
   a) $\{ e_{j+1} \}$ é incidente em v_j ;
   b) a menos que não exista outra alternativa, $\{e_{j+1}\}$ não é uma aresta de corte de $G_j = G - \{e_1 , e_2 , ..., e_j\}$;
3. Se o passo 2 não pode ser executado, Pare.

Essa construção da trilha fechada euleriana é efetuada conforme o Teorema seguinte:

Teorema 5.2.1. *Se G é grafo euleriano então, qualquer trilha fechada em G, construída com o uso do Algoritmo de Fleury, é uma trilha fechada euleriana de G.*

Demonstração. Conseqüência das observações anteriores e do Corolário 5.1.2. □

Se G não é grafo euleriano, então qualquer passeio fechado que contém todas as arestas de G e, particularmente, aquele que tenha comprimento mínimo, atravessa algumas arestas mais de uma vez. Assim, o tour ótimo passará mais de uma vez por algumas arestas. A pergunta que se faz é, então: já que é necessário passar mais de uma vez por algumas arestas, quais serão as escolhidas? A seção seguinte trata deste problema mais geral. Antes disso, vamos nos deter num caso particular bastante simples de ser resolvido: O que fazer no caso do grafo G não ser euleriano mais que tem apenas dois vértices de grau ímpar?

Sejam u e v os dois vértices de G, ambos de grau ímpar. Com a ajuda do Algoritmo de Djikstra, é possível encontrar o caminho de comprimento mínimo entre os vértices u e v e, a partir desse caminho, duplicamos cada uma de

suas arestas. Feito isso, ficamos com um grafo G^* que é euleriano, pois G^* é a união do grafo G com esse caminho e daí, os vértices u e v passam a ter grau dois (par) cada um e os vértices internos desse caminho (que já tinham grau par antes da duplicação) também têm grau par. Daí, agora, é só utilizar o Algoritmo de Fleury em G^*.

5.2.1. ***O Problema do Carteiro Chinês em grafos quaisquer.***

Outro modo de solucionar o Problema do Carteiro Chinês em um grafo conexo G é o de determinar um multigrafo euleriano H, com uma quantidade mínima de arestas e de tal modo que H contém G como um subgrafo gerador. Como fazer isso? Veremos a seguir.

Se, a cada aresta de G trocarmos por duas arestas paralelas formando um multigrafo H, teremos então:

(1) H é um multigrafo euleriano, vez que H é conexo e;
(2) $d_H(v) = 2.d_G(v)$ para todo vértice v de $V(G)$ (o grau de cada vértice de H é o dobro do grau de cada vértice correspondente de G).

Além do mais, G é um subgrafo gerador de H. Assim, uma trilha fechada em H produz um passeio fechado em G, que contém todas as arestas de G. Se um grafo conexo G possui m arestas, ele tem um passeio euleriano de comprimento maior do que ou igual a m e menor do que ou igual a $2m$. Mas se G é euleriano com m arestas, então o comprimento de uma trilha fechada euleriana de G é m. Mas, por exemplo, se G é uma árvore com n vértices, o comprimento de um passeio euleriano fechado é $2(n-1)$.

Sabemos que se G não é euleriano, G contém um número par de vértices de grau ímpar, pois o somatório dos graus dos vértices de um grafo é par.

Seja $V*(G) = \{u_1, u_2, \ldots, u_{2n}\}$, $n \geq 1$, o conjunto de vértices de grau ímpar de G. Uma partição par de $V*(G)$ é uma partição de $V*(G)$ em n subconjuntos de tamanho 2 (dois), cada. Para uma partição par π, dada por $\pi = \{\{u_{11}, u_{12}\}, \{u_{21}, u_{22}\}, \ldots, \{u_{n1}, u_{n2}\}\}$, definimos:

(1) $D(\pi)$ por $D(\pi) = \sum_{j=1}^{n} d(u_{j1}, u_{j2})$;
(2) $m(G) = min\{D(\pi)\}$, onde o mínimo é obtido de todas as partições pares π de $V*(G)$.

Se G é grafo euleriano temos que $V*(G) = \varnothing$, e então definimos $m(G) = 0$.

Exemplo 5.2.2.1- Considere um grafo G que tenha 4 vértices de grau ímpar, a saber: u_1, u_2, u_3 , e u_4. Então, $V*(G)$ tem três partições pares (essa quantidade é obtida por uma parcela dos Números de Stirling de segunda espécie $S(4,2)$. Veja (Teixeira, 2006)), a saber:

$$\pi_1 = \{\{u_1, u_2\}, \{u_3, u_4\}\};$$

$$\pi_2 = \{\{u_1, u_3\}, \{u_2, u_4\}\};$$

$$\pi_3 = \{\{u_1, u_4\}, \{u_2, u_3\}\}.$$

O Teorema a seguir descreve uma maneira de determinar o tamanho de um passeio euleriano de qualquer grafo conexo G, e também permitirá obter um algoritmo que produz um passeio euleriano em um grafo conexo.

Teorema 5.2.2.2 - *Se G é um grafo conexo com m arestas, então um passeio euleriano de G tem comprimento $m + m(G)$.*

Uma vez que encontramos uma partição par $\pi_1 = \{\{u_{11}, u_{12}\}, \{u_{21}, u_{22}\}, \ldots, \{u_{n1}, u_{n2}\}\}$ de $V*(G)$ para a qual $m(G) = D(\pi_1)$, podemos então determinar um caminho P_j de u_{j1} para u_{j2} de comprimento mínimo $d(u_{j1}, u_{j2})$ em G, para $j = 1, 2, \ldots, n$. Pela duplicação das arestas de G que estão no caminho P_j, para cada $j = 1, 2, \ldots, n$, obtemos um multigrafo euleriano H que tem G como seu subgrafo gerador. Uma trilha fechada euleriana de H pode então ser obtida com o uso do algoritmo de Fleury, que gera um passeio euleriano em G. Muito embora tenhamos um modo de fornecer uma possível solução do Problema do Carteiro Chinês via o Teorema acima, esta solução requer:

(1) a determinação de todas as partições pares π de $V*(G)$;
(2) o cálculo de todos os $D(\pi)$ para essas partições e, em seguida, calcular $m(G)$.

Esse procedimento não é eficiente uma vez que a determinação do número de partições pares de $V*(G)$ é de complexidade $O(n^n)$, onde $2n$ é a quantidade de vértices de grau ímpar de G.

Mas, felizmente, já há uma maneira eficiente de determinar uma partição par π de $V*(G)$ e tal que $m(G) = D(\pi)$.

Vamos iniciar o algoritmo construindo um grafo completo $F \cong K_{2n}$ com pesos nas arestas. Os vértices de F correspondem aos vértices de grau ímpar de G e são rotulados com os mesmos índices. O peso de uma aresta de F é definido como a distância (ou o menor caminho) entre os correspondentes vértices em G. O Algoritmo de Moore (BFS) ou o Algoritmo de Djikstra (onde se considera G como um grafo com pesos, onde o peso $\mu(e_j) = 1$, para toda aresta e_j de $E(G)$) podem ser utilizados para determinar as distâncias entre os pares de vértices de grau ímpar de G.

Seja μ o maior peso dentre todos os pesos das arestas de G, e seja $\sigma = \mu + 1$. Seja F^* o grafo completo com $2n$ vértices, com pesos nas arestas, e obtido a partir do grafo F pela troca do peso $\mu(e_j)$ de cada aresta e_j de F pelo peso $\sigma - \mu(e_j)$ (ou seja, cada aresta de F^* tem seu peso determinado por $\sigma - \mu(e_j)$). O próximo passo é determinar um emparelhamento perfeito de F, cujo peso total é o mínimo possível. [Edmonds] desenvolveu um algoritmo eficiente para encontrar um emparelhamento de peso máximo em grafos valorados. Vamos então aplicar esse algoritmo ao grafo F^* para obter um emparelhamento perfeito de peso máximo $M = \{u_{11}u_{12}, u_{21}u_{22}, \ldots, u_{n1}u_{n2}\}$. Então, M é um emparelhamento perfeito de peso mínimo em F e, assim, $\pi = \{\{u_{11}, u_{12}\}, \{u_{21}, u_{22}\}, \ldots, \{u_{n1}, u_{n2}\}\}$ é uma partição par de $V*(G)$ para a qual $m(G) = D(\pi)$.

Em seguida, determinamos o menor caminho P_j de u_{j1} a u_{j2}, para todo $j = 1, 2, \ldots, n$ e duplicamos as arestas de G que fazem parte do caminho P_j. Este método faz com que se produza um multigrafo euleriano H onde G é seu subgrafo gerador. Finalmente, devemos encontrar uma trilha fechada euleriana de H utilizando o Algoritmo de Fleury. Essa trilha fechada euleriana de H corresponde a um passeio euleriano de G.

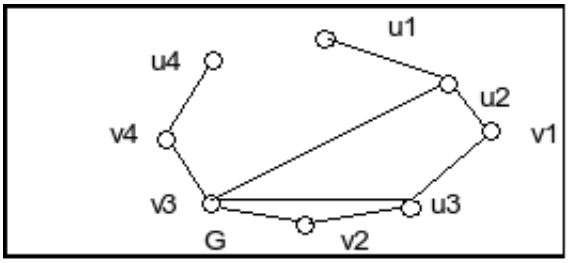
Apresentamos, a seguir, o detalhamento desse algoritmo:

Algoritmo para grafos arbitrários

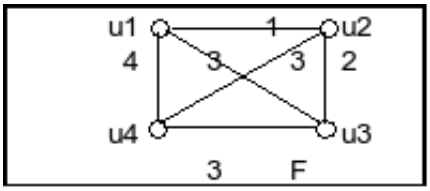
(1) Seja $\{u_1, u_2, \ldots, u_{2n}\}$, $n \geq 1$, o conjunto de vértices de grau ímpar do grafo G. Execute o Algoritmo de Djikstra em (G, u_k), para algum k, $1 \leq k \leq 2n$. Obtenha, dessa forma, o caminho de comprimento mínimo entre cada par de vértices de grau ímpar de G;

(2) Construa um grafo completo com pesos $F \cong K_{2n}$, tal que seus vértices correspondem aos vértices de grau ímpar de G, e o peso de uma aresta de F é definido como a distância entre os vértices correspondentes em G;
(3) Seja μ o maior entre os pesos das arestas de F, e seja $\sigma = \mu+1$. Construa F^* o grafo completo com $2n$ vértices, com pesos nas arestas, e obtido a partir do grafo F pela troca do peso $\mu(e_j)$ de cada aresta e_j de F pelo peso $\sigma - \mu(e_j)$ (ou seja, cada aresta de F^* tem seu peso determinado por $\sigma - \mu(e_j)$);
(4) Encontre um emparelhamento perfeito de peso máximo $M = \{u_{11}u_{12}, u_{21}u_{22}, \ldots$ em F^*. M é um emparelhamento perfeito de peso mínimo em F e, assim, $\pi = \{\{u_{11}, u_{12}\}, \{u_{21}, u_{22}\}, \ldots, \{u_{n1}, u_{n2}\}\}$ é uma partição par de $V*(G)$ para a qual $m(G) = D(\pi)$;
(5) Para o menor caminho Pj de u_{j1} para u_{j2}, com $j = 1, 2, \ldots, n$, duplique as arestas de G que aparecem em P_j. Seja H o multigrafo euleriano produzido neste processo;
(6) Utilize o Algoritmo de Fleury para encontrar uma trilha fechada euleriana de H. Esta trilha fechada euleriana de H corresponde a um passeio euleriano de G.

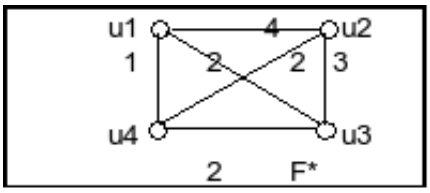
Exemplo Vamos mostrar, com um exemplo, a execução do algoritmo acima:
Considere o grafo G apresentado na figura abaixo:



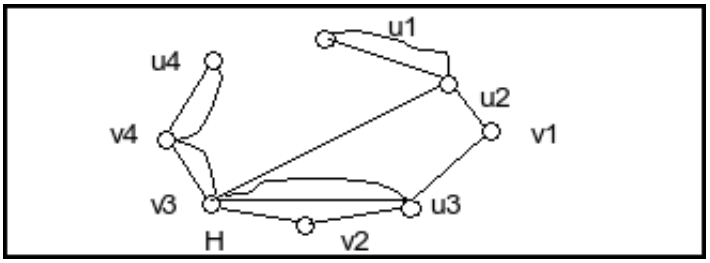
Os vértices u_1, u_2, u_3 e u_4 têm, ambos, grau ímpar. Assim, o grafo F é obtido e representado a seguir.



De posse do grafo F e considerando que $\mu = 4$, seja $\sigma = \mu + 1 = 4 + 1 = 5$, construímos, então, o grafo completo F^* com $2n$ vértices, com pesos nas arestas, e obtido a partir do grafo F pela troca do peso $\mu(e_j)$ de cada aresta e_j de F pelo peso $\sigma - \mu(e_j) = 5 - \mu(e_j)$ (ou seja, cada aresta de F^* tem seu peso determinado por $5 - \mu(e_j)$). O grafo $F*$ está na figura abaixo:

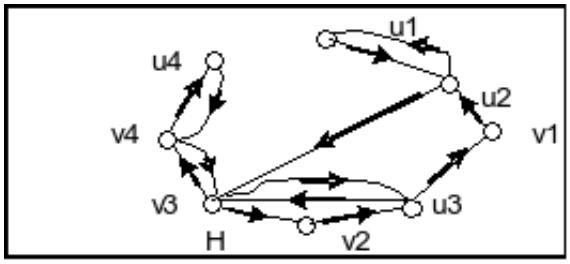


Observe que $M = \{u_1u_2, u_3u_4\}$ é um emparelhamento máximo para $F*$. Portanto, $\pi = \{\{u_1, u_2\}, \{u_3, u_4\}\}$ é uma partição par de $V*(G)$ tal que $m(G) = D(\pi)$. Como $d_G(u_1, u_2) = 1$ e $d_G(u_3, u_4) = 3$, tem-se que $m(G) = 4$. O multigrafo H pode ser obtido pela duplicação das arestas do menor caminho de u_1 para u_2, bem como do menor caminho de u_3 para u_4.



Arestas: e1={v4,u4} e2={u4,v4}
e3={v3,v4} e4={v4,v3}
e5={u3,v3} e6={v3,u3}
e7={v3,v2} e8={v2,u3}
e9={u3,v1} e10={u2,v3}
e11={v1,u2 e12={u1,u2}
e13={u2,u1}

Como $u_1, e_{12}, u_2, e_{10}, v_3, e_3, v_4, e_1, u_4, e_2, v_4, e_4, v_3, e_7, v_2, e_8, u_3, e_5, v_3, e_6, u_3, e_9, v_1, e$ é uma trilha fechada euleriana para H, segue que:

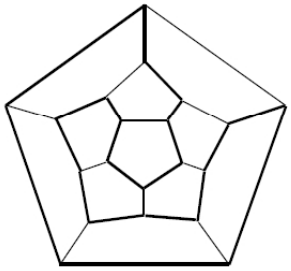


$u_1, u_2, v_3, v_4, u_4, v_4, v_3, v_2, u_3, v_3, u_3, v_1, u_2, u_1$ é um passeio euleriano de G.

6. Grafos Hamiltonianos

6.1. **Histórico.**

Um circuito passando exatamente uma vez por cada vértice de um grafo é chamado de **circuito hamiltoniano**, em homenagem ao matemático irlandês William Rowan Hamilton (1805-1865), que estudou este problema no grafo determinado pelas arestas de um dodecaedro regular. Um grafo que possui um circuito hamiltoniano é então chamado de **grafo hamiltoniano**.



Um grafo hamiltoniano

Apesar de terem sido estudados por vários séculos, não há uma boa caracterização dos grafos hamiltonianos. Há diversas famílias de grafos para os quais existe um circuito hamiltoniano:

- um exemplo trivial é um grafo completo com pelo menos 3(três) vértices;
- outro exemplo trivial é um ciclo simples.

Mas também é possível estabelecer certas condições que implicam na não-existência de um circuito hamiltoniano, como veremos nos teoremas que serão apresentados a seguir.

Mas, ***uma caracterização geral dos grafos hamiltonianos ainda não foi encontrada*** e, considerando os avanços obtidos em teoria da computação, nas últimas décadas, parece improvável que ela seja encontrada algum dia.

O problema de decidir se um grafo é hamiltoniano está na mesma lista de muitos outros problemas ilustres, e que têm as seguintes características em comum:

(1) O problema possui uma assimetria fundamental: é muito fácil ficar convencido da existência de um circuito hamiltoniano em um grafo: basta, para tal, exibir ao menos um tal circuito. No entanto, é difícil, em geral, convencer alguém de que tal circuito não existe no grafo dado. Num grafo que não possui um circuito hamiltoniano não existe um argumento simples e geral para demonstrar esse fato, pois os teoremas que asseguram tal condição são de constatação complexa, em muitos grafos.

(2) Não se conhece um algoritmo eficiente (de complexidade exponencial) para verificar se um grafo é hamiltoniano (por eficiente, entende-se um algoritmo em que o número de passos seja limitado por um polinômio no número de vértices do grafo, ou seja, um algoritmo de complexidade polinomial). Além disso, parece improvável que tal algoritmo possa algum dia ser encontrado, pois a sua existência implicaria na existência de algoritmos eficientes para um grande número de outros problemas para

os quais também não se conhecem algoritmos eficientes. Estes problemas (incluindo o de verificar a existência de circuito hamiltoniano) formam uma classe de problemas chamados de **NP-completos** (mais adiante serão dadas idéias mais gerais sobre tais problemas). Um outro problema famoso da classe dos **NP-completos** é o de determinar o número mínimo de cores que podem ser usadas para colorir os vértices de um grafo, de modo que vértices de mesma cor não sejam ligados por uma aresta e o Problema Quadrático de Alocação, que trata do seguinte:

Vamos apresentar, a seguir, uma **condição necessária** que permite decidir, de forma não muito eficiente, se um grafo é hamiltoniano.

Teorema 6.1.1. *Se G é grafo hamiltoniano então, para qualquer subconjunto próprio S de $V(G)$, vale $\omega(G-S) \leq |S|$.*

Demonstração. Seja C um circuito hamiltoniano no grafo G. Então, qualquer que seja o subconjunto próprio e não vazio S, tem-se que: $\omega(C-S) \leq |S|$. Como $(C-S)$ é um subgrafo gerador de $G-S$, tem-se que $\omega(G-S) \leq \omega(C-S) \leq |S|$. □

Utilizar o Teorema acima nem sempre garante uma boa técnica, vez que, por exemplo, o Grafo de Petersen não é hamiltoniano e isso não pode ser deduzido do Teorema 6.1.2. Segundo [Jurkiewicz], as condições necessárias são de dois tipos:

- ou resultados **bastante simples**, como, por exemplo, os seguintes resultados:
  (1) Todo grafo hamiltoniano é 2-conexo;
  (2) Todo grafo hamiltoniano é 3-circulável. Um grafo G' com n vértices, $n \geq 3$, é dito k-circulável se existem k vértices quaisquer de G que pertencem a um mesmo ciclo;
  (3) Todo grafo hamiltoniano tem um 2-fator. Seja um grafo G. Um grafo parcial de G é dito um grafo k-fator, se é regular e de grau igual a k.
  (4) Todo grafo hamiltoniano tem o ciclo hamiltoniano como subgrafo 2-fator.
  (5) Esse resultado foge à regra: (Tutte): Se $G=(V,E)$ é grafo hamiltoniano, então não existe uma partição X, formada por 3(três) conjuntos de arestas: R, S e T tais que $2|S|+|U_T|+\frac{|R|}{2}+\frac{1}{2}|U_{RT}| < n$, onde U_T e U_{RT} são conjuntos de arestas unindo vértices de T entre si e vértices de R a vértices de T, respectivamente.

- ou resultados cuja verificação é **bastante complexa**, sendo pouco efetivos para verificar se um grafo é ou não hamiltoniano.

A seguir, vamos analisar condições suficientes para que um grafo seja hamiltoniano.

Teorema 6.1.2. *(Dirac, 1952) Seja G um grafo simples com n vértices, $n \geq 3$. Se $\delta \geq \frac{n}{2}$, então G é hamiltoniano.*

Demonstração. Suponha, por absurdo, que existe um grafo simples G^* com n vértices, $n \geq 3$,e $\delta \geq \frac{n}{2}$ e com o maior número m de arestas possíveis ($m \leq \frac{n(n-1)}{2}$), tal que G^* não é hamiltoniano. Como G^* não é hamiltoniano, ele não é um grafo completo. Sejam u e v dois vértices não adjacentes em G^*. Pela escolha de G^*, acrescentando a aresta $\{u, v\}$ ao grafo G^*, ou seja, obtendo o grafo $G^* + \{u, v\}$, tem-se que esse grafo é hamiltoniano. Como G^* não é hamiltoniano, todo circuito hamiltoniano no grafo $G^* + \{u, v\}$ deverá conter, necessariamente, a aresta $\{u, v\}$. Assim, existe um caminho hamiltoniano $v_1, v_2, \ldots, v_n$ em G^* cujo vértice inicial é $u = v_1$ e vértice final em $v = v_n$. Agora, considere os conjuntos $S = \{v_j/(u, v_{j+1}) \in E(G)\}$ e $T = \{v_j/(v_j, v) \in E(G)\}$. É claro que $d(u) = |S|$ e $d(v) = |T|$. Como $v_n \notin (S \cup T)$, temos que $|S \cup T| < n$. Assim, tem-se que $d(u) + d(v) = |S| + |T| = |S \cup T| + |S \cap T|$. Como $d(u) + d(v) \geq n$ e $|S \cup T| < n$, tem-se que $|S \cap T| \geq 1$. Seja v_j o elemento de $S \cap T$. Então, $v_1, v_{j+1}, v_{j+2}, \ldots, v_n, v_j, v_{j-1}, \ldots, v_1$ é um circuito hamiltoniano em G. □

Teorema 6.1.3. *(Ore, 1960) Seja G um grafo simples com n vértices, $n \geq 3$ e sejam u e v dois vértices não adjacentes em G tais que $d(u)+d(v) \geq n$. Então, G é hamiltoniano.*

Demonstração. Utilize os mesmos argumentos da prova do teorema anterior. □

Os dois teoremas anteriores necessitam tomar um número de arestas muito grande: no Teorema de Dirac é preciso que o grafo possua, no mínimo, $\frac{n^2}{4}$ arestas $> \frac{n(n-1)}{2}$ (maior do que a metade do total de arestas).

Com base no Teorema de Ore, [Bondy e Chvátal] modificaram a hipótese de modo a propor um resultado mais efetivo, com base no conceito de fecho: O fecho de um grafo G, indicado por $\Phi(G)$, é um grafo obtido a partir do grafo G, adicionando, iterativamente, uma aresta a cada par de vértices não adjacentes u e v cuja soma dos graus é, no mínimo, igual a n, ou seja: $d(u) + d(v) \geq n$. Essa noção está intimamente associada à noção de que, dada um propriedade S, então $G + (u, v)$ satisfaz S se e somente se G satisfaz S. Podemos então pensar em: $G + (u, v)$ é hamiltoniano se e somente se G é hamiltoniano. Ora, se G é hamiltoniano, é claro que $G + (u, v)$ também é hamiltoniano. Por outro lado, supondo que $G + (u, v)$ é hamiltoniano e, então, utilizando os mesmos argumentos utilizados na prova do Teorema de Dirac, mostramos que G é hamiltoniano. Segundo [Bondy e Chvátal], o fecho de um grafo pode ser obtido através de um algoritmo de complexidade polinomial, bem como um ciclo hamiltoniano no fecho $\Phi(G)$ pode ser transformado num ciclo hamiltoniano em G, por um outro algoritmo de complexidade polinomial. Assim, se $\Phi(G) = K_n$ para um qualquer grafo G, o problema da obtenção do ciclo hamiltoniano será de complexidade polinomial para esse fecho. Assim, podemos apresentar os seguintes resultados:

Teorema 6.1.4. *O fecho $\Phi(G)$ está bem definido.*

Demonstração. Sejam G_1 e G_2 dois grafos obtidos a partir do grafo G pela adição iterativa de arestas ligando pares de vértices não adjacentes e cuja soma dos graus é de, no mínimo, $n = |V|$. Sejam $e_1, e_2, \ldots, e_r$ e $f_1, f_2, \ldots, f_s$ as respectivas seqüências de arestas adicionadas ao grafo G para obter os grafos G_1 e G_2, respectivamente. Vamos mostrar que cada aresta e_j é aresta do grafo G_2 e cada aresta f_j é aresta do grafo G_1. Se possível, seja $\{e_{k+1}\} = (u, v)$ a primeira aresta da seqüência $e_1, e_2, \ldots, e_r$ que não é aresta de G_2. Considere o grafo $H = G + \{e_1, e_2, \ldots, e_r\}$. Por definição do grafo G_1, tem-se que $d_H(u) + d_H(v) \geq n$. Mas, pela escolha de $\{e_{k+1}\}$, concluímos que H é subgrafo de G_2, e portanto, $d_{G_2}(u) + d_{G_2}(v) \geq n$. Mas isso é uma contradição visto que, pela hipótese, $\{e_{k+1}\} = (u, v)$ não é aresta de G_2. Logo, cada aresta e_j é aresta de G_2 e, de modo similar, cada aresta f_j é aresta de G_1. Portanto, tem-se que $G_1 = G_2$ e, assim, o fecho $\Phi(G)$ está bem definido. □

Teorema 6.1.5. *Um garfo G é hamiltoniano se e somente se o seu fecho é hamiltoniano.*

Demonstração. Basta aplicar o Teorema 6.1.2 a cada vez que uma aresta é adicionada. □

Teorema 6.1.6 (Bondy e Chvátal)**.** *Seja G uma grafo com n vértices, $n \geq 3$. Se $\Phi(G) = K_n$ então G é grafo hamiltoniano.*

Teorema 6.1.7 (Chvátal e Erdös)**.** *Seja G uma grafo com n vértices, $n \geq 3$. Se $\kappa(G) \geq \alpha(G)$ então G é grafo hamiltoniano, onde $\kappa(G)$ é a conectividade vértices e $\alpha(G)$ é o número de estabilidade interna.*

Teorema 6.1.8. *Um grafo G é hamiltoniano se G é grafo adjunto de um grafo euleriano.*

Teorema 6.1.9 (Haggkvist e Micoghossian,1981)**.** *Seja G uma grafo com conectividade vértices $\kappa(G) \geq 2$ e de grau mínimo $\delta(G)$ que satisfazem à desigualdade $\delta(G) \geq \frac{n+k}{3}$. Então G é grafo hamiltoniano.*

6.2. **O Problema do Caixeiro Viajante.**

6.2.1. ***Descrição do Problema do Caixeiro Viajante e definições básicas.***

O famoso Problema do Caixeiro Viajante (PCV) é um dos problemas de roteamento mais conhecidos e ao mesmo tempo o mais simples de ser enunciado. Neste problema, temos um número finito de cidades que devem ser visitadas uma única vez por um vendedor (caixeiro), que deve retornar à cidade de onde partiu. A rota (tour) deve ser construída de forma que seu custo seja minimizado e este custo é normalmente associado à distância a ser percorrida. Este problema, freqüentemente, é o ponto de partida de diferentes pesquisadores para iniciar testes de novas idéias, questionamentos e conjecturas, antes de se lançarem para outras questões mais complexas ou formulações de modelos correlatos. É pura fonte de inspiração em muitos anos dedicados à pesquisa de formas eficientes de buscar soluções ótimas.

Entretanto, apesar da forma simples de se enunciar o problema e descrevê-lo, encontrar uma rota de custo mínimo (menor distância) visitando todas as cidades uma única vez e retornando à cidade inicial é um problema difícil. Por esse motivo, muito embora tenham sido dedicados anos de pesquisas à solução deste problema, progressos continuam a ser realizados como veremos no decorrer deste trabalho. Este problema pertence à classe de problemas NP-hard, ou

seja, considerados difíceis no sentido de não existir um algoritmo em tempo polinomial capaz de permitir a obtenção de sua solução ótima. Isso motivou ao oferecimento do prêmio de um milhão de dólares para se provar que as classes de problemas NP = P ou apresentar uma prova de que a igualdade não é verdadeira. Caso seja verdadeira a igualdade, isso implica em que, por exemplo, o Problema do Caixeiro Viajante possa pertencer à classe dos problemas que podem ser resolvidos com a utilização de um algoritmo em tempo polinomial e, aí, uma nova corrida científica buscaria determinar um melhor algoritmo que fornecesse uma solução ótima para o problema. Um verdadeiro fascínio para os pesquisadores da área.

O PCV é formulado como a minimização do valor de uma função f chamada de função objetivo que mede o custo a ser minimizado e é um dos mais famosos problemas de otimização combinatória graças à conexão com muitos problemas de otimização, à facilidade de sua modelagem enquanto um problema de otimização e ao fato de ser considerado um problema NP-hard, conforme provado por Held e Karp (1970-1971) e Garey e Johnson (1976). Isso justifica o uso de heurísticas para sua solução, principalmente para problemas de larga escala. Johnson e McGeoch (1997) apresentam uma recente revisão sobre os métodos de busca local aplicados ao problema.

Até para situações particulares do PCV, tal como: dado um grafo G com m arestas cujo custo de cada aresta de G seja tal que: $C(\{e_j\}) \in \{1, 2\}$ para todas as arestas $\{e_j\}$, não se conhece algoritmos polinomiais que encontrem a solução ótima para o PCV, conforme Garey e Johnson (1978). Há, porém, algumas variações para o problema e também em algumas situações particulares já é possível encontrar soluções muito boas desde que façamos restrições ao conjunto de entradas, utilizando propriedades que facilitam a obtenção de uma solução de forma bastante eficiente.

Considere a situação de sequenciamento de n tarefas numa única máquina. Um custo de operação é atribuído a cada vez que uma nova tarefa é agendada, mas esse custo de operação depende da tarefa imediatamente anterior que foi processada na máquina. O objetivo é determinar a seqüência que minimiza a soma total dos custos de operação. A este tipo de problema está associado o clássico nome de "caixeiro viajante"onde cada tarefa pode ser entendida como uma "cidade a ser visitada". As "distâncias"entre as cidades estão associadas aos custos de operação. O caixeiro deve visitar cada cidade uma única vez e

a rota a ser construída deve permitir a visita a todas as cidades de forma a minimizar o custo associado à distância a ser viajada.

Definição 6.2.1.1: Dizemos que a desigualdade abaixo é a desigualdade triangular (DT) quando: Dada a matriz de distâncias $C = (c^{ij})$, onde c^{ij} é a distância entre a cidade i e a cidade j, se $c^{ij} \geq \overline{c}^{ij}$ onde $\overline{c}^{ij} = c^{ii_1} + c^{i_1 i_2} + c^{i_2 i_3} + \cdots + c^{i_k j}$, então o elemento c^{ij} pode ser substituído por $\overline{c}^{ij}$ em C. Em outras palavras, o elemento c^{ij} da matriz C deve determinar a menor distância entre duas cidades. Isso pode ser obtido se utilizarmos um algoritmo de determinação do menor caminho.

Definição 6.2.1.2: Uma solução do Problema do Caixeiro Viajante (PCV) constitui-se em um tour (rota) se, começando de uma cidade arbitrária i cada cidade é visitada exatamente uma vez antes de retornar a i (supõe-se que a matriz C de distâncias satisfaz à DT).

Definição 6.2.1.3: Um subtour (subrota) pode ser definido como um tour incluindo k cidades, onde $k < n$. Assim, um subtour não é uma solução viável para o problema.

Definição 6.2.1.4: Na forma geral do Problema do Caixeiro Viajante, é dado um conjunto finito de pontos V, onde V é o conjunto de vértices de um grafo completo $G = (V, E)$ e um custo c_{ij} de viagem entre cada par de vértices i, j ($\{i, j\}$ é dita a aresta que liga o vértices i ao vértice j). Um tour é um circuito que passa exatamente uma vez por cada ponto de V. O problema do caixeiro viajante (PCV) é encontrar um tour de custo mínimo.

6.2.1.5 - Formulação matemática do PCV

O PCV pode ser modelado como um problema de grafos e formulado como:

Dado um grafo G conexo, com n vértices e m arestas, deseja-se, se possível, determinar um circuito hamiltoniano em G com custo mínimo, ou seja: deseja-se minimizar o custo de um circuito hamiltoniano C em G, dado por $MinZ = c(C)$, onde $c(C) = \sum_{j=1}^{n} C(\{e_j\})$, onde $\{e_j\}$ indica a aresta do circuito hamiltoniano e $C(\{e_j\})$ é o custo de cada aresta desse circuito.

PCV pode ser formulado matematicamente como: Uma rota incluindo n cidades consiste de n arcos direcionados (um arco representa o sentido do percurso ligando duas cidades consecutivas). Esses arcos serão designados de acordo com a ordem na rota, tal que ao k-arco direcionado é associado o k-ésima comprimento do itinerário.

Defina:

$$x_{ijk} = \begin{cases} 1, & \text{se o } k\text{-ésimo arco direcionado vai da cidade } i \text{ para a cidade } j \\ 0, & \text{caso contrário} \end{cases}$$

As restrições do problema podem ser classificados em 4 tipos, a saber:

(1) somente uma cidade pode ser alcançada a partir de uma determinada cidade i.

$$\sum_j \sum_k x_{ijk} = 1, \quad i = 1, \ldots, n$$

(2) somente uma rota pode ser associada a um determinado k.

$$\sum_i \sum_j x_{ijk} = 1, \quad k = 1, \ldots, n$$

(3) somente uma outra cidade pode iniciar um arco a partir de uma determinada cidade j.

$$\sum_i \sum_k x_{ijk} = 1, \quad j = 1, \ldots, n$$

(4) Dado que o k-ésimo arco direcionado termina numa determinada cidade j, o $(k+1)$-ésimo arco deve começar na mesma cidade j. Isto assegura

que a viagem consiste de segmentos "conexos":

$$\sum_{\substack{i \\ i \neq j}} x_{ijk} = \sum_{\substack{r \\ r \neq j}} x_{jr(k+1)}, \quad \text{para todo } j \text{ e } k$$

A função objetivo é então dada por:

$$Min\ z \sum_i \sum_j \sum_k d^{ij} x^{ijk}, \ \ i \neq J$$

onde d_{ij} é a distância mínima (sentido único) da cidade i para a cidade j.

A viabilidade da resolução de problemas de tamanho prático usando a formulação acima é, portanto, altamente questionável.

A estrutura especial do problema pode ser explorada para desenvolver um algoritmo especializado eficiente. Ou seja, podemos observar que se as restrições (4) são temporariamente ignoradas, o problema pode ser reduzido para um problema de alocação bidimensional simples. Esta é a base para o desenvolvimento de um novo algoritmo, mas hipóteses especiais devem ser feitas para assegurar que a solução obtida constitui uma rota (de itinerário conexo).

6.2.2. ***Tipos particulares de PCV.***

Entre algumas variações de abordagem do PCV, destacamos as seguintes:

- O PCV é dito **simétrico** se, dados dois quaisquer vértices v_i e v_j, representando, respectivamente, as cidades C_i e C_j, a distância entre a cidade C_i e a cidade C_j é igual à distância entre as cidades C_j e C_i. Caso contrário, o PCV é dito **assimétrico**1.
- O PCV é dito **completo** se existe um caminho direto entre todos os vértices do grafo. Caso contrário, diz-se que é **não-completo**.
- O PCV restrito ao conjunto de instâncias onde G é grafo completo e os pesos de cada uma de suas arestas satisfazem à desigualdade triangular é conhecido como o PCV **métrico**.
- O PCV é dito **Euclideano (ou geométrico)** se o peso de cada uma das arestas do grafo é determinado pela distância euclideana entre os vértices extremos dessa aresta, considerando um sistema de eixos ortogonais necessário para conceber a definição de distância entre dois pontos, como por exemplo, $d(u, v) = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}$, onde $u = (x_1, \ldots, x_n)$, $v = (y_1, \ldots, y_n)$.

- O PCV é dito **retangular**, ou que satisfaz à métrica de Manhattan se $d(u, v) = |x_i - x_j| + |y_i - y_j|$.
- Há ainda outras duas métricas que podem ser utilizadas:
  $\left[(x_i - x_j)^2 + (y_i - y_j)^2 + 0.5(x_i - x_j).(y_i - y_j)\right]^{1/2}$ chamada de métrica afim ou então a
  $max\{|x_i - x_j|, |y_i - y_j|\}$, chamada de métrica de Chebyshev.

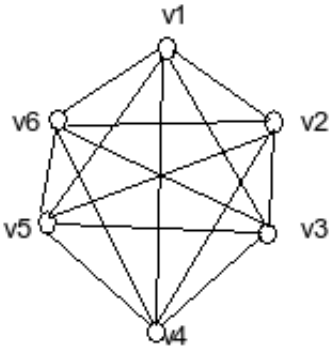
Em todas essas variações ainda temos o PCV como um problema NP-hard.

6.2.3. ***Um algoritmo de aproximação para o PCV.***

Como o PCV é um problema da classe dos problemas NP-completo, a tentativa de encontrar um algoritmo eficiente para a sua solução é prioridade bastante baixa. Porém, muitos algoritmos eficientes foram desenvolvidos e fornecem circuitos hamiltonianos com peso "pequeno", mas não necessariamente com peso mínimo. Considerando que em muitas situações práticas não se pode esperar um enorme tempo para a obtenção da solução ótima do problema, uma solução desse tipo é perfeitamente aceitável. Estes algoritmos, por não fornecerem soluções ótimas, mas soluções que estão bastante próximas das soluções ótimas, são então chamados de ***algoritmos de aproximação***. Em geral, associam-se a estes tipos de algoritmos os problemas que estão na classe NP.

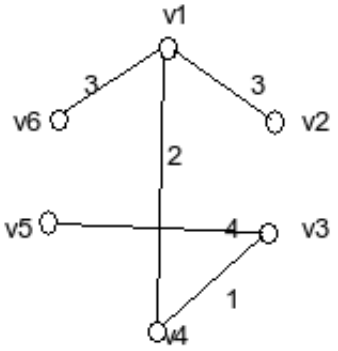
Consideremos que o grafo G de entrada para o problema satisfaz à desigualdade triangular e seja T uma árvore geradora de peso mínimo em G. A remoção de qualquer aresta de um circuito hamiltoniano C de peso mínimo em G produz, então, uma árvore geradora de G. Daí, $p(T) \leq p(C)$, onde $p(T)$ é o peso total de T. Seja H um multigrafo euleriano obtido pela duplicação de cada aresta de T. Então, uma trilha fechada euleriana de T fornece um passeio fechado em G, cujo peso é $2.p(T)$. Um algoritmo que produz um circuito hamiltoniano em um grafo completo e valorado em suas arestas, satisfazendo à desigualdade triangular e cujo peso total é no máximo $1,5$ o peso de um circuito hamiltoniano mínimo foi proposto por Christofides, 1976.

Exemplo: Considere o grafo G valorado em suas arestas, mostrado na figura abaixo:



{v1,v2}=3 {v2,v6}=5
{v1,v3}=3 {v3,v4}=1
{v1,v4}=2 {v3,v5}=4
{v1,v5}=7 {v3,v6}=4
{v1,v6}=3 {v4,v5}=5
{v2,v3}=3 {v4,v6}=5
{v2,v4=4 {v5,v6}=4
{v2,v5}=5

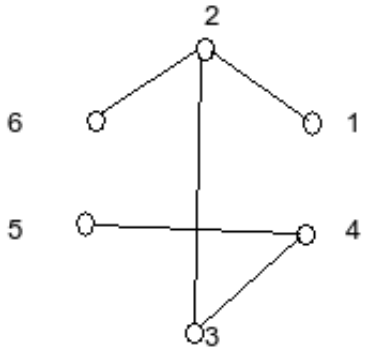
A figura abaixo mostra uma árvore geradora de peso mínimo em G.



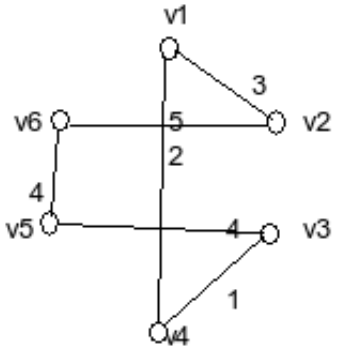
Árvore geradora de custo mínimo T

Assim, tem-se que $v_2, v_1, v_4, v_3, v_5, v_3, v_4, v_1, v_6, v_1, v_2$ é um passeio fechado de comprimento igual a 26 para o grafo G.

Executamos, agora, uma busca em profundidade em T, começando com um vértice folha de T, ou seja, um vértice de grau igual a 1. Sejam $v_{i1}, v_{i2}, \ldots, v_{in}$ os vértices de T ordenados pelos seus tempos de descoberta de execução do DFS. Então, segue da desigualdade triangular que o peso de um circuito $C^* = v_{i1}, v_{i2}, ..., v_{in}, v_{i1}$ é, no máximo, o peso do multigrafo H. Portanto, $p(C^*) \leq 2.p(T) \leq 2.p(C)$. A figura a seguir mostra uma busca em profundidade da árvore da figura acima.



A figura abaixo mostra um circuito hamiltoniano de peso 19 do grafo completo com pesos G.



De maneira mais formal, apresentamos o algoritmo que implementa os procedimentos efetuados acima:

Algoritmo de Aproximação para o Problema do Caixeiro Viajante (PCV)

(esse algoritmo determina um circuito hamiltoniano de "pequeno"peso em um grafo completo G, com $n = |V| \geq 3$, que satisfaz à desigualdade triangular).

- Utilize o Algoritmo de Kruskal ou o Algoritmo de Prim para obter uma árvore geradora de peso mínimo T no grafo G;
- Conduza uma buisca em profundidade em T, utilizando o DFS sobre um vértice de grau 1 de T;
- Se $v_{i1}, v_{i2}, \ldots, v_{in}$, é a ordem em que os vértices de T são visitados no Passo 2, então mostre como resultado o circuito hamiltoniano $v_{i1}, v_{i2}, \ldots, v_{in}, v_{i1}$.

Podemos mostrar que o algoritmo acima constrpoi um circuito hamiltoniano C, de um grafo G valorado em suas arestas, de tal forma que seu peso total nunca ultrapassa, por uma fator igual a 2, o peso de um circuito hamiltoniano ótimo C^*. Ou seja, $p(C) \leq 2.p(C^*)$. (Em algumas situações esse limitante é satisfeito, visto que foi obtido utilizando-se um algoritmo eficiente). Isso pode ser descrito no seguinte teorema:

Teorema: *Para um grafo completo G valorado em suas arestas, o peso de um circuito hamiltoniano de G, obtido pelo algoritmo descrito acima, é menor que duas vezes o peso de um circuito hamiltoniano de peso mínimo em G.*

Nas seções seguintes discutiremos algoritmos que funcionam muito bem empiricamente, mas para os quais ainda podemos oferecer garantias muito fracas quanto à qualidade de suas soluções. Nestas seções, serão apresentadas grande parte das preocupações dos pesquisadores das áreas de Ciência da Computação e Pesquisa Operacional que, ao longo de muitos anos, têm se dedicado ao PCV

(já há um livro inteiramente dedicado ao PCV. Veja E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan and D.B.Shmoys, 1985).

Nesta visão, a seguir, deixamos de lado um pouco do olhar matemático do problema e o encaramos com preocupações de pesquisadores ligados às áreas mencionadas acima. Fizemos questão de colocar, nessas seções, o avanço enorme, já obtido, na melhoria de encontrar soluções ótimas do PCV, muito embora o problema seja da classe NP-hard.

6.2.4. **Métodos de solução do PCV.**

Uma boa coletânea de trabalhos traçando a história e a pesquisa sobre o PCV pode ser encontrada em Lawler, Lenstra, Rinnooy Kan, e Shmoys (1985).

Nenhum algoritmo em tempo polinomial é conhecido para resolver o PCV em geral. Conseqüentemente, muitos pesquisadores acreditam que não exista tal método de solução eficiente porque, se tal algoritmo existisse, isso implicaria em que pudéssemos resolver praticamente todos os problemas em otimização combinatória com algoritmos em tempo polinomial.

Os métodos ótimos de resolução conhecidos são os do tipo "Enumeração Implícita": como os algoritmos "Branch and Bound" (MB&B), "Plano de Corte" (MPC e "Programação Dinâmica", segundo Burkard(1983).

Na literatura científica encontram-se inúmeras outras técnicas de resolução para o PCV, pois os métodos enumerativos nem sempre são eficazes no sentido de que sua eficiência implica em enorme utilização de tempo computacional, comprometendo, por vezes, o objetivo a ser alcançado. Desse modo, os métodos heurísticos e meta-heurísticos ganharam destaque para obtenção da solução do PCV. Esses métodos podem ser classificados em dois tipos:

Construtivos - baseiam-se na construção de uma solução viável para o problema partindo de uma solução não muito boa, dita trivial, e sobre a qual são aplicadas diferentes técnicas de modo a melhorar a qualidade da solução a ser obtida;

Melhoramento - baseiam-se na busca da melhoria das soluções a partir de modificações feitas sobre as soluções viáveis já anteriormente encontradas.

Entretanto, os PCV's surgem realmente na prática, e, então, problemas relativamente grandes podem, agora, ser resolvidos por otimização, com a obtenção de soluções "boas", próximas da solução ótima. O Problema do Caixeiro Viajante com o maior número de vértices até agora resolvido contém 7397 vértices e foi resolvido em Agosto de 1994 (Ver Applegate, Chvátal, Cook [1995]). As

coordenadas dos vértices para outras instâncias desse problema estão contidas na biblioteca de problemas de teste "PCVLIB"descrita em Reinelt (1991).

6.2.5. ***Uma formulação baseada no Modelo de Alocação.***

O conhecido Modelo de Alocação (PAL) pode ser adaptado para gerar uma formulação particularmente atraente do PCV. Essa formulação é base tanto para o (MPC) quanto para o (MB&B).

Seja $x_{ij} = 1$, se o caixeiro viajar diretamente da cidade i até a cidade j, e $x_{ij} = 0$ em caso contrário. Uma condição necessária para que tenhamos um tour é que a cidade i se conecte a apenas uma cidade e que se chegue à cidade j partindo de exatamente uma cidade. Isso gera a seguinte formulação:

$$Min\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}, \ \text{ onde } \ c_{ij} = \infty \text{ para } \ i = j$$

s.a.

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n$$

$$x_{ij}(0, 1)$$

a solução é um tour(forma ciclo)

A condição $c_{ij} = \infty$ para $i = j$ garante que $x_{ij} = 0$, como se desejaria que fosse numa condição em que o tour seja uma solução.

Exceto pela exigência de que a solução seja um tour, a formulação é um modelo de alocação. Infelizmente, não há garantia de que a solução ótima do modelo de alocação seja um tour. Mas, provavelmente, tal solução consistirá de subtours.

A formulação acima introduz o PCV para determinar o ciclo mais curto para visitar n cidades sendo que cada cidade é visitada uma vez. Uma afirmação básica para esta formulação é que a matriz das distâncias $C = (c_{ij})$, atende à desigualdade triangular.

A suposição de que o problema atende à desigualdade triangular, permite garantir que cada cidade pode ser visitada exatamente uma vez. Por outro

lado, se a desigualdade triangular não é satisfeita, cada cidade pode ser visitada mais de uma vez para obter-se o caminho mais curto. Uma vez que C pode sempre satisfazer à desigualdade triangular, vamos assumir que cada cidade é visitada exatamente uma vez.

O objetivo dos algoritmos que vamos apresentar a seguir, é mostrar como a restrição de tour pode ser imposta, explicitamente, de forma a que a solução ótima seja um tour. Note que para um problema com n cidades, existem $(n-1)!$ tour´s possíveis, de forma que a enumeração explícita de todos esses tour´s não é prática recomendada pois, por exemplo: $10! = 3.628.800$, o que é inviável!.

6.2.6. ***Algoritmos Branch & Bound (B&B).***

Os três métodos que serão apresentados a seguir são baseados na formulação do modelo de alocação do PCV. De fato, o problema em cada nó (do algoritmo B&B) é essencialmente um modelo de alocação que modifica o problema de alocação nxn original, de forma que os subtours sejam eliminados.

6.2.6.1 - Algoritmo de Eliminação de Subtours

Esse algoritmo é baseado no princípio geral de B&B. Os únicos detalhes exigidos para completar o algoritmo são:

- determinação dos limites superior e inferior em relação ao ótimo da função objetivo do PCV; isto é, dado que z^* é o valor ótimo da função objetivo associado a um tour, então e são determinados tais que $\underline{z} \leq z^* \leq \overline{z}$, e
- especificar o procedimento exato para a ramificação em cada nó.

Inicialmente, seja $\overline{z} = \infty$. Entretanto, se o tour (viável) $(1, 2, ..., n-1)$ tem $c_{12} + c_{23} + ... + c_{n1} < \infty$, então $\overline{z}$ pode ser complementado até ficar igual a esta quantidade. O valor inicial de $\underline{z}$ é determinado resolvendo o problema de alocação associado ao PCV (original). Se z^0 é o valor ótimo do problema de alocação, então$\underline{z} = z^0$ é um legítimo limite inferior inicial.

Naturalmente, se a solução associada a z^0 é um tour, terminam os cálculos. Do contrário, a solução dada consiste de pelo menos dois subtours.

Selecione o subtour com menor número de cidades. Faça com que tal subtour inclua as cidades $i_1, i_2, ..., \text{ e } i_k$. Então $x^{i_1 i_2} = x^{i_2 i_3} = ... = x^{i_k i_1} = 1$. A ramificação é determinada de forma que os problemas de alocação associados com os nós subseqüentes originários do nó corrente eliminarão este subtour. Isso pode ser efetuado escolhendo uma das variáveis$x^{i_1 i_2}, ..., x^{i_k i_1}$ igual a zero, o que gera k ramificações ou subproblemas. Para cada subproblema continuar

sendo um problema de alocação, a condição $x_{ij} = 0$ pode ser implementada fazendo $c_{ij} = \infty$ na matriz C, no nó imediatamente anterior.

O algoritmo exato é resumido como se segue:

Na t-ésima iteração (nó) seja z' o valor ótimo da função objetivo do problema de alocação associado. Pelo fato de que o limite inferior de z^* muda com o nó, z' vai automaticamente definir $\underline{z}$ no t-ésimo nó.(note que $\overline{z}$ é o mesmo para todos os nós).

PASSO 0 : Determine z^0. Se a solução associada é um tour, PARE;
Se não, faça $\overline{z} = c_{12} + c_{23} + ... + c_{n1}$ e tome $(1, 2, ..., n, 1)$ como o seu tour associado. Faça $t = 0$, e vá para o passo 1.
PASSO 1 : Selecione um subtour associado com z' que tenha o menor número de cidades e gere tantos ramos quantos forem o número de variáveis x_{ij} no nível que define o subtour. Para o ramo (i,j) defina uma nova matriz de custo (que difere da matriz do nó gerador), na qual $c_{ij} = \infty$. Faça $t = t + 1$, e vá para o PASSO 2.
PASSO 2 : Selecione um dos nós não ramificados. Se não restou nenhum, PARE. O tour associado com $\overline{z}$ é ótimo. Caso contrário, vá para o PASSO 3.
PASSO 3: Resolva o problema de alocação associado com o nó selecionado. Isso resulta num dos seguintes casos:

(i) Se $z' \geq \overline{z}$, então o nó corrente é cortado, uma vez que não pode gerar um tour melhor que o associado com $\overline{z}$. Faça $t = t + 1$, vá para o PASSO 2.
(ii) Se $z' < \overline{z}$ e a solução associada é um tour, então faça $\overline{z} = z'$ e faça o tour associado ser a melhor solução avaliada até o momento. Faça $t = t + 1$ e vá para o PASSO 2.
(iii) Se $z' < \overline{z}$, mas a solução associada não é um tour, então faça $t = t + 1$ e vá para o PASSO 1.

Exemplo: Considere o PCV cuja matriz C é indicada abaixo :

$$C = \begin{pmatrix} \infty & 2 & 0 & 61 & \\ 1 & \infty & 4 & 4 & 2 \\ 5 & 3 & \infty & 1 & 5 \\ 4 & 7 & 2 & \infty & 1 \\ 2 & 6 & 3 & 6 & \infty \end{pmatrix}$$

O valor inicial de $\overline{z}$ é: $\overline{z} = c_{12} + c_{23} + c_{34} + c_{45} + c_{51} = 10$ e o tour associado é dado por $(1, 2, 3, 4, 5, 1)$.

C^0 é igual à matriz de custo mínimo, ou seja, a matriz C acima.

Isso gera $z^0 = 8$ e a solução: $x_{12} = x_{21} = 1$ (subtour $(1, 2, 1)$, $x_{34} = x_{45} = x_{53} = 1$ (subtour $(3, 4, 5, 3)$, a qual consiste de dois subtours.

Como o subtour $(1, 2, 1)$ tem um menor número de cidades, ele é usado para a ramificação da árvore. Isso cria dois ramos, correspondentemente a $x_{12} = 0$ (ou, equivalentemente $c_{12} = \infty$ na matriz C^0) e o ramo $x_{21} = 0$ (ou, equivalentemente, $c_{21} = \infty$ na matriz C^0).

Selecionando, inicialmente o ramo associado com $x_{12} = 0$, o problema resultante gera $z^1 = 9$ com um tour solução dado por $(2, 1, 3, 4, 5, 2)$. Assim $\overline{z} = 9$ e o tour é anotado.

Agora, o ramo restante $x_{21} = 0$, onde C^2 é obtida a partir de C^0 determinando que $c_{21} = \infty$, o que nos dá $z^2 = 9$.

Como $z^2 = \overline{z}$, o nó 2 não pode fornecer uma solução melhor e, portanto, a busca termina.

Assim, a solução ótima é dada por $(2, 1, 3, 4, 5, 2)$ com $z^* = 9$.

Neste exemplo, a seleção do próximo nó a ser ramificado é arbitrária. Mas como ocorre com qualquer algoritmo B&B, ela é crucial. Infelizmente, não existe uma regra específica para selecionar o próximo nó já que o limite inferior é determinado somente por um modelo de alocação. Esse ponto é tratado no algoritmo seguinte, desenvolvendo-se uma estimativa mais simples de um limite inferior.

A razão para selecionar o subtour com o menor número de cidades é puramente intuitivo já que isso pode levar a gerar um número menor de ramos antes que o algoritmo termine. Geralmente, entretanto, isso pode não ser verdadeiro.

Experiência computacional com esse algoritmo e sua extensão foi relatada por D.Shapiro (1966), que lista consideráveis dificuldades, em geral, presumivelmente, porque um problema de alocação é otimizado em cada nó. Ele afirma, mais adiante, que uma matriz C, simétrica, apresenta uma particular

dificuldade em computação, principalmente porque o número de subtours incluindo 2 (duas) cidades é excessivo. Shapiro sugere um número de ramificações para este problema.

Um outro algoritmo baseado na solução de um modelo do problema de alocação em cada nó foi desenvolvido por Bellmore e Malone(1971). A ramificação em cada nó é baseada em eliminar subtours que criarão tantos novos ramos quantos forem os elementos em um subtour, mas as condições definindo cada ramo são diferentes. A experiência computacional relatada mostra que o algoritmo é bem sucedido em resolver problemas de pequeno tamanho ($n \leq 30$). Para problemas maiores, a razão do número de problemas resolvidos com sucesso entre aqueles tentados diminui com o aumento de n. Além disso, o tempo de computação aumenta num ritmo exponencial, com o aumento de n.

Outros algoritmos foram desenvolvidos por Held e Karp (1970,1971) para problemas simétricos ($c_{ij} = c_{ji}$). Problemas com $20 \leq n \leq 64$ foram tentados. Mas parece que o tempo de computação, como sempre, depende da informação, já que um dos problemas com 64 cidades, foi resolvido em menos de 3 minutos, enquanto que outro problema com 46 cidades levou cerca de 15 minutos.

6.2.6.2 - Algoritmo de Construção de um Tour

No algoritmo de eliminação de subtour, um modelo de alocação é resolvido para cada nó. Isso representa um custo muito grande sob o ponto de vista computacional. Nesse algoritmo, uma nova regra de ramificação é pesquisada e que não depende de resolver o modelo de alocação para cada nó. Um limite inferior do valor ótimo z^* do PCV é estimado usando a seguinte observação simples:

Seja $p_i = min_j\{c_{ij}\}$ e $q_j = min_i\{c_{ij} - p_i\}$, isto é, p_i é a entrada mínima da i-ésima linha da matriz C, e q_j é a entrada mínima da coluna j da matriz C após os p_i´s serem subtraídos. Defina

$$C'_{ij} = c_{ij} - p_i?q_j.$$

Pela própria construção do tour, temos que cij´ deve ser não-negativo para todos i e j.

Assim,

$$z = \sum_i \sum_j c_{ij}x_{ij} = \sum_i c'_{ij}x_{ij} + \sum_i p_i + \sum_j q_j \geq \sum_i p_i + \sum_j q_j$$

Isso significa que

$z = \sum_i p_i + \sum_j q_j$ é um limite inferior próprio de z.

O processo de ramificação é baseado na idéia da construção de tours, isto é, sucessivamente atribuímos o valor 1 para variáveis próprias x_{ij}. Isso deve ser feito garantindo-se que não haverá subtours criados nessa alocação. O procedimento seguinte alcança esse resultado. Uma variável é dita livre se ela não tem alocação binária na árvore. Para todo nó t, exatamente dois ramos são construídos. Um primeiro ramo é associado com $x_{kr} = 1$, onde:

x_{kr} é uma variável livre com $c_{kr} < \infty$. Um segundo ramo corresponde naturalmente a $x_{kr} = 0$. Essa condição vigora atualizando-se a matriz C' tal que $c_{kr} = \infty$.

Duas condições devem ser impostas na seleção de x_{kr} de modo a não formar subtours:

- Seguindo-se os ramos, começando no nó t de volta ao primeiro nó da árvore, as variáveis começando do 1 ao longo desses ramos não devem formar um subtour quando x_{kr} é igual a 1. Por exemplo, se $n > 3$ e se $x_{12=}1$ e $x_{23} = 1$ levam ao nó corrente, então $x_{31} = 1$ não é apropriado para a criação de um novo ramo.
- Para armazenar, $x_{kr} = 1$ é inaceitável para qualquer ramo alcançado por $x_{kr} = 1$ (caso contrário, um subtour (k, r, k) é criado). A matriz C', no nó t, é atualizada tomando $c_{rk} = \infty$.

Uma anotação sobre o limite inferior $\underline{z}$ é agora ordenado. Como a matriz C é substituída por C' no nó t e como o limite inferior é computado em C', é apropriado designar por $\underline{z'}$. Como os ramos levando ao nó t atribuem valor 1 para algumas das variáveis, isso é equivalente a eliminar algumas das linhas e colunas da matriz original C. Por exemplo, se $x_{ij} = 1$, então o tamanho da matriz C é reduzido pela eliminação da linha i e coluna j. Em geral, sejam $(i_1, i_2), ..., (i_{k-1}, i_k)$ as rotas retiradas para C'. Então p'_i e q'_j são relativamente definidas em relação a C'. Mas, como $x_{i_1 i_2} = ... = x_{j_{k-1} j_k} = 1$, então um limite inferior de z^* em t é dado por $\underline{z'} = (c_{i_1 i_2} + ... + c_{i_{k-1} i_k}) + \sum_i p'_i + \sum_j q'_j$ onde i e j são sobre as linhas e colunas não eliminadas de C'.

Os passos do algoritmo são os seguintes:

PASSO 0: Determine $\overline{z} = c_{12} + c_{23} + ... + c_{n1}$ e registre seu tour ssociado $(1, 2, ..., n, 1)$. Faça $t = 0$ e vá para o Passo 1.

PASSO1: Selecione uma variável livre x_{kr} com $c_{kr} < \infty$ e crie dois ramos a partir do nó t: um associado com $x_{kr} = 1$ e $c_{kr} = \infty$, e o outro associado com $x_{kr} = 0$, ou equivalentemente $c_{kr} = \infty$. A variável x_{kr} deve ser apropriada no sentido de que seu ramo não leve à formação de um subtour. Faça $t = t + 1$ e vá para o Passo 2.
PASSO 2: Selecione um dos nós não ramificados. Se não existir nenhum, pare; o tour associado com é ótimo. Caso contrário, vá para o Passo 3.
PASSO 3: Determine $\underline{z'}$.

(i) se $\underline{z'} \geq \overline{z}$, então o nó corrente é podado; Faça $t = t + 1$ e vá para o Passo 2.

(ii) se $\underline{z'} < \overline{z}$ e os ramos levando ao nó t gerarem um tour, Faça $\overline{z} = \underline{z'}$ e registre esse tour. Faça $t = t + 1$ e vá para o Passo 2.

(iii) se $\underline{z'} < \overline{z}$ mas o nó t não gera um tour, então, faça $t = t + 1$ e vá para o Passo 2.

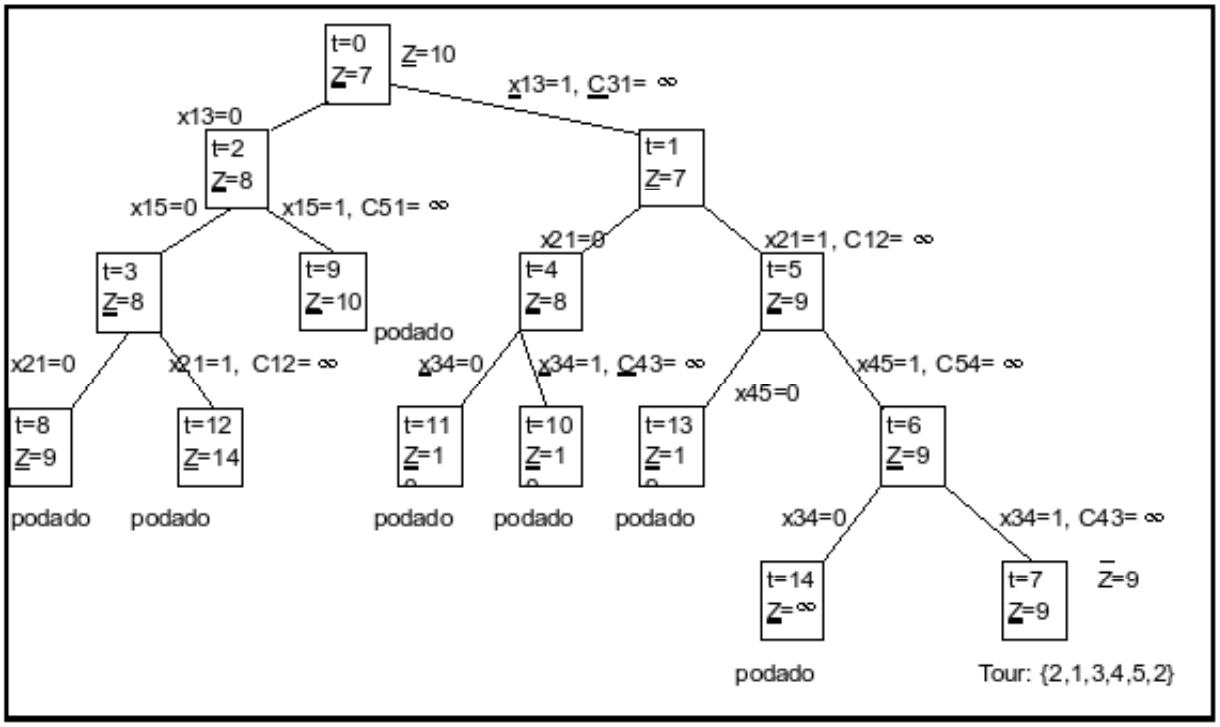
É importante notar que este algoritmo (quando comparado com o mostrado anteriormente) simplifica muito os cálculos em cada nó. Entretanto, esta simplificação depende de se produzir uma árvore de busca aumentada. Em particular, um tour é formado apenas após somar-se pelo menos $n - 1$ ramos sucessivos à árvore.

A regra usada para selecionar o próximo nó a ser ramificado (passo 2) é a seguinte: Quando os dois ramos de um nó corrente são formados, selecione o nó associado com o ramo gerando o menor limite inferior. O nó restante é armazenado numa lista na ordem em que é gerado. Se o nó selecionado é "esgotado", a lista armazenada é pesquisada numa base (o último fica dentro e o primeiro fica fora) (LIFO-last-in-first-out). A regra LIFO foi criada por Little et al. e provou ser eficaz em resgatar problemas armazenados de unidades de fitas de computador, rapidamente. Entretanto, este fator foi importante em 1963 e com o desenvolvimento das unidades de armazenagem em discos magnéticos cujo tempo de acesso é praticamente independente da ordem na qual os problemas estão armazenados, esta regra não é, hoje, de modo algum essencial. Em outras palavras, pode haver vantagens em considerar o nó com o menor limite inferior da árvore inteira, uma vez que isso poderia levar a uma solução (tour) melhorada, e assim aumentando a probabilidade de esgotar nós com limites inferiores grandes.

No exemplo seguinte, a regra LIFO não é utilizada. Ao contrário, o nó seguinte selecionado é o que tem menor $\underline{z}$ dentre todos os nós não-ramificados.

Exemplo 6.2.6.2.1 - Considere a matriz dada no exemplo anterior. A árvore-solução associada é dada na figura abaixo.

Inicialmente, $\overline{z} = 10$ corresponde a $x_{12} = x_{23} = x_{34} = x_{45} = x_{51} = 1$. O limite inferior $\underline{z}^0 = 7$ é computado como se segue.



Da matriz C, $p_1^0 = 0, p_2^0 = 1, p_3^0 = 1, p_4^0 = 1, p_5^0 = 2$. Isto leva à matriz "reduzida":

$$C^{ij} - p_i^0 = \begin{pmatrix} \infty & 2 & 0 & 6 & 1 \\ 0 & \infty & 3 & 3 & 1 \\ 4 & 2 & \infty & 0 & 4 \\ 3 & 6 & 1 & \infty 0 & \\ 0 & 4 & 1 & 4 & \infty \end{pmatrix}$$

Daí, temos que: $q_0^1 = 0, q = 2, q_3^0 = 0, q_4^0 = 0, q_5^0 = 0$. Logo,

$$\underline{z}^0 = \sum_{i=1}^{5} + \sum_{j=1}^{5} = 7$$

Isso significa que o ótimo z^* do PCV deve satisfazer $7 \leq z^* \leq 10$.

A seleção de x_{13} para ramificar em $t = 0$ é arbitrária. Mas, uma regra geral útil é selecionar uma variável livre "apropriada"tendo o menor c_{ij} dentre todas as livres.

Para ilustrar como $\underline{z}$ é computado em algum nó posterior, considere $t = 4$. Uma vez que $x_{13} = 1$, isto é equivalente a apagar a linha 1 e a coluna 3 da matriz C^0. Além disso, estabelecendo $c_{31} = \infty$ (isto é, bloqueando a rota 3,1), isso gera a matriz C^1 como abaixo:

$C^1 =$

	1	2	3	4	5
1					
2	1	∞		4	2
3	∞	3		1	5
4	4	7		∞	1
5	2	6		6	∞

A próxima matriz C^4 abaixo, é obtida de C^1 fazendo $c_{21} = \infty$. Assim, temos a matriz:

$C^4 =$

	1	2	3	4	5
1					
2	∞	∞		4	2
3	∞	3		1	5
4	4	7		∞	1
5	2	6		6	∞

onde p_1^4 e q_3^4 são indefinidos, $p_2^4 = 2, p_3^4 = 1, p_4^4 = 1, p_5^4 = 2, q_1^4 = 0, q_2^4 = 2, q_4^4 = 0, q_5^4 = 0$. Isso fornece:

$\underline{z}^4 = c_{13} + (p_2^4 + p_3^4 + p_4^4 + p_5^4) + (q_1^4 + q_2^4 + q_4^4 + q_5^4) = 0 + (2 + 1 + 1 + 2) + (0 + 2 + 0 + 0) = 8$

Uma solução viável é determinada em $t = 7$ quando se torna evidente que, estabelecendo-se $x_{52} = 1$, este nó vai gerar o tour $(2, 1, 3, 4, 5, 2)$ com um valor objetivo melhorado $z = 9$. Isso muda o limite superior para $\overline{z} = 9$. É interessante notar que 7 (sete) nós foram considerados antes de verificar-se que $t = 7$ gera o ótimo.

A simplicidade de computar limites inferiores $\underline{z}$ é alcançada às custas de aumentar-se a árvore. Basta ver que na figura anterior obtivemos uma árvore com muitos ramos. Desta forma, enquanto o tempo de computação consumido em otimizar o modelo de alocação em cada nó é reduzido drásticamente, isto pode ser equilibrado pelo aumento do número de nós. Além disso, poderia

parecer que este algoritmo pudesse representar um problema devido à excessiva necessidade de armazenamento no computador.

Little et al. relatam a solução de um problema de 42 cidades por seu algoritmo. Entretanto, isto não é informação suficiente para tirar conclusões generalizadas.

Tanto na eliminação de subtour quanto no algoritmo de construção de tour, a determinação de um bom limite superior inicial $\underline{z}$ é muito crucial em promover o esgotamento precoce de nós. Todavia, nenhum dos dois algoritmos fornece um método eficaz para determinar um bom limite superior inicial. Em outras palavras, um bom limite superior inicial só é produzido como um resultado secundário dos passos básicos do algoritmo.

A melhor maneira, até o momento disponível, para produzir um bom limite superior inicial é usar uma heurística. Lin (1965) dá um método aproximado que parece promissor e outros incluem Karg e Thompson (1964), Krolak et al. (1971), e Webb (1971). Nenhuma experiência computacional é relatada sobre o impacto de utilizar-se tais heurísticas com os algoritmos Branch and Bound anteriores, mas seria de se esperar que a combinação da heurística e métodos exatos levaria a um algoritmo improvável.

6.2.7. **Modelos Heurísticos para o Problema do Caixeiro Viajante.**

A seguir vamos descrever algumas heurísticas utilizadas para atacar o PCV, assumindo que todas as instâncias consideradas serão completas, simétricas e que a desigualdade triangular se aplica.

6.2.7.1 - Abordagens utilizando Heurísticas

Heurísticas são métodos que não garantem o fornecimento de soluções ótimas mas que, esperamos, possam produzir soluções razoavelmente boas pelo menos por uma boa parte do tempo. Para o PCV, existem dois diferentes tipos de heurísticas: a de **construção** e a de **melhoramento** . O primeiro tipo tenta construir um "bom"tour desde o início (do zero). O segundo tipo tenta aperfeiçoar um tour existente, através de melhorias "locais". Na prática parece muito difícil alcançar um método de construção realmente bom. Mas é do segundo tipo de método, em particular, através de um algoritmo desenvolvido por Lin e Kernighan [1973], que geralmente resulta nas melhores soluções e que forma a base da maioria dos códigos de computação eficazes.

Há 2(dois) teoremas que limitam o comportamento de qualquer heurística para o PCV. Dada uma heurística H e uma instância I do PCV, vamos denotar A(I) como o custo do circuito hamiltoniano obtido, no grafo, utilizando A e OPT(I) como o custo da solução ótima.

Teorema 6.2.1. *Assumindo que $P \neq NP$, nenhuma heurística de tempo polinomial pode garantir que $A(I)/OPT(I) \leq 2^{P(N)}$, para algum P fixo e qualquer instância I.*

Teorema 6.2.2. *Assumindo que $P \neq NP$, existe um $\varepsilon > 0$ tal que nenhuma heurística de tempo polinomial pode garantir que $A(I)/OPT(I) \leq 1 + \varepsilon$ para todas as instâncias I que satisfaçam a desigualdade triangular (na maioria das aplicações as distâncias satisfazem à desigualdade triangular).*

6.2.7.1 - Heurística do vizinho mais próximo

É a mais intuitiva das heurísticas que aqui vamos tratar.

Considere um grafo G com $\mid V \mid = n$ nós, $\mid E \mid = m$ arestas, completo e simétrico.O algoritmo do Vizinho Mais Próximo (NNA) para o PCV, de

modo a determinar um circuito hamiltoniano T de G é o seguinte: Comece em qualquer nó; visite o nó mais próximo ainda não visitado, e então retorne ao nó inicial quando todos os outros nós tiverem sido visitados. Temos então o seguinte:

(1) Escolha aleatoriamente um nó $v \in V(G)$;
(2) Inclua v no circuito T;
(3) Considere o nó u, mais próximo do nó v e ainda não incluído no circuito T. Inclua o nó u e a aresta (v, u) no circuito T e faça $v = u$;
(4) Se $\mid V(T) \mid \leq \mid V(G) \mid$, então vá para o PASSO 3;
(5) Se não, inclua a aresta entre o primeiro e o último dos nós incluídos.

O tempo computacional envolvido no algoritmo do vizinho mais próximo é de $O(n^2)$ para instâncias que satisfazem à desigualdade triangular.

Assim, $NN(I)/OPT(I) \leq (1/2)([logN] + 1)$.

Na prática, quando usamos o Algoritmo do Vizinho mais Próximo, quase sempre parece acontecer que o tour obtido inclui algumas arestas muito caras,. Johnson, Bentley, Mc Geoch e Rothberg [1997] relatam que em problemas da PCVLIB os custos médios dos tours encontrados pelo NNA são cerca de 1,26 vezes os custos dos correspondentes tours ótimos. Assim, para algumas aplicações, o NNA pode ser um método eficaz: ele é fácil de implementar, roda rápido, e geralmente produz tours de qualidade razoável.

Deve ser percebido, entretanto, que a estimativa de "1,26 do tempo ótimo"é uma observação empírica, o que não é uma garantia de desempenho. De fato, é fácil construir problemas com apenas quatro nós para os quais o NNA pode produzir um tour de custo arbitrariamente muitas vezes superior àquele de custo ótimo.

Para obter um limite bom, vamos assumir que os custos das arestas são não-negativos e que satisfazem à desigualdade triangular. Neste caso, Rosenkrantz, Stearns e Lewis[1977] mostraram que um tour do Vizinho mais Próximo nunca é mais que $\frac{1}{2}[log_2 n] + 1/2$ vezes o ótimo, onde n é a cardinalidade de V.

Este limite pode parecer muito fraco (particularmente quando comparado ao limite 1,26, observado nos problemas PCVLIB), mas Rosenkrantz, Stearns e Lewis[1977] provaram que não podemos fazer muito melhor que isso. Eles o demonstraram descrevendo uma família de problemas com custos não-negativos satisfazendo à desigualdade triangular e com uma quantidade arbitrariamente muito grande de nós, tal que o NNA pode produzir um tour de custo $1/3 \times log_2(n + 1) + 4/9$ vezes o ótimo. Este resultado mostra que se quisermos dar

um pior exemplo de limite, no desempenho desta heurística, o limite que vamos conseguir é tão ruim que não será de muito interesse prático.

As provas dos dois resultados acima são bastante técnicas e é recomendável sua verificação consultando a referência citada.

6.2.7.3 - Métodos de Inserção

Os Métodos de Inserção fornecem um conjunto diferente de heurísticas de construção de tours. Elas começam com um tour unindo dois dos nós, e então somam os nós restantes um a um, de tal forma que o custo do tour é aumentado numa quantidade mínima. Há muitas variações, dependendo de quais dos dois nós são escolhidos para começar e, o mais importante, qual nó é escolhido para ser inserido em cada etapa. Vamos descrever, ligeiramente, cada uma dessas variações:

6.2.7.3.1 - Inserção do mais Distante

Na prática, geralmente o melhor método de inserção é o de Inserção do mais Distante. Neste caso, começamos com um tour inicial passando através de dois nós que sejam as pontas de alguma aresta de alto custo. Para cada nó não-inserido v, vamos computar o custo mínimo entre v e qualquer nó construído até então. Então escolhemos como o próximo nó a ser inserido aquele para o qual este custo é máximo.

De início, isto pode parecer contra-intuitivo. Entretanto, na prática, geralmente funciona bem. Isso ocorre provavelmente porque a forma rígida do tour final a ser produzido é obtida razoavelmente cedo, e, em etapas posteriores, só modificações quase imperceptíveis são feitas.

6.2.7.3.2 - Inserção mais Próxima (heurística gulosa)

O método de **Inserção mais Próxima** é uma heurística que, a cada etapa, escolhe como próximo nó a ser inserido aquele para o qual o custo para qualquer nó do tour seja mínimo.

De maneira similar à heurística do vizinho mais próximo, esta heurística inclui uma nova aresta em relação às anteriores pertencentes ao circuito que está sendo construído a cada iteração.

Portanto, dado um grafo G completo e simétrico com $\mid V(G) \mid = n$ nós, $\mid E(G) \mid = m$ arestas e $C(\{e_j\})$ é o custo de cada aresta e_j, o algoritmo guloso para construir um circuito hamiltoniano T de G é o seguinte:

(1) Faça $V(T) = V(G)$;

(2) Seja e e(G) - E(T) tal que o custo C (ej) é mínimo e de tal modo que sua inclusão no circuito não implique na formação de um ciclo em T ou deixe algum nó de T com grau maior do que 2(dois);
(3) Inclua a aresta $\{e\}$ em $E(T)$;
(4) Se $\mid E(T) \mid < \mid E(G) \mid$, então vá para o Passo 2.

O tempo computacional envolvido no algoritmo guloso é de $O(N^2 logN)$, mais lento que o algoritmo NN, porém, na maioria dos casos a solução encontrada geralmente é melhor que a anterior, ou seja: $Greedy(I)/OPT(I) \leq (1/2)([logN] + 1)$.

6.2.7.3.3 - Inserção mais Barata (Método das Economias(savings))

Outra variante é a Inserção mais Barata. Neste caso, o próximo nó para inserção é aquele que provoca o menor aumento no custo do tour.

Esse método, originalmente proposto por Clarke e Wright (1964) para o Problema do Roteamento de Veículos, foi posteriormente utilizado para o PCV por Golden, Bodin, Doyle e Stewart(1980).

O procedimento inicia pela escolha arbitrária de um nó base do grafo completo e constrói ciclos de tamanho 2 deste nó a cada um dos $(n-1)$ restantes. A cada iteração, dois ciclos são combinados eliminando dois arcos (para o nó base e do nó base) e adicionando um arco de ligação. Os ciclos ou subtours são escolhidos para o processo de combinação de forma a maximizar a distância economizada em relação à configuração anterior. Repete-se o procedimento até obter um ciclo hamiltoniano. Assim, o algoritmo de Clarke e Wright, adaptado ao PCV, tem os seguintes passos:

(1) Selecionar qualquer nó como referência, digamos o nó 1.
(2) Obter $s_{ij} = c_{1i} + c_{j1} - c_{ij}, i, j = 2, 3, ldots, n$, onde s_{ij} é a economia alcançada se o nó i for ligado diretamente ao nó j.
(3) Ordenar as economias em uma lista de forma monótona decrescente.
(4) Percorrer seqüencialmente a lista de economias, iniciando no topo. Tentar a ligação correspondente ao primeiro s_{ij} da lista. Se a inserção da aresta $\{i, j\}$ e a retirada das arestas $\{1, i\}$ e $\{j, 1\}$ resultar em uma rota iniciando em 1 e passando pelos demais nós, eliminar s_{ij} da lista. Caso contrário, tentar a ligação seguinte na lista. Repetir até obter o ciclo hamiltoniano.

O algoritmo de Clarke e Wright é dominado pelo processo de construção da lista ordenada de economias e, portanto, terá complexidade $O(n^2.log_2 n)$. Com relação à qualidade da solução encontrada, podemos garantir que

$CW(I)/OPT(I) \leq [logN] + 1$, para instâncias que satisfaçam à desigualdade triangular. Se, com o intuito de eliminar a influência da escolha do nó base, repetirmos o procedimento para cada um dos $(n-1)$ nós restantes como ponto de referência, o algoritmo de Clarke e Wright passará a ser de ordem $O(n^3.logn)$.

6.2.7.4 - Heurística de Duplicação da Árvore Geradora de custo Mínimo (AGM)

Uma árvore geradora de um grafo G é um subgrafo conexo de G que possui todos os nós de G e não possui circuitos. Uma árvore geradora de custo mínimo (AGM) é aquela onde a soma do custo de suas arestas é mínima.

Uma (AGM) dá uma boa delimitação inferior para o valor ótimo do PCV métrico (PCVM). Vamos definir $C(S)$ como o custo de um dado subconjunto S de arestas de G. Se removermos uma aresta de um circuito hamiltoniano, temos uma árvore geradora de custo não superior ao do circuito. Portanto, ótimo $(G, C) \leq C(T)$.

A heurística de duplicação da AGM utiliza uma estratégia que utiliza-se de 3(três) passos, a saber:

(1) Construir uma AGMT de G;
(2) Duplicar as arestas de T obtendo um multigrafo P;
(3) Obter um circuito hamiltoniano a partir de P.

Para resolver o Passo 1 temos algoritmos eficientes, inclusive um algoritmo cuja complexidade é O(N2), onde N é o número de nós do grafo.

A resolução do Passo 2 é direta e simples, bastando criar, para cada aresta de T, duas arestas em P.

O Passo 3 da heurística é bem simples, pois no Passo 2 garantimos que P é um circuito euleriano (dobramos o grau de todos os nós, garantindo grau par para todos). O Passo 3 se resume no seguinte: percorrer os nós do circuito $P, (v_1, v_2, \ldots, v_m)$, na ordem em que aparecem em P e armazenar a seqüência de nós não repetidos. O seguinte procedimento ilustra esta idéia:

(1) $w_0 \leftarrow v_0$;
(2) $n \leftarrow 0$;
(3) para i de 1 até m faça
(4) se $v_i \notin \{w_0, \ldots, w_n\}$, então
(5) $n \leftarrow n + 1$
(6) $w_n \leftarrow v_i$.

Como o grafo é completo, a seqüência $(w_1, w_2, \ldots, w_n, w_1)$ é um circuito. E este circuito contém todos os nós do grafo, pois o circuito dado passa uma única vez por todas as arestas (é euleriano). E ainda, o circuito não possui nós repetidos, pois isto é garantido na condição da linha 4. Portanto, o circuito resultante é um circuito hamiltoniano.

Teorema 6.2.3. *O algoritmo Double MST é uma 2-aproximação polinomial para o PCV simétrico.*

Demonstração. Como P é um ciclo euleriano em $T+E(T)$, temos que $C(P) = 2C(T)$. Então, utilizando o fato de $OPT(G, C) \geq C(T)$ e que $C(C) \leq C(P)$ temos:

$$C(C) \leq C(P) = 2C(T) \leq 2OPT(G, C)$$

. A linha 1 do algoritmo consome $O(\mid V_G \mid^2)$, enquanto que as demais linhas consomem tempo $O(\mid V_G \mid)$. Logo, o algoritmo é de ordem polinomial.

□

6.2.7.5 - Heurística de Christófides

Vamos agora descrever mais um método de construção de tours, atribuído a Christófides [1976]. Esse método tem o melhor limite de pior caso dentre os métodos conhecidos. Ele sempre fornece uma solução de custo no máximo 3/2 vezes o ótimo (considerando que o grafo seja completo e os custos sejam não-negativos e satisfaçam à desigualdade triangular).

O algoritmo começa ao encontrar-se uma árvore geradora de custo mínimo T, do grafo G, usando, por exemplo, o algoritmo de Kruskal. As arestas em T serão usadas na busca de um tour ótimo.

Seja W o conjunto de nós que têm grau ímpar em T, e encontre um acoplamento perfeito M de $G[W]$, o subgrafo de G induzido por W, que é de custo mínimo com respeito a c.

Agora, seja J consistindo de $E(T) \cup M$, onde, se alguma aresta estiver tanto em T como em M, pegaremos duas cópias da aresta. Então, J é o conjunto de arestas de um grafo conexo com conjunto de nós V para o qual cada nó tenha grau par. Se todos os nós tiverem grau 2, então J é o conjunto de arestas de um tour e terminamos a busca. Se não, seja v qualquer nó de grau no mínimo 4 em (V, J). Então, há arestas $\{u, v\}$ e $\{v, w\}$ tais que se apagarmos estas arestas de J, e adicionarmos a aresta $\{u, w\}$ a J, então o subgrafo permanecerá conexo. Mais ainda, o novo subgrafo tem grau par a cada nó. (Isto ocorre porque o subgrafo induzido por J tem um tour Euleriano (escolhemos $\{u, v\}$ e $\{v, w\}$

para serem as arestas consecutivas do tour). Faça este "atalho" e repita o processo até que todos os nós sejam incidentes com duas arestas de J.

Teorema 6.2.4. *Suponha que temos um PCV com custos não-negativos e cujos elementos da matriz de custos satisfaçam à desigualdade triangular. Então, qualquer tour construído pela Heurística de Christófides tem custo no máximo* $3/2$ *vezes o custo de um tour ótimo.*

Demonstração. Seja H^* um tour ótimo. Ao remover qualquer aresta de H^* gera-se uma árvore geradora, e então o custo $c(T)$ de uma árvore geradora de custo mínimo T é, no máximo, $c(H^*)$. Podemos definir um circuito C no conjunto W de nós ímpares de T unindo esses nós na ordem em que eles aparecem em H^*. Note que se $\mid W \mid$ é par, o conjunto de arestas de C formam partições em dois acoplamentos perfeitos de $G[W]$. Como c satisfaz à desigualdade triangular, cada aresta dessas partições tem um custo inferior ao correspondente subcaminho de H^*. Portanto um desses acoplamentos tem custo de, no máximo, $c(H^*)/2$. Isso implica em que o custo do acoplamento perfeito de custo mínimo M de $G[W]$ seja no máximo $c(H^*)/2$. Assim, $c(J) \leq \frac{3}{2}c(H^*)$. Uma vez que cortar caminho só pode melhorar o custo $c(J)$, o tour final produzido também tem custo de no máximo $\frac{3}{2}c(H^*)$, como queríamos demonstrar.

□

Nos testes sobre problemas da PCVLIB por Johnson, Bentley, McGeoch e Rothberg[1997], a Heurística de Christófides produziu tours que eram aproximadamente 1,14 vezes o ótimo. Eles também fizeram a interessante descoberta de que, se a cada passo do atalho, o melhor atalho para o nó dado for o escolhido, então o desempenho do algoritmo melhora até $1,09$ o ótimo.

As heurísticas de construção são importantes, não somente pela sua perspectiva mas também porque podem ser usadas para gerar pontos de partida usados por outra classe de heurísticas, as de busca local. Os algoritmos de busca local são baseados em modificações simples no circuito. Dado um circuito hamiltoniano, esses algoritmos tentam fazer trocas para que seu comprimento seja reduzido, até que não seja possível reduzi-lo mais (ou seja: chegou-se a um tour localmente ótimo). A seguir, vamos descrever as heurísticas de busca local 2-Otimal e 3-Otimal, que são mais simples e famosas, além de argumentar sobre o que se sabe sobre elas de um ponto de vista teórico. Além de oferecerem bons

resultados, essas heurísticas são usadas por muitos pesquisadores como parte de outras técnicas usadas no PCV.

6.2.8. **Métodos de Melhoria de Tour: 2-otimal e 3-otimal.**

Há muitos métodos padronizados para tentar melhorar um tour T existente. O mais simples é o chamado 2-otimal. Ele funciona considerando cada par de arestas de T não-adjacente, por vez. Se estas arestas forem deletadas, então T se divide em dois caminhos: T_1 e T_2. Há somente uma maneira destes dois caminhos serem religados para formar um novo tour T' :

Se $c(T') < c(T)$, então substituímos T por T' e repetimos. Este processo é conhecido com dupla troca (2-troca). Veja Figura (1).

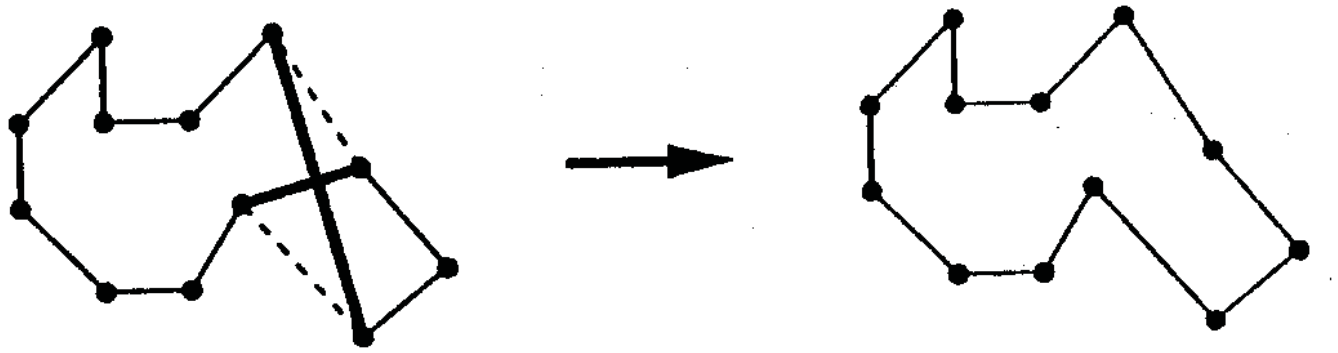


FIGURA 1. Dupla troca (2-troca).

Se $c(T') \geq c(T)$ para cada escolha de pares de arestas não-adjacentes, então T é 2-otimal e terminamos.

Para uma função de custo geral, de modo a checar se um tour é 2-otimal checamos $O(\mid V \mid^2)$ pares de arestas. Para cada par, o trabalho exigido para ver se a troca diminui o custo do tour pode ser computado em tempo constante. Assim, o tempo exigido para checar um tour quanto à 2-otimalidade é $O(\mid V \mid^2)$. Mas isso não significa que possamos transformar um tour em um tour 2-otimal em tempo polinomial. De fato, Papadimitrion e Steiglitz [1977] mostraram que se fizermos escolhas infelizes, podemos, em alguns casos, realizar um número exponencial de trocas, antes que um tour 2-otimal seja encontrado.

O algoritmo 3-otimal, proposto por Bock(1958), funciona quase da mesma maneira que o algoritmo 2-otimal. A primeira diferença ocorre no movimento básico, onde três arestas são apagadas, quebrando o circuito em três caminhos. Outra diferença está na reconexão destes caminhos, que pode ser feita de duas maneiras diferentes da qual estavam conectados. O movimento é realizado se

uma das duas maneiras diferentes reduz o comprimento do circuito. Se as duas maneiras reduzem o circuito, então opta-se por reconectar os caminhos da maneira que implique na maior redução possível. Além dos algoritmos 2-otimal e 3-otimal, temos o $k-$otimal que é uma generalização que permite trocas de k arestas.

Assim, o algoritmo 2-otimal pode ser generalizado naturalmente para um algoritmo $k-$otimal onde consideramos todos os subconjuntos de um conjunto de arestas de um tour de tamanho k, ou tamanho no máximo k, removendo cada subconjunto por e então verificamos se os caminhos resultantes podem ser religados de modo a formar um tour de custo menor. O problema é que o número de subconjuntos cresce exponencialmente com k e, rapidamente, chegamos num ponto de retorno decrescente. Por essa razão, $k-$otimal para $k > 3$ é raramente usado.

Johnson, Bentley, McGeoch e Rothberg [1997] relatam que em problemas da PCVLIB, o algoritmo 2-otimal produz tours de aproximadamente 1,06 do ótimo e o 3-otimal cerca de $1,04$ do ótimo.

6.2.8.1 - Avaliação teórica dos Métodos do tipo k-otimal:

A primeira questão a se preocupar envolve a qualidade dos circuitos obtidos através de um algoritmo de busca local. Para instâncias arbitrárias, esses algoritmos são limitados pelos teoremas citados anteriormente, mas algo mais forte pode ser dito: se $P \neq NP$, nenhum algoritmo de busca local que gasta tempo polinomial por movimento, pode garantir um limite superior constante para a razão entre o circuito encontrado e o circuito ótimo. Para os casos específicos do 2-otimal, 3-otimal e $k-$otimal ($k < 3n/8$), Papadimitriou e Steiglitz (1987) mostraram que existem instâncias que possuem um único circuito ótimo e muitos circuitos localmente ótimos, cada um distante do ótimo por um fator exponencial. Para um circuito de partida qualquer, a melhor performance garantida para o 2-otimal é uma razão de, no mínimo $(1/4)\sqrt{N}$, e para o 3-otimal é no mínimo $(1/4)\sqrt[6]{N}$. Generalizando, a melhor performance garantida para o k-otimal, assumindo a desigualdade triangular, é de no mínimo $(1/4)\sqrt[2k]{N}$. O lado positivo é que nenhum desses algoritmos produz uma razão pior que $4\sqrt{N}$. A situação fica um pouco melhor se restringirmos a atenção às instâncias onde as cidades são pontos em R para qualquer d fixo e as distâncias são computadas de acordo com alguma regra deste espaço.

Uma outra questão importante sobre os algoritmos de busca local envolve quantos movimentos eles podem fazer antes de atingirem uma solução localmente ótima. Para o 2-otimal e o 3-otimal esse número pode ser bem grande, mesmo assumindo que estamos diante da desigualdade triangular. Lueker (1976) mostrou que existem instâncias e circuitos de partida nos quais o algoritmo 2-otimal faz (2) movimentos. Um exemplo similar foi provado para o algoritmo 3-otimal por Chandra et al (1994) e estendido para o $k-$otimal para qualquer k fixo. Esses resultados mostram que não devemos tomar um circuito de partida ruim. Porém, para um k suficientemente grande, o problema de encontrar uma solução ótima local em uma vizinhança $k-$otimal é PLS-Completo como definido por Johnson et al (1988). Isso significa que o tempo gasto para encontrar um circuito localmente ótimo não pode ser polinomial, a menos que todos os problemas PLS sejam resolvidos em tempo polinomial.

Completando a discussão sobre tempo de processamento, temos que considerar também o tempo gasto em um movimento. Isso inclui o tempo gasto para encontrar um movimento que reduza o comprimento do circuito (ou verifique que não há), juntamente com o tempo gasto para realizá-lo. No pior caso, 2-otimal e 3-otimal requerem tempo $\Omega(N^2)$ e $\Omega(N^3)$, respectivamente, para verificar localidade ótima, assumindo que todos os movimentos possíveis devem ser considerados. Esses custos podem ser ainda maiores, dependendo das estruturas de dados utilizadas.

O comportamento empírico (baseado em resultados práticos) dos algoritmos de busca local contrasta nitidamente com o que é conhecido em teoria sobre eles. Isto ocorre porque muitos resultados teóricos são baseados no pior caso ou em casos muito distantes do caso médio.

Beardwood., Halton e Hammersley (1959), provaram que, para n grande, o comprimento do tour ótimo, considerando os n nós distribuídos aleatoriamente em uma região limitada do plano de área S, converge quase certamente para um valor proporcional a $\sqrt{SN}$. Experimentos computacionais empíricos realizados por Stein (1977) avaliaram os limites para a constante de proporcionalidade $k : 0.765 \leq k \leq (0.765 + 4/n)$.

Com base neste resultado conjectura-se que heurísticas com performance média satisfatória para o PCV exibem comportamento assintótico similar. Estudos estatísticos realizados por Ong e Huang (1987) utilizando as Heurísticas de Clark Wright, 3-otimal, do vizinho mais próximo e da inserção mais próxima, reforçam esta conjectura. Desta forma, o valor esperado de um tour produzido

por uma destas heurísticas em instâncias geradas com distribuição uniforme no quadrado unitário é da forma:

$$EH(n) = \beta\sqrt{n} + \phi(n)$$

onde $\phi(n)/\sqrt{n} \to +\infty$. Adicionalmente $EH(n)$ aproxima-se de uma função linear de para n grande. Os experimentos empíricos realizados por Ong e Huang obtiveram as seguintes estimativas para o valor de β:

Heurística	Estimativa de β
3-otimal	0, 7425
Clark Wright	0,7936
Inserção mais Próxima	0,8704
Vizinho mais Próximo	0,952

Os coeficientes de determinação foram superiores a 99.99%, com variância do comprimento dos tours relativamente insensível ao valor de n. Em particular, a estimativa para β foi muito próxima ao valor 0, 75 obtida nos estudos de Eilon, Watson-Gandy e Christfides (1971) para a heurística 3-otimal.

Estes resultados oferecem um recurso adicional para avaliação da performance média de heurísticas para o PCV.

6.2.9. **Heurística Grasp.**

GRASP (Greedy Randomized Adaptive Search Procedures) é uma meta-heurística iterativa para problemas combinatórios, onde cada iteração consiste basicamente em duas fases: construção e busca local. A fase de construção gera um circuito hamiltoniano, cuja vizinhança é investigada até que um mínimo local é encontrado durante a fase de busca local. A melhor solução até o momento é mantida como resultado. A condição de parada do algoritmo fica a critério do programador, podendo inclusive ser um número máximo de iterações.

Fase de Construção:

A cada iteração desta fase, considere o conjunto de elementos candidatos, formado por todos os elementos que podem ser incorporados à solução parcial em construção sem destruir a viabilidade. A seleção do próximo elemento é determinada pela avaliação de todos os elementos candidatos de acordo com uma função de avaliação gulosa. Essa avaliação cria uma lista restrita de candidatos (RLC) formada pelos melhores (aspecto guloso da heurística). O elemento a ser incorporado na solução parcial é selecionado randomicamente entre os elementos da lista (aspecto randômico da heurística). Depois que o elemento selecionado é incorporado à solução parcial, a lista de candidatos é atualizada e

os custos reavaliados (aspecto adaptativo da heurística). Esta estratégia é similar à da heurística semigulosa proposta por Hart e Shogan (1987), que também é uma aproximação iterativa baseada em construções gulosas randômicas, mas em busca local.

Fase de Busca Local:

As soluções geradas pela construção gulosa randômica não são necessariamente ótimas, até mesmo no que diz respeito à vizinhança simples. A fase de busca local geralmente melhora a solução construída. Um algoritmo de busca local funciona de modo iterativo trocando sucessivamente a solução corrente por uma solução melhor na sua vizinhança. Ele termina quando nenhuma solução melhor é encontrada. A eficácia de um procedimento de busca local depende de muitos aspectos, tais como: a estrutura da vizinhança; a técnica de busca utilizada; a eficiência da função de avaliação dos vizinhos e o percurso de partida. A fase de construção tem um papel importante em relação ao último aspecto, construindo um percurso de partida de boa qualidade.

Vizinhanças simples, como a 2-otimal, são geralmente usadas. A busca pode ser implementada usando a estratégia de melhor troca ou de primeira troca. Na prática, já foi observado que as duas estratégias rumam quase sempre para a mesma solução final, mas num tempo computacional menor quando a estratégia de primeira troca é utilizada.

6.2.9.1 - Regulagem de Parâmetros

Uma característica especial da heurística GRASP é a facilidade de implementação. Poucos parâmetros precisam ser inicializados ou ajustados. A heurística GRASP tem dois parâmetros principais: um relacionado ao critério de parada e outro relacionado à qualidade dos elementos na lista de candidatos.

O critério de parada pode ser um número máximo de iterações. Apesar da probabilidade de encontrar uma solução melhor que a atual reduzir-se a cada iteração, a qualidade da melhor solução encontrada só pode melhorar. Como a computação não varia muito de iteração para iteração, o total é previsível e aumenta linearmente com o número de iterações. Conseqüentemente, quanto mais iterações, maior o tempo de computação e melhor a solução encontrada.

Para a construção da lista RCL usada na fase de construção, vamos considerar $c(e)$ o custo de adicionar o elemento $e \in E$ na solução em construção. Em qualquer iteração da heurística GRASP, temos que c_{min} e c_{max} são, respectivamente, o menor e o maior custos. A lista RCL é composta dos elementos $e \in E$ com menores custos. Esta lista pode ser limitada pelo número de elementos

ou pela sua qualidade. No primeiro caso, a lista é composta dos p elementos de menor custo, onde p é um parâmetro. Podemos também, associar a lista RCL com um parâmetro $\alpha \in [0, 1]$. A lista é composta por todos os elementos $e \in E$ que podem ser inseridos na solução parcial sem destruir a viabilidade e cuja qualidade é superior ao valor limiar, ou seja,

$$c(e) \in [c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$$

Um valor $\alpha = 0$ corresponde a um algoritmo guloso puro, enquanto $\alpha = 1$ é equivalente a uma construção randômica.

6.2.10. **Métodos de Melhoria de Tour: Lin-Kernighan.**

Lin e Kernighan [1973] desenvolveram uma heurística que funciona extremamente bem na prática. Ela é basicamente um método k-otimal com duas características originais: Primeiramente, o valor de k pode variar. Em segundo lugar, quando uma melhoria é obtida, não é necessariamente usada de imediato. Ao invés disso, a busca continua com a esperança de encontrar uma melhoria ainda maior. Para descrevê-la, precisamos de várias definições que passaremos a descrever a seguir:

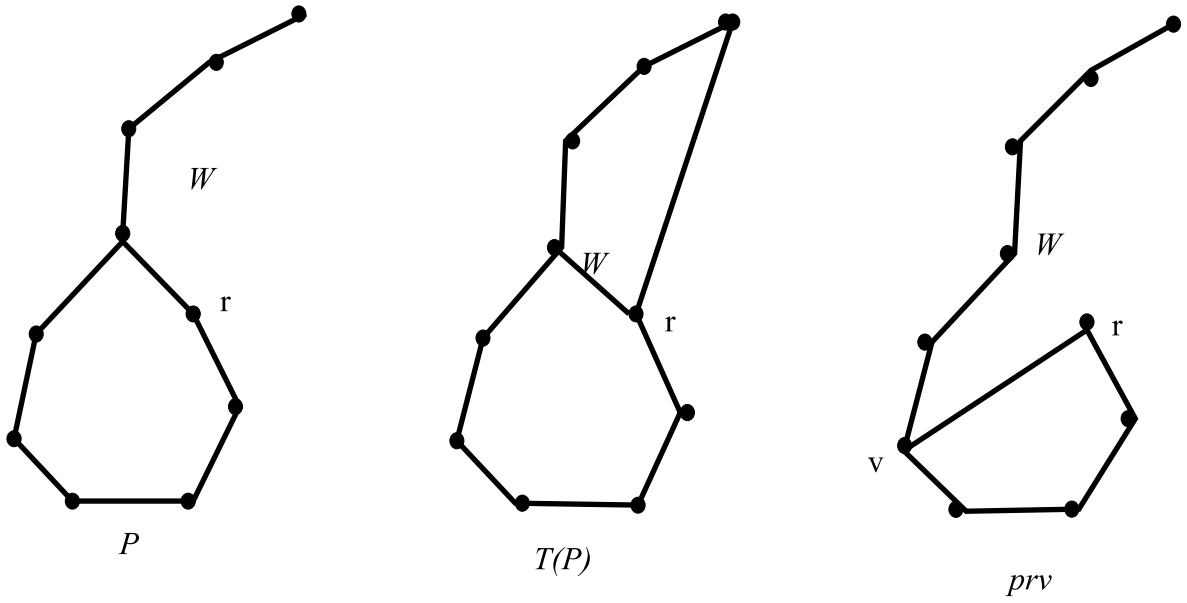


FIGURA 2. Um $\delta-$caminho

Definição 6.2.10.1 - Um $\delta-$caminho num grafo G com n nós é um caminho contendo n arestas e $n + 1$ nós dos quais todos são distintos, exceto o último, que aparecerá em algum lugar mais cedo no caminho.(O nome vem da forma do caminho).Veja a figura abaixo:

Repare que um tour é um $\delta-$caminho para o qual o último nó é o mesmo que o primeiro nó. Se P é um $\delta-$ caminho que não seja um tour, então

podemos obter um tour $T(p)$ como se segue: Seja w o último nó do caminho (que também aparece mais cedo no caminho). Seja $\{w, r\}$ a primeira aresta do sub-caminho de P entre duas ocorrências de w. Removendo-se a aresta $\{w, r\}$ e adicionando-se a aresta unindo r ao primeiro nó do caminho, nós obtemos o conjunto de arestas de um tour. Veja a figura acima.

Suponha que P seja um d-caminho que não seja um tour. Novamente, seja w o último nó e seja $\{w, r\}$ a primeira aresta do sub-caminho de P entre duas ocorrências de w. Se removermos $\{w, r\}$ obteremos o conjunto de arestas de um caminho terminando em r. Se então adicionarmos mais uma aresta $\{r, v\}$ e o nó v, obtemos um novo $\delta-$caminho P terminando em v. Chamamos esta operação de $rv-$troca. Note que $c(P^{rv}) = c(P) + c_{rv} - c_{wv}$. Novamente, veja a figura acima.

A heurística de Lin-Kernighan começa com um tour e então constrói uma seqüência de d-caminhos não-tours, cada um deles obtido do antecedente por uma $rw-$troca. Para cada $\delta-$caminho P assim produzido, computa-se o custo de $T(P)$. Se este for melhor que o melhor tour conhecido, então ele é "lembrado". Quando a checagem está completa, ele substitui o tour inicial pelo melhor tour encontrado na checagem. Uma descrição completa da essência da heurística de Lin-Kernighan é dada a seguir:

Passo1: volta ao tour/pares de arestas]. Para cada nó v de G, para cada uma das duas arestas $\{u, v\}$ de T incidentes em v, por cada vez, execute os Passos 2 a 5 numa tentativa de obter uma melhora. Este processo é chamado de checagem de aresta.

Passo 2:[inicializa a checagem de aresta]. Inicialmente, o melhor tour encontrado é T. Seja $u_0 = u$. Remova a aresta $\{u_0, v\}$ e adicione uma aresta $\{u_0, w\}_0$, para algum $w_0 \neq v$, verificando que tal w_0 possa ser encontrado para o qual $cw_0u_0 \leq cu_0v$. Se tal w_0 não puder ser encontrado, então esta checagem está completa e seguimos adiante para o próximo par de nós/arestas.

Agora temos um $\delta-$caminho P^0 (com última aresta $\{u_0, w_0\}$) e $c(P^0) \leq c(T)$. Faça $i = 0$ e vá para o Passo3.

Passo 3: [tour de teste]. Construa o tour $T(P')$. Se $c(T(P'))$ for menor que o custo do melhor tour até então encontrado, então armazene este como o novo melhor tour encontrado até então. Em qualquer um dos casos siga para o próximo passo.

Passo 4:[construa o próximo δ-caminho]. Seja u_{i+1} o vizinho de w_{i+1} em P^i que pertence ao subcaminho unindo w_i a u_i . Se a aresta $\{w_i, u_{i+1}\}$ era

uma aresta somada a um δ-caminho nesta iteração, então vá para o Passo 5 e pare esta checagem. Se não, tente encontrar um nó w_{i+1} tal que $\{u_{i+1}, w_{i+1}\}$ não esteja em T e quando realizamos a troca $u_{i+1}w_{i+1}$, o novo δ-caminho P^{i+1} , com última aresta $\{u_{i+1}, w_{i+1}\}$, obtemos um custo não superior àquele de T. De novo, se tal w_{i+1} não puder ser encontrado, iremos para o Passo 5 e paramos esta checagem. Mas, se formos bem-sucedidos, então façamos $i = i+1$ e voltamos para o Passo 3. (Veja a figura abaixo)

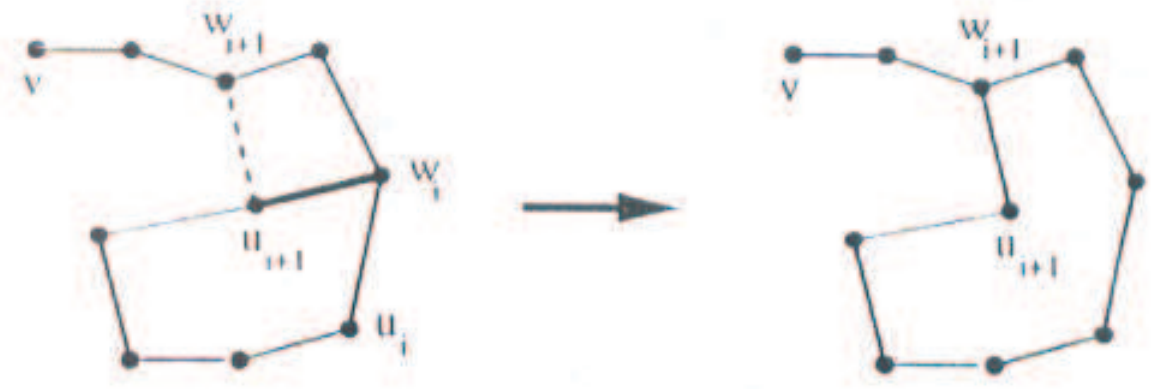


FIGURA 3. Construção do próximo $\delta-$caminho

Passo5:[fim da checagem de nó/aresta]. Se tivermos encontrado um tour cujo custo seja menor que aquele de T, substituímos T por tal tour de custo mínimo encontrado. Se ainda houver combinações de nós/arestas não-testadas, voltamos ao Passo 2 e tentamos a próxima.

Agora, vamos fazer alguns comentários:

Primeiramente, repare que no processo de checagem de um único nó/aresta, para qualquer aresta dada, podemos adicioná-la a um δ-caminho ou removê-la de um δ-caminho, mas não podemos fazer as duas coisas. Assim, faz sentido falar de arestas adicionadas ou removidas. Cada sucessivo δ-caminho gerado terá custo não superior àquele de T, o tour inicial. Isto equivale a dizer que a soma dos custos das arestas removidas menos a soma do custo das arestas adicionadas mantém-se não-negativa. Esta diferença é, às vezes, chamada de soma de ganhos.

Note que quando um melhor tour $T(P^j)$ é encontrado, não abandonamos de imediato o processo. Ao contrário, continuamos buscando uma finalização ainda melhor.

Nos Passos 2 e 4 escolhemos um nó w_{i+1}, sujeito a certas condições. Em geral, há muitas escolhas possíveis para esses nós. Pode consumir um tempo demasiado longo tentar todas as possibilidades a cada etapa. Então, Lin e Kernighan

sugerem a seguinte concessão, com a intenção de limitar a quantidade de retrocessos pelo mesmo caminho:

Para cada candidato w, compute $l(w) = c_{wu_{i+2}} - cu_{i+1w}$, onde u_{i+2} é o nó que seria selecionado na vez seguinte, ao longo dessa etapa. Repare que u_{i+2} é completamente determinado. Quando escolhemos cada um dos w_0 e w_1 , consideramos cada um dos cinco candidatos w para os quais $l(w)$ é o máximo por vez. Para todas as iterações subseqüentes, consideramos apenas o melhor candidato. Assim, no processo de checagem associado a um só par de nós/arestas, iremos na verdade considerar exatamente 25 escolhas para as duas primeiras arestas a serem ligadas. Para cada uma, seguimos completamente sua cadeia de trocas. Se um tour melhor for encontrado, então T é substituído e, começamos de novo. Se não, passamos ao seguinte.

Lin e Kernighan recomendam uma modificação adicional à questão discutida acima. A primeira vez que executamos o Passo 4, quando escolhemos a aresta $\{u, i\}$, consideramos uma segunda alternativa. Esta é o vizinho de w_o no caminho de volta a v. Agora, remover a aresta $\{u_i, w_o\}$ dá origem a um circuito e a um caminho unindo v e u_i. Ao ligar u_i a um nó w_1 no circuito, obtemos uma vez mais um δ-caminho começando em v (podemos orientar o δ-caminho em qualquer uma das direções em volta do circuito). Escolhemos a melhor w , de acordo com os critérios acima. Veja a figura (a) abaixo.

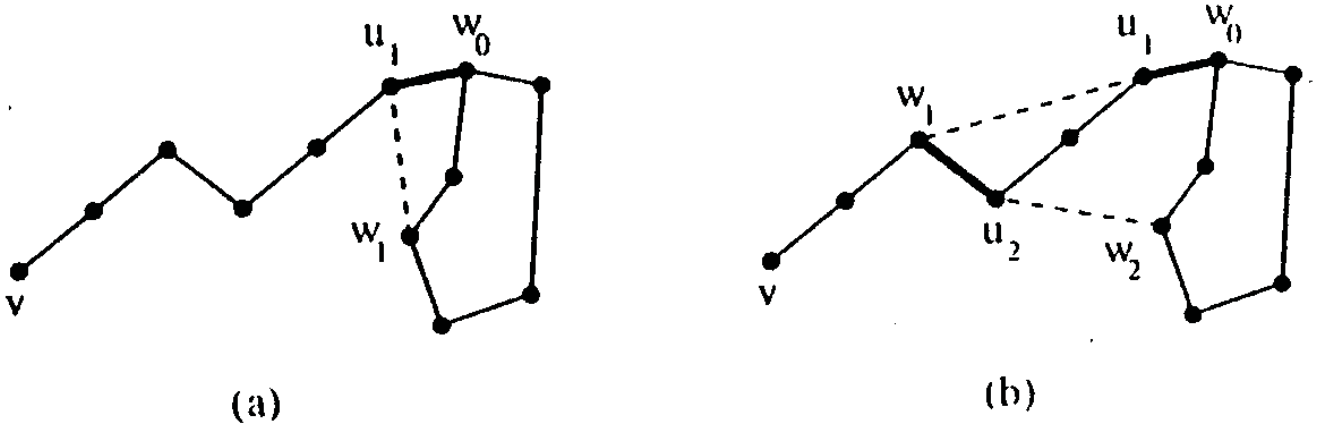


FIGURA 4. Primeiro passo.

Avançando um passo nessa situação, Lin e Kernighan também permitem que w_1 seja um nó no caminho unindo v e w_1 (em vez de ser no circuito). Neste caso, consideremos u2 o primeiro nó no sub-caminho de w_1 a u_2 , apagamos a aresta $\{w_1, v_2\}$, e formamos um δ-caminho considerando w_2 um nó no circuito. Veja a figura (b) acima.

Isso completa a descrição do algoritmo principal. Há um leque amplo de modificações possíveis na essência do algoritmo que podem ser consideradas. Algumas variantes interessantes são descritas em Johnson e McGeoch [1997], Mak e Morton [1993], e Reinelt [1994].

Para produzir um tour de excelente qualidade precisamos inserir o algoritmo principal num procedimento de busca mais amplo. Lin e Kernighan propõem rodar o principal repetidamente, começando de muitos tours diferentes. Eles também propõem diversos métodos diferentes para reduzir a quantidade total de trabalho exigido pelo fato de ter que voltar a rodar inúmeras vezes a rotina principal.

Uma alternativa foi proposta por Martin, Otto e Felten [1992] que parece funcionar muito bem na prática. Cada vez que completamos uma rodada da rotina principal e desta forma temos um tour "ótimo local´´ T, aplicamos a ele um "passo inicial´´ que vai perturbar o tour de forma que ele provavelmente não será mais ótimo local. Rodamos então, de novo, a rotina principal a partir desse novo tour. Se a rotina principal produz um novo tour T^* que é mais barato que T, então substituímos T por T^*, e repetimos o processo do "passo inicial"com este novo tour. Do contrário, voltamos e repetimos o processo com nosso melhor tour T.

Um "passo inicial´´ que eles propõem é uma 4-troca para a qual a rotina principal é incapaz de realizar. Ela consiste em escolher aleatoriamente quatro arestas não-adjacentes $\{u_0, v_0\}, \{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}$ do tour onde consideramos que os nós apareçam na ordem acima, no tour. Removemos estas arestas e adicionamos as arestas $\{u_0, v_2\}, \{u_1, v_3\}, \{u_2, v_0\}, \{u_3, v_1\}$. Veja a figura abaixo. Esse se torna o nosso novo tour inicial. Seu custo provavelmente será muito pior que aquele do antigo ótimo local, mas ele realmente fornece um novo ponto inicial para rodarmos de novo a rotina principal.

Este procedimento é chamado de Lin-Kernighan **Encadeado**. Martin e Otto [1996] descrevem-no num contexto geral para procedimentos de busca para otimização combinatória. Um ponto que eles levantam é de que pode ser útil substituir T por T^* esmo quando T^* for ligeiramente mais caro que T. Esta flexibilidade, acrescida, pode permitir ao procedimento escapar de um **tour ótimo local** que não parece permitir bons passos iniciais. A regra que eles sugerem é substituir T por T^* com uma certa probabilidade que depende da diferença nos custos dos dois tours e do número de iterações do procedimento que eles até o momento já desenvolveram.

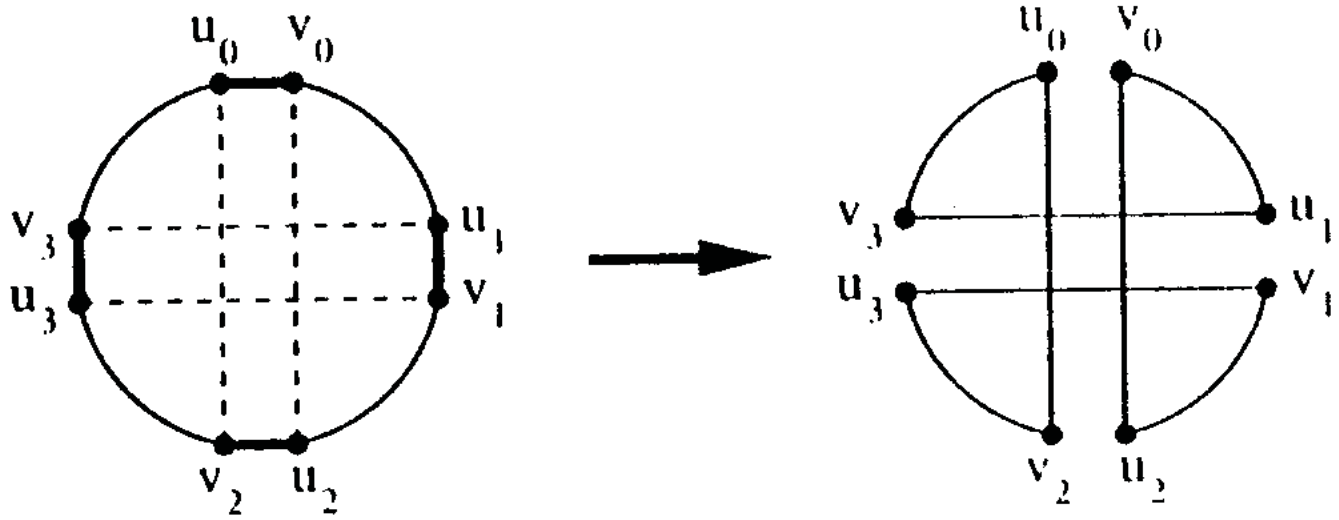


FIGURA 5. Troca para reiniciar a questão central

Note que o procedimento de Lin-Kernighan **Encadeado** não é um algoritmo finito, uma vez que não foram dadas quaisquer regras para parar. Normalmente iríamos deixa-lo rodar até verificarmos um longo período sem melhorias. Às vezes, entretanto, temos disponível um bom limite **mais baixo** no custo ótimo de solução , o que irá permitir parar mais cedo. Obter tais limites é o assunto das próximas duas seções.

Em problemas da PCVLIB, Johnson, Bentley McGeoch e Rothberg [1997] relatam que o Método de Lin-Kernighan produz tours aproximadamente $1,02$ vezes o ótimo, e o procedimento de Lin-Kernighan Encadeado abaixo de $1,01$ o ótimo.

6.2.10.1 - Tempos Computacionais

Todos os métodos descritos acima podem ser implementados para rodar eficazmente, mesmo em problemas bastante grandes. Para uma extensa abordagem deste assunto veja em Johnson, Bentley McGeoch e Rothberg [1997]. Eles relatam, por exemplo, que num problema Euclidiano de 10.000 nós, gerados aleatoriamente, os tempos computacionais numa unidade de trabalho rápida (considere rápida para a época: 1991), são estes: $0,3$ segundos para Vizinho Mais Próximo, $7,0$ segundos para Inserção Mais Distante, $41,9$ segundos para Heurística de Christófides, $3,8$ segundos para 2-otimal, $4,8$ para 3-otimal, e $9,7$ segundos para Lin-Kernighan.

6.2.10.2 Limites Inferiores

Anteriormente já enfatizamos o uso de relações min-max em procedimentos para computar soluções ótimas em problemas combinatórios. O limite inferior fornecido pelo lado "max´´ da relação dá uma prova da otimalidade da solução dada ao lado "min´´. Infelizmente, para o PCV e muitos outros problemas conhecidos por serem tão difíceis quanto o PCV, nenhuma relação do tipo min-max é conhecida. Entretanto, é importante em algumas situações práticas dar limites inferiores como uma medida da qualidade de uma solução proposta.

6.2.10.2.1 O limite de Held e Karp

Uma abordagem clássica dos limites inferiores para o PCV envolve a computação de árvores geradoras de custo mínimo. A técnica geral é padrão: Para obter um limite inferior para um problema difícil afrouxamos suas restrições até chegarmos a um problema que saibamos resolver eficazmente. Neste caso, a idéia é de que se removemos de um tour dado, as duas arestas incidentes em um nó particular, então teremos um caminho correndo através dos nós restantes. Embora em geral não saibamos como computar tal "caminho gerador"de custo mínimo (ele é tão difícil quanto o PCV), o que sabemos realmente é como computar uma árvore geradora de custo mínimo, e isto nos dará um limite inferior do custo do caminho.

Mas, precisamente, suponha que tenhamos um grafo $G = (V, E)$ com custos de arestas $(c_l : c \in E)$ e um tour $T \subseteq E$. Seja $v_1 \in V$, sejam $\{e\}$ e $\{f\}$ duas arestas em T que são incidentes com v_1 , e seja P o conjunto de arestas do caminho obtido removendo-se $\{e\}$ e $\{f\}$ de T. O custo de T pode ser escrito como $c_l + c_f + c(P)$. Assim, se temos números A e B tais que $A \leq c_l + c_f$ e $B \leq c(P)$, então $A + B$ serão um limite inferior no custo do tour T. Nosso objetivo é definir A e B de forma que eles sejam válidos para todas as escolhas de T. Deste modo, obteremos um limite inferior para todos os tours.

Então, o que podemos dizer sobre as arestas e e f ? Uma vez que tudo o que sabemos é que ambas são incidentes com v_1 , não podemos fazer nada mais do que estabelecer A igual à soma dos custos das duas arestas mais baratas em E incidentes com aquele nó.

A parte interessante é o limite em $c(P)$. Note que P é uma árvore geradora (embora de uma forma especial) para o grafo $G - v_1$ que conseguimos apagando

v_1 e as arestas incidentes de G. Assim, se B for o custo mínimo de uma árvore geradora em $G - v_1$ (que podemos computar com os métodos descritos anteriormente), então saberemos que P deve ter um custo de, no mínimo, B. Então temos nosso limite, $A + B$. Isto é comumente chamado o limite de $1-$árvore , e um conjunto de arestas consistindo de duas arestas incidentes com o nó v_1 mais uma árvore geradora de $G - v_1$ é chamada uma $1-$árvore. O nome vem da prática comum de denominar o nó que apagamos como nó v_1 ou nó "1´´. Podemos resumir esta discussão como se segue:

Limite de $1-$ árvore

Seja $G = (V, E)$ com custos de arestas $(c_l : e \in E)$ e seja $v_1 \in V$. Agora seja $A = min\ \{c_l + c_f : e, f \in \delta(v_1), e \neq f\}$ e seja B o custo de uma árvore geradora mínima em $G - \{v_l\}$. Então $A + B$ é o limite inferior para o PCV em G.

significativo. A chave para obter-se uma real melhoria pode ser encontrada examinando-se a estrutura da 1-árvore. Muitos nós não têm grau 2. Com isso, podemos tirar proveito.Para ver com clareza o que está acontecendo, considere o pequeno exemplo dado na figura abaixo:

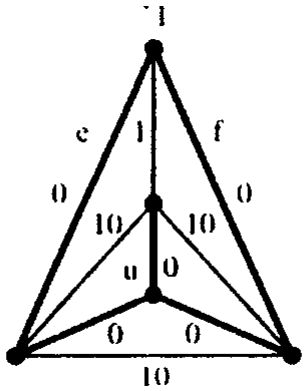


FIGURA 6. Uma $1-$árvore otimal

Com v_1 escolhido como foi indicado, o limite da $1-$árvore é 0. Como pode ser visto, entretanto, o tour mais barato tem custo 10. O que deu errado foi que a árvore geradora em G^* pode usar todas as três arestas de custo 0 incidentes com o nó u, enquanto um tour pode apenas fazer uso de duas delas. Há um modo de contornar este problema, embora a princípio possa parecer um truque de mágica.

O que aconteceria se adicionássemos 10 ao custo de cada uma das arestas incidentes com o nó u? Todo tour usa exatamente duas dessas arestas, então o custo de cada tour aumenta até precisamente 20. Assim, no que diz respeito ao PCV, realmente não mudamos nada. O antigo tour ainda é ótimo, sendo

seu custo, agora, de 30. Mas o que aconteceu ao limite da 1−árvore ? Um simples cálculo mostra que ele também tem valor 30. Então, temos uma prova simples de que nenhum tour no grafo alterado pode custar menos que 30. Isto significa que nenhum tour no grafo original pode custar menos que 10. Com esta simples transformação aumentamos, portanto, o limite inferior de 0 para 10. O ponto é que, embora a transformação não altere o PCV, ela na verdade altera fundamentalmente a determinação da árvore geradora mínima.

No exemplo acima, dizemos que "atribuímos ao nó u o número -10´´. Isto é, nos referimos ao processo de subtrair k do custo de cada aresta incidente com um dado nó v ao atribuir ao nó v o número k. Note que poderíamos atribuir diversos números ao nó, de uma só vez. A mudança no custo de qualquer tour será apenas duas vezes a soma dos números atribuídos. Com este fato, podemos formalmente mencionar uma técnica de limite inferior introduzida por Held e Karp [1970]:

O limite de Held Karp : Seja $G = (V, E)$ um grafo com custos de arestas $(C_l : e \in E)$, seja $v_1 \in V$, e para cada vértice $v \in V$, seja $y\nu$ um número real. Agora, para cada aresta $e = uv \in E$ seja $\bar{c}_l = c_l - y_\mu - y_\nu$ e seja C o limite da 1−árvore para G com respeito ao custo das arestas $(\bar{c}_l : l \in E)$. Então $2\sum(y_v; \nu \in V,) + C$, é um limite inferior para o PCV em G (com respeito aos custos da aresta original $(c_l : l \in E)$.

Com um bom conjunto de números de nós, a diferença entre o limite inferior de Held-Karp e o limite inferior da 1-árvore pode ser dramática.

Para a mais importante questão computacional, que é a de como encontrar um bom conjunto de números, Held e Karp [1971] propuseram um simples esquema iterativo. O passo principal é o seguinte. Suponha que tenhamos computado uma 1-árvore ótima T, no que diz respeito aos custos de aresta alterados $(c_{\mu\nu} - y_\mu - y_\mu : v \in V)$ para algum conjunto de números de nós $(y_\mu : v \in V)$. Para cada nó $v \in V$, seja $d_T(v)$ o número de arestas de T que são incidentes com ν. Baseado em nossa discussão anterior, se $d_T(v)$ é maior que 2 então deveremos diminuir y_ν e se $d_T(\mu)$ for igual a 1 (não pode ser menor que 1) devemos aumentar y_ν . Isto é simplesmente o que Held e Karp nos indicam a fazer: Para cada nó μ, substitua y_ν por $y_\nu + t(2 - d_T(v))$ para algum número real positivo t (o tamanho do passo).

Iterando este passo, obtemos uma seqüência de limites inferiores de Held-Karp. Embora não seja verdade que o limite melhora a cada iteração, sob certas condições naturais na escolha dos tamanhos dos passos pode ser demonstrado

que o limite irá convergir ao limite ótimo de Held-Karp (isto é, o máximo limite Held-Karp sobre todas as escolhas de números de nós). (Veja Held, Wolfe e Crowder (1974). Infelizmente, nenhuma condição garante que a convergência ocorrerá em tempo polinomial. Mas nem tudo está perdido. Como relatado em Grötschel e Holland (1988), Holland (1987), Smith e Thompson (1977), e em outros, diversas maneiras de selecionar os tamanhos dos passos mostraram bom desempenho na prática. Vamos descrever um desses métodos.

Motivado por considerações geométricas, na k-ésima iteração, Held e Karp (1971) sugerem o tamanho do passo: $t^{(k)} = \alpha^{(k)}(U - H)/\sum((2 - d_T(v))^2 : v \in V)$ onde u é algum valor-meta (um limite superior no custo de um tour mínimo), H é o atual limite de Held-Karp, e $\alpha^{(k)}$ é um número real satisfazendo $0 \leq \alpha^{(k)} \leq 2$. Seguindo Held, Wolfe e Crowder (1974), começamos com $\alpha^{(0)} = 2$ e diminuímos $\alpha^{(k)}$ por algum fator fixo após cada bloco de iterações, onde o tamanho de um bloco depende do número de nós no PCV que estamos resolvendo e da quantidade do tempo de computação que estamos pretendendo gastar para obter o limite inferior.

O método inteiro está resumido na caixa abaixo.

Método Iterativo Held-Karp
Entrada
Grafo: $G = (v, E)$ com custos de arestas $(c_l : l \in E)$ e $v_1 \in V$
Número real: U (uma valor-meta)
Número real positivo: ITERATIONFACTOR (por exemplo, 0.015)
InteIRO positivo: MAXCHANGES POR EXEMPLO, 1000
Inicialização
$y_v = 0$ para todo $v \in V$, $H^* = -\infty$, TSMALL= 0.001, $\alpha = 2, \beta = 0.5$
NUMITERATIONS=ITERATIONFACTOR $x/V/$
Algoritmo
Para $i = 1$ to MAXCHANGES
Para $K = 1$ TO numiterations
Seja T o ótimo da $1-$árvore com respeito ao custo das arestas
$(c_{uv} - y_u - y_v : uv \in E)$
e seja H o correspondente limite de Held Karp;
Se $H \succ H^*$, então faça $H = H^*$
Se T é um tour, então pare.
Seja $t^{(k)} = \alpha(U - H)/\sum((2 - dT(v))^2 : v \in V$
Se $t^{(k)} \prec TSMALL$, então pare.Substitua y_v por $y_v + t^{(k)}(2 - dT(v))$
para todo $v \in V$: Substitua α por $\beta\alpha$;

Se fizermos experimentações com esse método, vamos reparar que o limite que se obtém para um dado grafo e custos de arestas vai depender das escolhas dos parâmetros introduzidos (particularmente o valor alvo U). Somente experimentando em diferentes ambientes é que vai ser possível otimizar o método para uma particular classe de problemas. Também, deve-se ter em mente que muitas outras escolhas para o tamanho do passo são possíveis, e os experimentos podem sugerir um esquema alternativo que funcione melhor para cada tipo de problema.

6.2.11. **A relaxação Lagrangeana/surrogate aplicada ao PCV.**

A relaxação lagrangeana é uma técnica bem conhecida e usada frequentemente na obtenção de limitantes para problemas de otimização combinatória (veja em Parker(1988) e Shapiro(1971)) . Held e Karp (1970-1971) aplicaram com sucesso a relaxação lagrangeana ao PCV no início da década de 70. O limite da relaxação aproxima o limite conhecido atualmente como HK (Held e Karp),

um limite muito bom (menor que 1% do ótimo) para uma grande classe de problemas simétricos. Johnson et al. (1996) comentam que o limite HK exato foi conseguido para instâncias de até 33.810 cidades, usando um programa cuja programação foi especialmente adaptada. Para instâncias ainda maiores, é aplicado um método de subgradientes, proposto originalmente nos trabalhos de Held e Karp, mas melhorados por inúmeros recursos (Veja em Volgenant (1982) e Valenzuela (1995)). Como para algumas instâncias grandes não conhecemos ainda as soluções ótimas, a comparação de resultados de heurísticas com o limite HK tornou-se prática comum.

A relaxação Lagrangeana/surrogate combina de forma eficiente as relaxações Lagrangeana e surrogate de um determinado problema. A relaxação Lagrangeana é usada após a aplicação de uma relaxação do tipo surrogate em um conjunto de restrições pré-escolhido, onde as restrições são substituídas por apenas uma, modificada por um multiplicador multidimensional. Em seguida toma-se a relaxação Lagrangeana da restrição surrogate obtida. Isto induz a um dual local Lagrangeano na variável unidimensional, e esta otimização local tende a corrigir a norma do vetor de gradientes, evitando fortes oscilações em métodos de otimização que usam subgradientes como direção de busca.

A relaxação Lagrangeana/surrogate foi aplicada com sucesso em diversos problemas de natureza combinatória. A otimização local (dual Lagrangeano local) não necessita ser exata e uma busca unidimensional do tipo dicotômica é empregada. Esta otimização também não se mostrou necessária em todos os passos do método de subgradientes, bastando encontrar o valor do multiplicador por algumas iterações iniciais do método subgradientes.

A relaxação Lagrangeana/surrogate tem se firmado como alternativa à relaxação Lagrangeana, onde os mesmos resultados podem ser obtidos em tempos computacionais muito menores. Em Lorena e Narciso (2001) foi examinado o uso da relaxação Lagrangeana/surrogate com o PCV. Para uma classe de problemas-teste foram obtidos resultados semelhantes em limites aos do uso da relaxação Lagrangeana, mas consumido apenas 2% do tempo computacional para problemas maiores.

Embora seja fácil de implementar e possua condições de convergência simples de controlar, o método usual de subgradientes não é muito eficiente do ponto de vista computacional, podendo apresentar um comportamento errático em muitas iterações e gastar muito tempo. Várias são as propostas de correção

deste comportamento que apareceram na literatura (Veja, por exemplo, em Larsson(1996)).

6.2.6.14 - Aplicações

(1) Uma aplicação industrial interessante encontra-se no problema da perfuração de placas de circuitos impressos (Reinelt-1989). Outras aplicações e variações assim como algoritmos exatos são descritos em Melamed, Sergeev e Sigal(1989).

Quando as diversas camadas que compõem uma placa de circuitos impressos estão assentadas, perfurações são feitas utilizando máquinas automáticas de controle numérico. As perfurações são necessárias para permitir a fixação de componentes na placa (transistores, resistores, circuitos integrados, etc) ou para possibilitar os contatos entre as diferentes camadas. A minimização do número de perfurações para o contacto configura um Problema de Otimização Combinatória importante, discutido em Grotschel, Junger e Reinelt (1989). O PCV aparecerá, neste caso, na execução eficiente das perfurações.

A máquina de perfuração laser parte de uma posição fixa onde as regulagens da ferramenta são selecionadas, permitindo que os diversos diâmetros possam ser produzidos. Os pontos na placa são alcançados pelo movimento simultâneo da placa na direção horizontal e da perfuradora na vertical. Considerando que o tempo necessário à regulagem da ferramenta é bastante longo, quando comparado ao tempo de movimentação da mesma, é razoável programar todas as perfurações de mesmo diâmetro em uma mesma seqüência. É imediato que a determinação da seqüência de perfurações de mesmo diâmetro corresponde à solução de um Problema Simétrico do PCV onde o ponto inicial e final correspondem à origem da ferramenta. Seja C_{ij} a distância entre o ponto i e o ponto j e $C = (c_{ij})$. Na verdade, cada seqüência representa uma instância do problema, sendo que o número de nós varia de algumas centenas a alguns milhares.

Neste e em outros casos práticos a utilização de algoritmos heurísticos eficientes torna-se uma imposição.

(2) Uma outra aplicação, que já mencionamos anteriormente é o de Sequenciamento de Tarefas onde $C = (c_{ij})$ tal que c_{ij} é o custo para completar a tarefa j imediatamente após a tarefa i, T_{ij} é o tempo (custo) de preparar a máquina para executar a tarefa j após a tarefa i e P_j é o custo para realizar a tarefa j. Logo, $c_{ij} = T_{ij} + P_{ij}$.

(3) Outra aplicação, também apresentada na motivação é o PRV (**Problemas de Roteamento de Veículos**) como generalização do PCV.

(3).1 - Motivação

Os problemas de roteamento constituem-se de um subconjunto de problemas de logística que são importantes para a solução de diferentes e importantes situações do dia-a-dia de muitas empresas, sejam elas microempresas ou de pequeno, médio ou grande porte. Esses problemas aparecem em todas as empresas que lidam diretamente com setores de transporte, mas, mesmo outras empresas, que não trabalham especificamente com logística e transporte, precisam minimizar seus custos de transportes de modo a escoar seus produtos e serviços e receber os insumos necessários à produção de bens e serviços de forma a melhorar seu saldo financeiro. Por exemplo, podemos citar: empresas varejistas que vendem produtos que precisam ser entregues com rapidez e pontualidade, e necessitam otimizar o rendimento de sua frota de veículos; fábricas que transportam insumos e produtos, mesmo que i nternamente entre diferentes setores de armazenagem, fabricação, estocagem e despacho; e grandes empresas que têm entregas de correspondência interna e externa, como é o caso dos Correios.

No mundo real, portanto, muitas empresas têm como desafios atacar problemas ligados ao transporte de pessoas, bens ou informações. Todas elas devem otimizar suas atividades de transporte vez que seus custos representam considerável parcela de recursos que certamente podem influenciar no preço final de seus produtos e/ou serviços. Com a crescente globalização da economia mundial; o uso intensivo da Internet como meio de comércio; os avanços na modernização das indústrias de manufatura e a necessidade de atendimento personalizado da clientela, entre outros fatores, a tendência é que o transporte se torne um item cada vez mais crítico (a cada dia cresce muito o número de veículos e pessoas nas grandes cidades), num futuro próximo.

Por todos esses motivos, formular, modelar e resolver diferentes tipos de problemas de transporte e, particularmente, de roteamento, tornou-se uma área de grande interesse da Pesquisa Operacional nos últimos anos.

(3).2- Diferentes Problemas

O estudo dos problemas de roteamento e dos demais problemas relacionados a transportes constitui uma área de trabalho bastante ativa na área de Pesquisa Operacional, sub-área: Problemas de Logística e Localização. Durante os últimos anos muitos trabalhos foram publicados, implementados e estudados, e, várias revistas científicas se especializaram apenas na publicação de artigos envolvendo problemas de transporte.

Os problemas de roteamento são aqueles que envolvem distribuir: bens, pessoas ou informação em rotas, ou seja, determinar qual o caminho de menor custo que cada item deverá seguir para que seja entregue ou recebido um determinado serviço. De um modo geral os problemas estudados em pesquisa científica envolvem modelos gerais de atendimento a diferentes situações (modelo genérico) dos quais os problemas reais podem vir a ser incorporados com (na maioria dos casos) pequenas modificações de modo a atender especificidades inerentes a determinada particular situação. Assim, nas pesquisas científicas modelam-se propriedades básicas de problemas reais e, desse modo, obtém-se resultados importantes que são utilizados como base para análises e implementações de sistemas para resolver diferentes problemas reais. Por esse motivo, a literatura científica criou uma variedade de modelos básicos cuja investigação é considerada importante. Vimos isso, claramente, no particular estudo de procurar a solução ótima do Problema do Caixeiro Viajante (PCV) ou, então: "The Traveling Salesman Problem(TSP)´´.

Em um problema de roteamento puro, existe apenas um componente a considerar: o das distâncias geográficas, enquanto que em problemas de roteamentos mais reais, devem também ser incluídas (caso a situação em estudo assim o exija) componentes de tempos, o que dá origem aos problemas de scheduling, isto é, o de determinar uma seqüência de tarefas levando em consideração o tempo de duração de cada uma delas, o tempo entre as diferentes tarefas a serem executadas e o tempo de espera entre o término de uma tarefa e o início da tarefa seguinte.

Suponha agora que tenhamos k vendedores disponíveis, e que cada cidade deva ser visitada por um único vendedor (escolhido arbitrariamente), uma única vez. Todos os vendedores estão lotados na mesma cidade (fábrica, depósito ou filial) e devem retornar a ela assim que a rota for completada. O objetivo, novamente, é o de minimizar a soma dos custos das rotas de cada um dos vendedores. Este problema é conhecido como o $k-$TSP, que não é mais ou menos difícil que o problema anterior, uma vez que ele pode ser reformulado como um TSP (para

este resultado veja em [Bodin et al., 1983]. No sentido da definição que demos acima, ambos os problemas TSP e $k-$TSP são problemas de roteamento puros.

Se ao problema anterior ($k-$TSP) associarmos uma demanda a cada cidade, e cada vendedor só pode servir àquelas cidades cuja demanda total (somatório das demandas das cidades) não exceda a sua capacidade (que pode ser diferente para cada vendedor), então estamos diante de um problema conhecido na literaura como o Problema Clássico de Roteamento de Veículos (PRV). Neste caso, as cidades são chamadas de clientes, e os vendedores de veículos. A soma das demandas de uma rota, neste caso, não pode exceder a capacidade do veículo alocado para esta rota. Como no $k-$TSP, queremos minimizar o somatório dos custos das rotas. O PRV não é, no entanto, um problema puramente de ordem geográfica, uma vez que a restrição associada à demanda pode ser uma restrição bastante significativa.

O PRV clássico é o modelo básico para grande número de problemas de roteamento de veículos. Para uma análise mais detalhada sobre o PRV, veja em LAPORTE (1997), [Golden, 1984] e Fisher 1995. O PRV também é conhecido como Problema Capacitado de Roteamento de Veículos (PCRV).

Se, além da restrição da capacidade dos veículos adicionarmos ao PRV a restrição de que o veículo deva visitar cada cliente obedecendo a um determinado horário de entrega, isto é, a uma janela de tempo, temos o conhecido Problema de Roteamento de Veículos com Janelas de Tempo (PRVJT). Se retirarmos as restrições de capacidade dos veículos, recaímos no problema do caixeiro viajante com janelas de tempo (PCVJT). Se em cada uma das rotas do PRV uma determinada distância não puder ser excedida, estamos diante de um **Problema de Roteamento de Veículos com Restrições de Comprimento** (PRVRC).

Outras variantes podem ser, por exemplo, o acréscimo de mais de um depósito ou de mais de um tipo de item a ser entregue. No modelo de entrega distribuída (Split Delivery) a demanda de um determinado cliente não é, necessariamente, atendida por um único veículo, podendo ser distribuída entre dois ou mais veículos. As soluções obtidas, para este modelo, terão sempre, pelo menos a mesma qualidade daquelas que teriam sido obtidas resolvendo o modelo tradicional do PRV. Entretanto, como sempre é possível utilizar-se da melhor forma a capacidade de carga dos veículos (para esse tipo de problema temos o conhecido "Problema da Mochila $0-1$´´. Maiores detalhes veja em Kolesar(1967)

e Greenberg e Hegerich (1970), é também possível diminuir a quantidade de veículos a serem utilizados.

No caso do veículo que não só faz entregas mais também faz coletas enquanto percorre as rotas, temos o conhecido PRV com Coleta e Entrega (Pickup and Delivery). Este problema pode, ainda, ter outras variantes, no caso em que seja estabelecida uma ordem em que as entregas e as coletas sejam feitas.

Os custos envolvidos nos problemas de roteamento e scheduling consistem de custos fixos da utilização de veículos e custos variáveis de scheduling e roteamento. Os custos de scheduling incluem custos relacionados ao tempo de viagem e à distância percorrida, custos associados ao tempo de espera na entrega/coleta e custos de tempo de carga/descarga. Além disso, ainda podem ser incorporados custos relacionados ao transporte de indivíduos.

Os problemas de roteamento de veículos podem ser examinados sob um ponto de vista mais geral quando o termo veículo é considerado de forma mais abrangente. Neste caso, vários problemas de scheduling são estudados como problemas do tipo PRVJT. Um exemplo é o caso de uma única máquina para a qual queremos seqüenciar um número de tarefas, das quais conhecemos o tempo de duração e o tempo de transição entre as tarefas. Esse problema de scheduling pode ser analisado como um PRVJT com um único depósito, um único veículo e onde cada cliente representa a tarefa a ser executada. O custo de transição de uma tarefa para a seguinte é igual ao custo associado à viagem de um cliente para outro. E o tempo de serviço é o tempo para realizar a tarefa. Para uma visão mais profunda sobre os problemas de roteamento e scheduling veja em DESROSIERS et al. (1995), BREEDAM (1995) e CRAINIC e LAPORTE (1998).

(4)Outro exemplo ocorre no projeto de sistemas de hardware. Tais sistemas possuem muitos módulos, cada um contendo sua própria pinagem, isto é, uma certa quantidade de pinos. A posição física de cada módulo é pré-determinada. Desejamos interconectar um dado subconjunto de pinos por fios, sujeito às seguintes condições:

(1) no máximo dois fios podem ligar dois pinos (devido ao seu tamanho e possíveis mudanças futuras no layout dos fios);
(2) o comprimento total dos fios deve ser minimizado (para facilidade e organização da fiação).

Esta situação pode ser modelada por um grafo completo num grafo valorado G. Seus vértices correspondem a um dado conjunto de pinos rorulados com

$1, 2, \ldots, n$. O peso $p_{ij}(i < j)$ de uma aresta que conecta o pino i ao pino j é a distância entre o pino i e o pino j. A condição (1) determina que devemos encontrar um circuito hamiltoniano em G e a condição (2) especifica que tal circuito deve ter comprimento mínimo. Se introduzimos um pino auxiliar rotulado com 0, e fazemos $p_{0i} = 0, 1 \leq i \leq n$, então o problema da fiação torna-se um PCV com $n + 1$ cidades.

7. **Conclusões**

Este trabalho foi concebido a partir de uma afirmação, repetida inúmeras vezes, quando o contexto assim o permitiu, seja em sala de aula num curso regular do Programa de Engenharia de Produção da COPPE/UFRJ, seja em conferências ou minicursos proferidos pelo Professor Samuel Jurkiewicz: "Os Problemas têm solução; Nós é que não temos tempo´´. O Professor Samuel enumere diferentes problemas em Teoria de Grafos cuja relevância na obtenção da solução ótima nos remete a questões de não possuir algoritmos em tempo polinomial para apresentar as soluções ótimas desses problemas em tempo razoável, e as possibilidades que dispomos para apresentar alternativas de torná-las exeqüíveis, quando possível.

Neste trabalho nos deparamos com a análise e alternativas de soluções de apenas dois desses problemas sérios: O Problema do Carteiro Chinês e o Problema do Caixeiro Viajante. Esperamos que o leitor tenha tido compreensão razoável e suficiente com esse pequeno tempo de maturação, de se permitir concluir o quão interessantes e atraentes são as pesquisas que estão em andamento para a obtenção de algoritmos capazes de permitir soluções ótimas para esses tipos de problemas, em diferentes aplicações presentes em nosso dia-a-dia.

Surge uma inevitável indagação do Professor Paulo Cezar Pinto de Carvalho, do IMPA-Instituto de Matemática Pura e Aplicada quanto ao que o leitor possa estar pensando, em um artigo escrito para a Revista Eureka:

"*mas será que esta história de algoritmos eficientes tem relevância, numa era de computadores cada vez mais velozes? Afinal de contas, existe um algoritmo extremamente simples para verificar se um grafo possui um circuito hamiltoniano. Se existir um tal circuito, ele corresponderá a uma*

permutação (circular) dos vértices com a propriedade de que vértices consecutivos sejam ligados por um arco do grafo. Ora, para verificar a existência de circuito hamiltoniano basta gerar todas as permutações circulares dos vértices e testar se uma delas corresponde a um percurso no grafo.´´

É claro que este algoritmo funciona para grafos de tamanho moderado (ele poderia ser o recurso usado pelo caixeiro viajante: com apenas 9 cidades, ele teria que testar "apenas´´ $8! = 40.320$ caminhos, o que seria feito com rapidez em um computador). Mas o que ocorre com grafos maiores? Vejamos, por exemplo, uma situação em que o número de cidades cresce para 50 (o que representaria um tamanho ainda bastante razoável para uma situação real). Neste caso, o computador deveria examinar 49! circuitos potenciais. Tentemos estimar a magnitude deste número. A forma mais simples é usar a fórmula de Stirling, que fornece a estimativa $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$. Mas, neste caso, podemos usar estimativas mais elementares. Por exemplo, podemos usar apenas potências de 2. Temos:

$49! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times ... \times 15 \times 16 \times ... \times 31 \times 32 \times \times 49 > 1 \times 2 \times 2 \times 4 \times 4 \times 4 \times 4 \times 8 \times \ldots \times 8 \times 16 \times \ldots \times 16 \times 32 \times \ldots \times 32 = 2^2 \times 4^4 \times 8^8 \times 16^{16} \times 32^{18} = 2^{2+8+24+64+90} = 2^{188}$.

Mas $2^10 = 1024 > 10^3$. Logo $49! > \frac{1}{4} \times 10^{57}$.

Ora, um computador moderno pode realizar cerca de 2 milhões de operações por segundo. Se em cada operação ele conseguir testar um circuito, ele ainda assim precisará de mais de $\frac{1}{4} \times 10^{57}/2 \times 10^6 = \frac{1}{8} \times 10^{51}$ segundos, o que corresponde a aproximadamente a $\frac{1}{32} \times 10^{44}$ anos. Assim, trata-se claramente de uma missão impossível para o algoritmo de força bruta, baseado na análise de cada permutação de vértices.

O resultado da discussão acima pode parecer bastante desanimador: não parece haver bons métodos para verificar a existência de um circuito hamiltoniano e algoritmos de força bruta só funcionam para problemas com pequeno número de vértices (é bom que se diga que existe um meio termo: há estratégias que permitem resolver o problema acima para valores razoáveis de n, reduzindo substancialmente o número de possibilidades a serem examinadas; mesmo estes algoritmos, no entanto, tornam-se impráticos a partir de um certo ponto). O mesmo ocorre com todos os chamados problemas NP-completos´´.

Portanto, neste trabalho, procuramos levar o leitor a perceber que "Probleminhas´´ podem e devem ser apresentados a jovens e crianças no sentido de

levantar simples questões que a Matemática Discreta, em particular a Teoria dos Grafos, oferece. Esses "probleminhas´´ poderiam ser apresentados através de propostas para discussão e solução com um pequeno número de vértices. De salutar importância, os "problemas´´ poderiam, se assim for conveniente, ser apresentados e as possibilidades de solução questionadas, encaminhando para a criação, discussão, obtenção e validação de algoritmos simples, não necessariamente formais, mas que levem os leitores a descobrir o fascínio que é a Teoria dos Grafos e seus diferentes e desafiadores problemas.

Esperamos ter conseguido motivar o leitor da importância desses "**probleminhas´´** e "**problemas´´** no nosso dia-a-dia, e, como é possível levá-los para a discussão em sala de aula e outros ambientes que necessitem de sua atuação.

Referências

[1] Bellmore, M., e Malone, J. C., Pathology of traveling salesman subtour elimination algorithms. Oper. Res., v.19, p.278 - 307,1971.

[2] Biggs, N.L., Lloyd, E.K. and Wilson, R.J., Graph Theory 1736-1936, Claredon Press, Oxford, 1976.

[3] Bock, F., "An algorithm for solving "travelling-salesman´´ and related network optimization problems´´, 14th ORSA National Meeting, 1958.

[4] Bondy, J.A. and Murty,U.S.R., Graph Theory with Applications, North-Holland, 1976.

[5] Bondy, J.A. and Chvátal,V., A Method in graph theory, Discr.Math.15, 111-136 (1976).

[6] Carvalho, P.C.P., "Dois problemas sobre grafos´´, Revista Eureka, a aparecer, 2006.

[7] Chandra, B., Karloff, H., Tovey, C., "New results on the old k-opt algorithm for the PCV,´´ in Proceedings 5th ACM-SIAM Symp on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[8] Chartrand, G. and Oellermann, O.R., Applied and Algorithmic Graph Theory, McGraw-Hill, Inc., 1993.

[9] Christofides, N., The Travelling Salesman Problem. In: Combinatorial Optimization, Edited by Christofides, N., Mingozzi, A, Toth, P., Sandi, C., John Wiley & Sons Inc., New York, p. 131 - 149, 1979.

[10] Christofides, N., Worst case analysis of a new heuristic for the traveling salesman problem, Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976. John Wiley & Sons Inc., New York, p. 131 - 149, 1979.

[11] Chvátal, V. and P.Erdös, A Note on Hamiltonian circuits, Discr.Maths.2, 111-113 (1972).

[12] Clarke, G. e Wricht, J. W., Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. Operations Research, v. 12, p. 568 - 581, 1963.

[13] Dantzig. G. B., Fulkerson, D. R., e Johnson, S. M., Solution of a large scale travelling salesman problem. Oper. Res. 2, p. 393 - 410,1954.

[14] Edmonds, J., Maximum matching and a polyhedron with 0,1 vertices, J. Res. Nat. Bur. Standards, 69 B, pp. 125-130, 1965.

[15] Eilon, S., Watson-Gandy, C. D. T. and Christofides, N., Distribution Management, Mathematical Modelling and Practical Analysis. London: Charles Griffin & Company Ltd., 1971.

[16] Fleischner, H., Eulerian Graphs and Related Topics (Part 1, Volume 1), Nort-Holland, 1990.

[17] Garey, M. R., Graham, R. L. and Johnson, D.S., "Some NP - Complete Problems, Eigth Annual Symp. Ou Theory of Comput., p.10-22,1976.

[18] Golden, B., Bodin, L., Doyle, T. and Stewart Jr., W., Approximate Travelling Salesman Algorithms. Oper. Res., v.2, p .694 - 711, 1980.

[19] Glover, F., Stronger cuts in integer programming. Operations Research, 15, p.1174 - 1176, 1967.

[20] Guan, M., Graphic programming using odd or even points, Acta Math. Sinica, 10, pp. 263-266, 1960; Chinese Math., 1, pp. 273-277, 1962.

[21] Held, M., Karp, R.M., "The Traveling salesman problem and minimum spaning trees´´.Operations Research, 18, p.1138-1162, 1970.

[22] Held, M., Karp, R.M., "The Traveling salesman problem and minimum spanning trees: Part II´´. Mathematical Programming 1, 6-25, 1971.

[23] Jonhson, D.S., McGeoch, L.A., "The traveling salesman problem: a case study in local optimization´´.In Local search in Combinatorial optimization, E. H. L. Aarts e J. K. Lenstra (eds.), John Wiley & Sons, New York, 1997.

[24] Jonhson, D.S., McGeoch, L.A., Rothberg, E.E., "Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound,´´ Proceedings od the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, p. 341 - 350, 1996.

[25] Johnson, D.S., Papadimitriou, C.H., Yannakakis, M., "How easy is local scarch?´´, J.Comput.System Sci, 37, 1988.

[26] Karg, R. L. and Thompson, G. L., A heuristic approach to solving traveling salesman problem. Management Sci. v.10, p. 225 - 248, 1964.

[27] Krolak, P., Felts, W., e Marble, G., A man-machine approach toward solving the traveling salesman problem. Commun. Assoc. Comput. Mach. 14, p.327 - 334. 1971.

[28] Euler, L., solution problematic ad geometriam situs pertinentis, Comment. Acad. Sci. Petropolitanae, 8, pp. 128-140, 1736.

[29] Jurkiewicz, S., Grafos planares hamiltonianos, Tese M.Sc., COPPE/UFRJ, 1990.

[30] Larsson, T., Patriksson, M. and Stromberg, A.B., "Conditional subgradient optimization - theory and applications´´, European Journal of Operational Research 88, p. 382 - 403, 1996.

[31] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B., eds., The Traveling Salesman Problem, John Wiley, and Sons, Chichester, 1985.

[32] Lin, S., Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. 44, p.2245 - 2269,1965.

[33] Little, J.D.C., Murty, K.G., Sweeney, D.W. and Karel, C., An algorithm for the traveling salesman problem. Operational Research 11, p.979 - 989,1963.
[34] Lorena, L.A.N., Narciso, M.G., "Using logical surrogate information in Lagrangean relaxation: na application to symmetric traveling salesman problems´´ European Journal of Operational Research, 2001 - to apear. Disponível em http://www.lac.inpe.br/ lorena/ejor/98296.pdf.
[35] Lucas, E., Récréations Mathématiques IV, Paris, 1921.
[36] Lueker, G., manuscript, Princeton University, 1976.
[37] Martin, G.T., Solving the traveling salesman problem by integer linear programming. CEIR, New York,1966.
[38] Miller, C.E., Tucker, A.W. and Zemlin, R.A Integer programming formulation of travelling salesman problems. J. Assoc. Comput. Mach. 7, p.326 - 329,1960.
[39] Ong, H.D. and Huang, H.C., Asymptotic Expected Performance of Some PCV Heuristics: An Empirical Evaluation. European Journal of Operational Research, v. 43, nº.2, p.231 - 238, 1989.
[40] Ore, O., Graphs and Their Uses, The Mathematical Association of America.
[41] Papadimitriou, C. H. e Steiglitz, K. Combinatorial Optimization: Algorithms and Complexity, New York: Prentice-Hall Inc., p.406 - 483, 1982.
[42] Papadimitriou, C.H. and Steiglitz, K., "Some examples of difficult travelling salesman problem.´´ Operations Res., 26, 1978.
[43] Parker, R.G.,Rardin, L.R., "Discrete Optimization´´. Academic Press, INC, London, 1988.
[44] Sahni, S. and Gonzalez , T., P-Complete Approximation Problems.J.ACM, v.23, p.555 - 565,1976.
[45] Shapiro, D., Algorithms for the solution of the optimal cost traveling salesman problem. Sc.D. Dissertation, Washington Univ., St. Louis, Missouri,1966.
[46] Shapiro, J. F., "Generalized lagrange multipliers on integer programming´´. Operations Research, 19, p.68 - 76, 1971.
[47] Szwarcfiter, J.L., Grafos e Algoritmos Computacionais, Editora Campus.
[48] Valenzuela, C.L., Jones, A.J., "Estimating Held-Karp lower bond for the geometric PCV´´, 1995.
[49] Volgenant, T., Jonker, R., "'A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation´´. European Journal of Operational Research, 9, p. 83 - 89, 1982.

Instituto Alberto Luiz Coimbra de Pós Graduação e Pesquisa de Engenharia - Programa de Engenharia de Produção - COPPE/UFRJ

E-mail address: pjorge@pep.ufrj.br / jurki@pep.ufrj.br