ALGORITMOS APROXIMATIVOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE

Andrea Rosario Pari Soto

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Sulamita Klein, D.Sc.

Prof. Nelson Maculan Filho, D.Habil

Prof. Luerbio Faria, D.Sc.

Prof. Luiz Satoru Ochi, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 1999

SOTO, ANDREA ROSARIO PARI

Algoritmos Aproximativos para o Problema do Caixeiro Viajante [Rio de Janeiro] 1999

XII, 87 pp., 29.7 cm, (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1999)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 – Teoria de Grafos

2 – Algoritmos Aproximativos

3 – Esquema Aproximativo

I. COPPE/UFRJ II. Título (série)

Aos meus pais,
Juana e Justino.

Agradecimentos

- Primeiramente, gostaria de agradecer aos meus orientadores Sulamita Klein e Nelson Maculan que estiveram ao longo das dificuldades no desenvolvimento deste trabalho: A Sulamita Klein pela paciência, pelo constante estímulo, pela confiança em meu trabalho e orientação durante o curso de mestrado. Ao Prof. Nelson Maculan Filho, pelo apoio, compreensão e pela liberação concedida para que eu pudesse continuar com tranquilidade o meu trabalho de mestrado.

- Ao Luerbio Faria a quem devo muito pela ajuda e atenção a mim dedicada e por ter tornado este trabalho possível.

- Aos meus irmãos que me incentivaram nas horas difíceis. A vocês, muito obrigado pelo carinho, força e atenção a mim despensados.

- Aos professores, colegas estudantes e amigos brasileiros e peruanos pela amizade e compartilhamento de conhecimentos, experiências e emoções na UFRJ e na cidade do Rio de Janeiro.

- Á CAPES (Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo apoio financeiro.

- À Universidade Federal do Rio de Janeiro.

- Principalmente a Deus.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ALGORITMOS APROXIMATIVOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE

Andrea Rosario Pari Soto

Outubro/1999

Orientadores: Sulamita Klein

Nelson Maculan Filho

Programa: Engenharia de Sistemas e Computação

Este trabalho consiste no estudo de algoritmos aproximativos para o problema do caixeiro viajante euclideano para a obtenção de um tour ótimo para grafos não direcionados. Apresenta-se também um esquema aproximativo desenvolvido recentemente por Sanjeev Arora para o problema do caixeiro viajante euclideano no plano. A idéia principal desse esquema é realizar um particionamento geométrico recursivo de uma instância em instâncias menores e posteriormente aplicar a técnica de programação dinâmica.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Masters of Science (M.Sc.)

APPROXIMATIVE ALGORITHMS FOR THE TRAVELING SALESMAN PROBLEM

Andrea Rosario Pari Soto

October/1999

Advisors: Sulamita Klein

Nelson Maculan Filho

Department: Computing and Systems Engineering

In this work we study approximation algorithms for the euclidean Traveling Salesman Problem. We also present an aproximation scheme for the Traveling Salesman Problem for the plane, that was recently developed by Sanjeev Arora. The key idea of this scheme is to realize a recursive geometric partition of an instance into smaller intances and then to apply Dinamic Programing.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste primeiro capítulo é fornecida uma visão geral do trabalho.

Dado um grafo não direcionado completo $G = (V, E)$, com custos nas arestas, o problema do caixeiro viajante tem como objetivo, encontrar um ciclo de custo mínimo passando exatamente uma vez por cada vértice em V. O problema do Problema do Caixeiro Viajante denotado por PCV é um dos problemas de otimização combinatória mais conhecidos, não só por ter sido inicialmente formulado por Menger em 1930 [27] e surgir em diversas formulações, bem como por pertencer a classe de problemas $NP - completos$ [30], i.e., não pode ser resolvido em tempo polinomial a menos que $P = NP$.

Este problema clássico motivou pesquisas em muitas áreas, tal como a programação linear, análise probabilística de algoritmos, esquemas aproximativos, e *MAX SNP*-completude [25]. Esse problema tem sido até agora exaustivamente estudado e podemos citar que até existe um livro inteiramente dedicado a ele [25] com muitos resultados interessantes sobre variantes e casos especiais. A sua resolução do PCV não é tão simples, uma vez que os métodos exatos demandam muito tempo para problemas de tamanho razoavelmente grandes e os métodos aproximados, em geral bons, porém nem sempre garamtem o ótimo, sendo estes ultimos de nosso interesse. O sucesso de um algoritmo aproximativo depende se sua capacidade de: adaptação para instâncias especiais, escapando do ótimo local, fazendo uso da estrutura do

problema, e além disso, sob o uso de: estrutura de dados eficientes, boas técnicas para a construção de soluções iniciais, reinicialização de procedimentos, solução melhorada por busca local, diversificação de busca quando não há possíveis melhoramentos, e intensificação de busca em regiões viáveis. Estas observações e idéias levam ao desenvolvimento de novas classes de algoritmos aproximativos, o qual consideravelmente melhora a sua eficiencia e o tamanho de instâncias a resolver para problemas dicíceis encontrados em prática, muitos freqüentemente basados sobre modelos de natureza (física, biológica, evolução). Pdemos mencionar varios técnicas de algoritmos aproximativos que podem ser métodos construtivos, de busca local, procedimentos de busca Greedy Randomized Adaptive, vizinhança mais próxima, algoritmos genéticos, algoritmos meméticos, otimizaão de Ant Colony, redes neuronais, veja [34].

Christofides [11] projetou um algoritmo aproximativo construtivo que calcula um tour de custo de no máximo 1.5 vezes o do ótimo. Arora, Mitchell mostraram que o PCV euclideano admite um Esquema de Aproximação em Tempo Polinomial ([2] e [29] respectivamente) denotado por $EATP$. O algoritmo de Arora [2] também generaliza para pontos em $\Re^d$. O algoritmo de Mitchell [29] trabalha somente no plano. Trevisan [37] apresentou evidências de que cada esquema aproximativo para $\Re^d$ deve ter um tempo de execução com uma dependência duplamente exponencial sobre d.

Recentemente, Grigni [16] estendeu as idéias de [2] e de [23] para projetar um $EATP$ para qualquer métrica que é o caminho mínimo métrico de um grafo planar com pesos (o trabalho de [23] projeta tal $EATP$ para grafos planares sem custos nas arestas).

O análise do pior caso procura estabelecer o desvio máximo que podera ocorrer quando um algoritmo aproximativo é aplicada a determinado problema. Neste trabalho os algoritmos apresentados, estudam este caso. Por outro lado a análise do comportamento médio consiste em programar o algoritmo aproximativo e aplica-la a um conjunto representativo da instância do problema que se quer resolver. Então a análise do comportamento médio dos algoritmos estudados forneceria evidência

estatística de como será o desempenho do algoritmo aproximado, que pode não ser muito preciso, dependendo da qualidade do código gerado, compilador e hardware utilizados, etc.

Várias e outras referências encontram-se no library TSPLIB [34] para muitos trabalhos experimentais sobre o *PCV*

1.1 Objetivos da Tese

Sumariamente, os principais objetivos dessa tese são:

- Estudo dos algoritmos aproximativos: Algoritmo da Árvore e Algoritmo de Christofides [25] e [31], para encontrar um tour ótimo de custo mínimo para o problema.
- Estudo de um esquema aproximativo em tempo polinomial para o *PCV* euclideano [2] que trabalha por um processo recursivo aplicando a têcnica da programação dinâmica.

1.2 Organização da Tese

O conteúdo restante da tese está organizado da seguinte forma:

No Capítulo 2 apresentamos os conceitos básicos na área de teoria dos grafos necessários para a compreensão do restante da tese. As principais referências para este capítulo citamos [8] e [36]; e para uma referência básica sobre Complexidade Computacional cita-se [15].

No Capítulo 3 contamos um pouco da história [19] do PCV e mostramos que tal problema pertence a classe de complexidade NP.

No Capítulo 4 centramos nossa atenção nos algoritmos aproximativos para achar um tour mínimo, da árvore mínima e de Christofides, como referências para os dois capítulos cita-se [9], [25] e [31].

No Capítulo 5 apresenta-se um esquema de aproximação em tempo polinomial para o PCV euclideano no plano, e descreve-se como o tour pode ser encontrado usando a técnica de programação dinâmica [7].

Finalmente, no Capítulo 6 apresentamos as conclusões.

Capítulo 2

Definições e Conceitos básicos

Neste capítulo apresentamos alguns conceitos e notações básicas que serão utilizados ao longo desta tese.

2.1 Grafos

Um *grafo simples* ou *grafo* G é um par ordenado $G = (V, E)$ onde V é um conjunto finito não vazio de elementos distintos denominados *vértices* e E é um conjunto de pares não ordenados de vértices distintos pertencentes a V denominados *arestas.* Vamos assumir que $|V| = n$ onde $|V|$ denota o número de vértices de V e $|E| = m$ onde $|E|$ denota o número de arestas de E. Cada aresta $e \in E$ será denotada pelo par de vértices $e = (v, w)$ que a forma. Nesse caso, os vértices v e w são os extremos (ou extremidades) da aresta e, sendo denominados *adjacentes.* A aresta e é dita *incidente* a ambos v, w. Duas arestas que possuem um extremo comum são chamadas de *adjacentes.*

Podemos estender a definição de grafos, de modo a permitir a existência de um par de vértices idênticos, i.e., do tipo $e = (v, v)$. Uma tal aresta é chamada de *laço.* Uma outra extensão possível consiste en sustituir, na definição de grafo, o conjunto de arestas E por um multiconjunto (conjunto o qual pode conter elementos repetidos). O efeito desta alteração, permite a existência de mais de uma aresta

associada a um mesmo par de vértices, que são denominadas *arestas paralelas.* A estrutura (V, E) assim definida é chamada de *multigrafo.* (Veja figura 2.1).

Figura 2.1: (a)Grafo, (b)Multigrafo

Um grafo $H = (V(H), E(H))$ é um *subgrafo* de um grafo $G = (V(G), E(G))$ se $E(H) \subseteq E(G)$ (denotamos $H \subseteq G$). Se além disso se toda aresta de $E(G)$ com extremidades em $V(H)$ pertence a $E(H)$, então H é um *subgrafo induzido* de G. Se $V(G) = V(H)$ denomina-se H *subgrafo gerador* de G.

Em um grafo $G = (V, E)$, define-se o *grau de um vértice* $v \in V$, denotado por *grau(v)*, como sendo o número de vértices adjacentes a v. Um grafo é *regular de grau* r, quando todos os seus vértices possuirem o mesmo grau r. Observe que cada vértice v é incidente a $grau(v)$ arestas e cada aresta é incidente a dois vértices. Conseqüêntemente temos o seguinte teorema.

Teorema 2.1. *Em qualquer grafo G temos:*

$$\sum_{v \in V} grau(v) = 2|E|$$

Observe que esse resultado vale também para grafos que têm laços, por que cada laço contribui exatamente com 2 unidades na soma do grau do vértice correspondente.

Como consequência do Teorema 2.1 temos ainda: (veja [39])

1. Em qualquer grafo, a soma de todos os graus dos vértices é um número par.
2. Em qualquer grafo, o número de vértices de grau ímpar é par.
3. Se G é um grafo o qual tem n vértices e é regular de grau r, então G tem exatamente $\frac{1}{2}nr$ arestas.

Um *passeio* em G é uma sequência finita não-vazia $P = [v_1, v_2, \ldots, v_k]$ de vértices de V tal que $(v_i, v_{i+1}) \in E$, para todo $1 \leq i \leq k-1$. P é dito um passeio de v_1 a v_k. O comprimento do passeio é dado pelo número de arestas percorridas. Se todos os vértices do passeio forem distintos ele se denomina *caminho simples* ou (simplesmente) *caminho*. Um *passeio fechado* é aquele em que $v_1 = v_k$.

Um *ciclo* é uma sequência $C = [v_1, v_2, \ldots, v_k, v_{k+1}]$ tal que $[v_1, v_2, \ldots, v_k]$ é um caminho, $v_1 = v_{k+1}$ e $k \geq 3$. Um grafo é *acíclico* se ele não possui ciclos.

Um *passeio euleriano* de G é um passeio fechado que contém cada aresta de G exatamente uma vez. Um grafo G é euleriano se ele possui um passeio euleriano.

Um *ciclo hamiltoniano* em um grafo G é um ciclo que passa por cada vértice de G exatamente uma vez. Tal ciclo chamaremos também de *tour*. Um grafo é *hamiltoniano* se ele possui ciclo hamiltoniano.

Um caminho que contém cada vértice do grafo exatamente uma vez é chamado *caminho hamiltoniano*. Por outro lado, algum caminho ou ciclo que contém cada aresta do grafo, também exatamente uma vez cada, é denominado *euleriano* (veja figura 2.2).

Observe que qualquer grafo hamiltoniano contém um caminho hamiltoniano, mas um grafo que contém um caminho hamiltoniano nem sempre implica que este grafo seja hamiltoniano (veja figura 2.2: O grafo I contém o caminho $[v_1, v_5, v_2, v_3, v_4]$).

O teorema a seguir caracteriza os multigrafos (em particular os grafos) que são eulerianos.

Teorema 2.2. *[31] Um multigrafo $G = (V, E)$ é euleriano se e somente se*

(a) G é conexo, e

(b) todos os vértices de V têm grau par

Prova: Primeiro provaremos as duas condições de necessidade. Como G é um multigrafo euleriano, seja $C = [v_1 v_2 \ldots v_i \ldots v_k]$ o ciclo que contém cada aresta de G exatamente uma vez, i.e., percorre todo o multigrafo G. Logo G é conexo. Observamos que cada vez que um vértice v_i ocorre como um vértice interno de C,

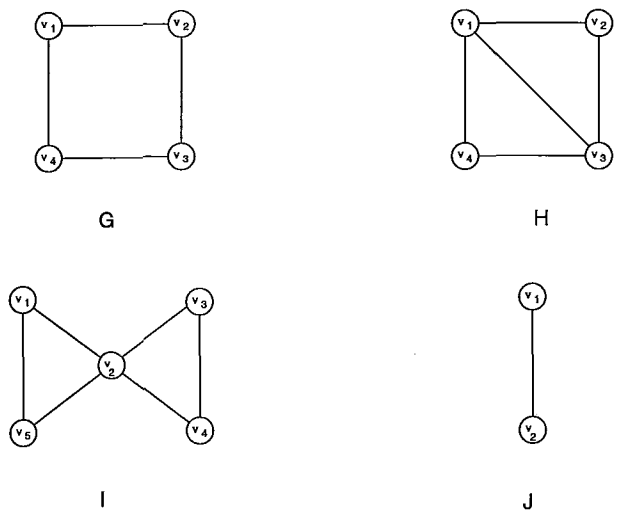


Figura 2.2: G é euleriano e hamiltoniano. H é hamiltoniano mas não é euleriano. I é euleriano mas não hamiltoniano. J não é euleriano nem hamiltoniano.

duas arestas incidentes de C são afetadas. Como um multigrafo euleriano contêm todas as arestas de G (uma vez) teremos $grau(v_i)$ par para $v_i \neq v_1$ e como C começa e termina em v_1 então $grau(v_1)$ é par, logo todos os vértices de V tem grau par.

Agora vamos mostrar que as condições (a) e (b) são suficientes, usando indução no número de arestas de G.

Tomemos como base da indução um multigrafo trivial (um único vértice e nenhuma aresta). Ele satisfaz trivialmente.

Seja $G = (V, E)$ um multigrafo satisfazendo condições (a) e (b).

Suponhamos agora, por hipótese de indução que todos os multigrafos com menos arestas que G e que satisfazem (a) e (b) sejam eulerianos. Escolhemos um vértice v de G e a partir dele começamos um passeio das arestas de G, nunca passando pelas arestas duas vezes, até v ser encontrado outra vez. Pela condição (b) isto é possível, pois todos os vértices de V tem grau par. Agora removendo as arestas deste passeio G, temos um número de componentes conexas. Porém cada componente satisfaz (a) e (b) e pela hipótese de indução é euleriano. E vemos que podemos criar um passeio euleriano de G "juntando" os passeios eulerianos das componentes do passeio

original. ∎

Observação: Este teorema sugere um algoritmo eficiente simples para determinar quando um multigrafo é euleriano e também fornece uma maneira recursiva para construir tal passeio se ele existir. Daremos uma descrição detalhada desse algoritmo no capítulo 4.

Um grafo $G = (V, E)$ é *conexo* se existe um caminho entre cada par de vértices de G. Caso contrário, G é desconexo.

Precisamos também da noção de árvore que damos a seguir.

Uma *árvore* é um grafo conexo e acíclico. Se um vértice v de uma árvore possuir $grau \leq 1$ então v é uma folha. Caso contrário, i.e., $grau(v) > 1$, então v é um vértice interior. Uma *árvore geradora* T de um grafo $G = (V, E)$ é um subgrafo gerador de G que é em particular uma árvore ou em outras palavras: dado um grafo $G = (V(G), E(G))$, T é árvore geradora de G se $V(T) = V(G)$, $E(T) \subseteq E(G)$ e T é conexo e acíclico. Uma *árvore binária* é uma árvore enraizada ordenada em que cada vértice não folha possui exatamente 2 filhos.

Um resultado importante que usaremos mais adiante é o seguinte.

Teorema 2.3. *[15] Todo grafo conexo G possui uma árvore geradora.*

2.2 As classes P, NP e NP-completo

Nesta seção vamos definir as classes P, NP e $NP - completo$ de problemas computacionais. Para isso precisamos de certas noções de problemas algorítmicos (veja [12] e [20]) que introduziremos agora, e sempre que possível, ou estiver ao nosso alcance, traduziremos os conceitos em termos de problemas de otimização combinatória, tratados em [31].

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 algorítmico consiste

em desenvolver um algoritmo, cuja entrada seja uma instância do problema e cuja saída responda ao objetivo do problema. Uma *instância* é um conjunto de dados de entrada do problema.

Para abordar a questão da complexidade, é conveniente classificar os problemas algorítmicos nas seguintes categorias:

- Problemas de Decisão
- Problema de Localização
- Problemas de Otimização

Num *Problema de Decisão*, o objetivo consiste em decidir a resposta *SIM* ou *NÃO* a uma questão. Usaremos a expressão *"instância SIM"* para nos referirmos a uma instância de um problema de resposta *"SIM"* que será utilizado para designar uma solução, ou uma sugestão de solução (seja viável, seja ótima, dependendo daquilo que se peça no problema). Além do mais, os problemas combinatórios, quando considerados na sua versão de decisão, serão chamados de "problemas de decisão". Num *Problema de Localização*, o objetivo é encontrar uma certa estrutura que satisfaça um conjunto de propriedades dadas. Se as propriedades a que certa estrutura devem satisfazer envolverem certos critérios de otimização, então o problema torna-se um *Problema de Otimização*. Logo é possivel formular um problema de decisão cujo objetivo é indagar se existe ou não a mencionada estrutura, satisfazendo às propriedades dadas. Isto é, existem triplas de problemas, um de decisão, outro de localização e outro de otimização que podem ser associados, dando origem a um problema de otimização combinatória. Denotamos $\Pi(D, Q)$ (ou simplesmente Π), um problema de otimização combinatória onde D é o conjunto de instâncias e Q é a questão solicitada.

Como ilustração desses conceitos consideremos o problema do caixeiro viajante que é o nosso foco de interesse neste trabalho.

Seja G um grafo (veja figura 2.3), tal que a cada aresta e esta associado um número real positivo $c(e)$ denominado custo da aresta e. Um *passeio fechado do*

caixeiro viajante é simplesmente um ciclo hamiltoniano de G. O custo de um passeio é a soma dos custos das arestas que o formam. Um *passeio de caixeiro viajante ótimo* é aquele cujo custo é mínimo. Um passeio de caixeiro viajante para o grafo da figura 2.3 é, por exemplo, v_1, v_2, v_3, v_4, v_5, cujo custo é 24 , enquanto que um passeio ótimo é v_1, v_2, v_4, v_3, v_5 de custo 19.

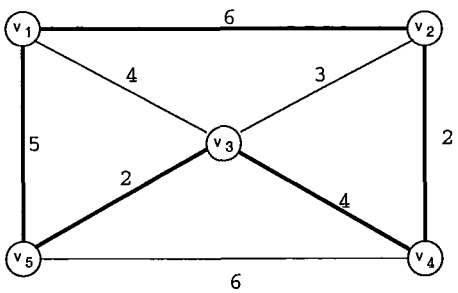


Figura 2.3: Grafo G com custos

Observa-se que os três problemas guardam em geral relação muito próxima, isto é, obter um passeio ótimo do caixeiro viajante, três problemas devem ser resolvidos em sequência:

Problema de decisão

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

Questão: Existe um passeio de caixeiro viajante em G de custo $\leq k$?

Se a resposta for "SIM", resolve-se o segundo problema

Problema de localização

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

Questão: Encontrar, em G, um passeio de caixeiro viajante de custo $\leq k$.

Resolver este problema consiste em obter um passeio do caixeiro viajante em G,

Problema de otimização

Instância: Um grafo G .

Questão: Localizar, em G, um passeio de caixeiro viajante ótimo.

Os três problemas de caixeiro viajante acima, estão relacionados. Suponha que o *Problema de Otimização* respectivo seja resolvido e denomine por Q o passeio ótimo encontrado. Então Q pode ser utilizado para resolver o *Problema de Localização* associado, da seguinte maneira: Seja $c(Q)$ o custo do passeio Q. Note que $c(Q)$ pode ser obtido somando os custos das arestas de Q. Então se $c(Q) \leq k$, Q é também uma solução para o *Problema de localização*. Caso contrário, $c(Q) > k$ e não existe em G passeio de caixeiro viajante de custo $\leq k$. Obviamente, isto resolve também o *Problema de Decisão* associado. Então para o caso de caixeiro viajante, o *Problema de Decisão* é de dificuldade não maior do que o de *Localização*, e este de dificuldade não maior do que o de *Otimização*. Por outro lado, em geral não é intuitivo que os problemas de otimização e localização apresentam ambos dificuldades não maior do que o da decisão associado.

Um algoritmo (determinístico) para um problema de decisão Π é *eficiente* ou *polinomial* se existe um polinomio $p(n)$ tal que este algoritmo pode resolver qualquer problema de tamanho n num tempo $O(p(n))$, sendo n o tamanho dos dados da entrada. Se considera aos problemas com solução polinomial como problemas tratáveis, e problemas sem solução polinomial como problemas intratáveis.

Vamos considerar os problemas a seguir como sendo problemas de decisão. Em geral os problemas em sua versão de decisão são mais fáceis de serem analisados. Por isso, se existir alguma forma de sua possível intratabilidade, podemos estendé-la as outras versões.

Define-se a classe P dos problemas de decisão como sendo aquela que compreende precisamente aqueles que admitem algoritmo polinomial.

Define-se a classe NP como sendo o conjunto de todos os problemas de decisão, tais que existe uma justificativa à resposta *SIM*, cujo passo de reconhecimento pode ser realizado em tempo polinomial.

Para verificar se um problema dado pertence ou não a NP seguimos o roteiro seguinte:

1. Define-se uma justificativa conveniente para a resposta *SIM* ao problema,

2. Elabora-se um algoritmo para reconhecer se a justificativa está correta. A entrada desse algoritmo consiste da justificativa e da entrada do problema dado.

Se o algoritmo resultante do Passo 2 for polinomial no tamanho da entrada do problema, então o problema pertence a NP. O caso contrário, naturalmente, não implica que não pertença a NP.

E finalmente nesta seção introduzimos a definição de redução ou transformação polinomial de dois problemas de decisão.

Sejam $\Pi_1(D_1, Q_1)$ e $\Pi_2(D_2, Q_2)$ dois problemas de decisão. Dizemos que existe uma *transformação polinomial* (veja figura 2.4) de Π_1 em Π_2 se existe uma função $f : D_1 \rightarrow D_2$ tal que as seguintes condições são satisfeitas:

(i) f pode ser calculada em tempo polinomial, e

(ii) dada qualquer entrada $I \in D_1$ para o problema Π_1, pode-se construir uma entrada $f(I) \in D_2$ para o problema Π_2 no tamanho $f(I)$ da entrada $I \in D_1$, tal que I é uma instância *SIM* para Π_1 se e somente se $f(I)$ é uma instância *SIM* para Π_2.

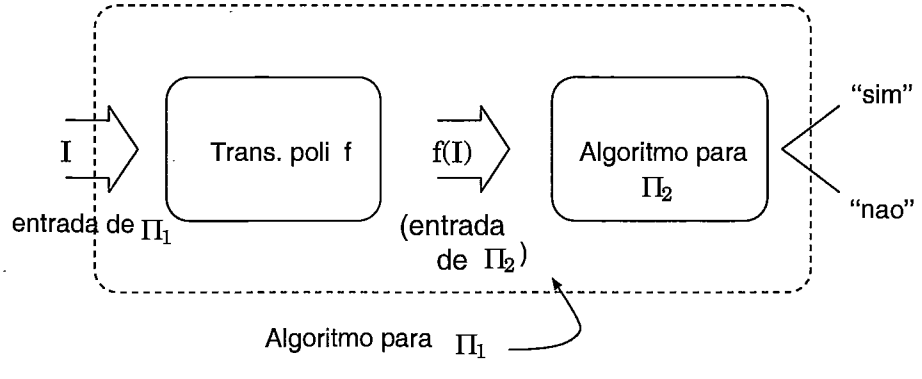


Figura 2.4: Transformação polinomial do problema Π_1 em Π_2

Um problema de decisão Π pertence à classe $NP-completo$ quando as seguintes condições forem satisfeitas:

(i) O problema $\Pi \in NP$, e

(ii) Para todo problema $\Pi' \in NP$ existe uma transformação polinomial de Π' em Π.

Quando existir uma transformação polinomial de Π_1 e Π_2 denotaremos $\Pi_1 \propto \Pi_2$ para indicar que Π_1 pode ser transformado (ou reduzido) polinomialmente em Π_2.

Note também que a relação $\propto$ é transitiva, i.e., $\Pi_1 \propto \Pi_2$ e $\Pi_2 \propto \Pi_3$ então $\Pi_1 \propto \Pi_3$. Se f_1 é uma função redução de Π_1 a Π_2 e f_2 é uma redução de Π_2 a Π_3 então $f_2(f_1)$ é uma função redução de Π_1 a Π_3 e se ambas f_1 e f_2 são calculadas polinomialmente, entao $f_2(f_1)$ tambem o é.

As classes P e NP estão relacionadas entre si. Primeiramente observamos o resultado $P \subseteq NP$ (como mostrado no esquema da figura 2.5). Todo problema de decisão solúvel por um algoritmo determinístico polinomial é também solúvel por um algoritmo que pode ser verificado em tempo polinomial. Se existir um algoritmo polinomial para a resolução de algum problema NP-completo, então todos os problemas da classe NP também poderão ser resolvidos em tempo polinomial. Também, se Π é NP-completo e se existir um algoritmo polinomial que resolve Π, então $\Pi \in P$. Logo cada $\Pi' \in NP$ estará também em P, o que implicará que $P \subseteq NP$ e conseqüêntemente $P = NP$.

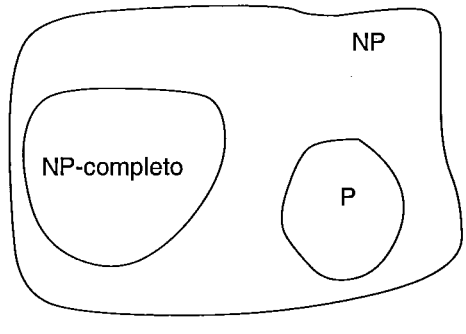


Figura 2.5: $P \subseteq NP$

Logo, um problema que pertence à classe P é chamado polinomial. Para um problema em que existe uma justificativa em tempo polinomial para uma resposta SIM o problema pertence à NP, e um problema que pertence à classe NP-completo é chamado NP-completo. Chamaremos problemas NP-árduos ou NP-difíceis a problemas de otimização cujo problema de decisão associado é NP-completo. (in-

dicamos como referências, [1], [15], e [36]). Uma boa introdução se encontra também em [20] e [21]. Para outras referências que definem a classe NP em termos de algoritmos não determinísticos, consultar [20].

2.3 Algoritmos aproximativos

Nesta seção define-se os algoritmos aproximativos [15] e [18]. Existem vários tipos de algoritmos para a solução de problemas NP-difíceis que encontram uma solução viável. Admitindo-se $P \neq NP$, quando se estuda problemas de otimização NP-Completos, pode-se considerar algoritmos que não produzam necessariamente soluções ótimas, mas soluções próximas ao ótimo.

Um algoritmo que retorna soluções próximas ao ótimo é chamado de um *algoritmo aproximativo.* Estes algoritmos se desenvolveram como uma alternativa à intratabilidade de grande variedade de problemas importantes de otimização.

Vamos considerar os problemas a seguir como sendo problemas de otimização. Seja Π um problema de otimização no qual cada solução viável tem um custo positivo, e deseja-se encontrar uma solução próxima ao ótimo. Dependendo do problema, uma solução ótima pode ser definida como um problema com custo máximo possível ou com custo mínimo possível, o problema pode ser de maximização ou de minimização respectivamente.

Os componentes básicos de um problema de otimização são o *conjunto de instâncias* ou *objetos de entrada*, o *conjunto de soluções viáveis* ou *objetos saída* associados a cada instância, e a função custo c que associa para cada instância $I \in \Pi$ e cada solução candidata $\sigma \in Q(I)$ um número racional positivo $c_\Pi(I, \sigma)$ chamado o *valor solução* ou custo para σ.

Se Π é um problema de minimização (maximização) então uma solução ótima para uma instância $I \in D$ é uma solução candidata $\sigma^* \in Q(I)$ tal que, para todo $\sigma \in Q(I)$,

$$0 < c(I, \sigma^*) \leq c(I, \sigma)$$

$$(0 < c(I, \sigma) \leq c(I, \sigma^*)),$$

e a razão de desempenho de um algoritmo de aproximação 'e $R(I, \sigma) = \frac{c(I,\sigma)}{c(I,\sigma^*)} (= \frac{c(I,\sigma^*)}{c(I,\sigma)})$.

Esta definição assume que $c(I, \sigma^*) > 0$, aplica-se para ambos problemas de minimização e maximização. A razão R são sempre bem definidos, desde que todas as soluções são assumidas tendo o custo positivo.

Define-se a seguir o conceito de algoritmo ε-aproximado.

Seja Π um problema de otimização e seja H um algoritmo aproximado para o problema Π e para qualquer instância I de este problema, retorna uma solução viável $H(I)$; denota-se a solução ótima de I por *OPT(I)*. Dado algum $\varepsilon > 0$, dizemos que H é um algoritmo ε-*aproximado* para Π se a razão de desempenho da solução viável $H(I)$ com respeito a I, verifica a seguinte desigualdade:

$$R(I, H(I)) \leq \varepsilon, \ \ \forall I \in \Pi$$

Se um problema de otimização admite um algoritmo $\varepsilon - aproximado$ em tempo polinomial dizemos que este problema é aproximável dentro da razão ε.

Ou seja, considerando que H retorna uma solução aproximada para um problema de minimização:

$$c(I, H(I)^*) \leq c(I, H(I))$$

como

$$R(I, H(I)) \leq \varepsilon$$

temos que

$$\frac{c(I, H(I))}{c(I, H(I)^*)} \leq \varepsilon$$

ou ainda

$$c(I, H(I)) \leq \varepsilon c(I, H(I)^*) \leq c(I, H(I)^*) + \varepsilon c(I, H(I)^*) = (1 + \varepsilon) c(I, H(I)^*) \quad (2.1)$$

o que implica que

$$c(I, H(I)) \leq (1 + \varepsilon) c(I, H(I)^*), \ \ 0 < \varepsilon < 1$$

Para o caso de problemas de maximização:

$$c(I, H(I)) \leq c(I, H(I)^*),$$

como

$$R(I, H(I)) \leq \varepsilon$$

temos que

$$\frac{c(I, H(I)^*)}{c(I, H(I))} \leq \varepsilon$$

ou ainda

$$0 < c(I, H(I)^*) \leq \varepsilon c(I, H(I))$$

e assim

$$-\varepsilon c(I, H(I)) \leq -c(I, H(I)^*)$$

como $c(I, H(I)^*) \geq c(I, H(I))$ temos que

$$-\varepsilon c(I, H(I)^*) \leq -\varepsilon c(I, H(I)) \leq -c(I, H(I)^*) \leq c(I, H(I)) - c(I, H(I)^*) \quad (2.2)$$

logo

$$c(I, H(I)) \geq (1 - \varepsilon) c(I, H(I)^*), \ 0 < \varepsilon < 1$$

De (2.1) e (2.2) temos que

$$-\varepsilon c(I, H(I)^*) \leq c(I, H(I) - c(I, H(I)^*) \leq \varepsilon c(I, H(I)*)$$

e em qualquer caso: $\frac{|c(I,H(I))-c(I,H(I)^*))|}{c(I,H(I)^*)} \leq \varepsilon$, o que justifica a definição.

No caso de minimização $(1 + \varepsilon)$ será a razão de desempenho. Portanto quanto menor o valor de ε, melhor o resultado a aproximação.

Um esquema de aproximação para um problema Π é um algoritmo ε - aproximado H que deve funcionar para qualquer valor de ε sendo portanto mais rigoroso que um Algoritmo ε-Aproximado.

Seja Π um problema de otimização. Um algoritmo H é dito um *Esquema de Aproximação* para Π se para qualquer instância $I \in \Pi$ e dado $\varepsilon > 0$, $H(I, \varepsilon)$ retorna uma solução viável de I cuja razão de desempenho é no máximo ε, isto é,

$$\frac{|c(I,\sigma) - c(I,\sigma^*)|}{c(I,\sigma^*)} \leq \varepsilon, \;\; c(I,\sigma^*) > 0.$$

Um esquema de aproximação H é *polinomial* se $\forall \varepsilon > 0$ fixo, seu tempo de computação é limitado por uma função polinomial no tamanho dos dados de entrada. Por outro lado, é dito *totalmente polinomial* se seu tempo de computação é limitado por uma função polinomial, tanto no tamanho dos dados de entrada quanto em $(1/\varepsilon)$.

Capítulo 3

O problema do caixeiro viajante

Um dos problemas clássicos e mais conhecidos de otimização combinatória é o Problema do Caixeiro Viajante (PCV).

Este problema pode ser descrito simplesmente como:

Dadas n cidades: $1, 2, \ldots, n$, e uma distância inteira não negativa d_{ij} entre quaisquer duas cidades i e j.

Um vendedor, começando de uma qualquer dessas cidades deseja visitar exatamente uma vez cada cidade dessa lista, e então voltar para a cidade de origem. O problema consiste em achar a ordem das cidades que ele deve percorrer de maneira que o total das distâncias percorridas a partir da cidade inicial até a seu retorno a essa mesma cidade seja o menor possível.

Obviamente, esse problema pode ser resolvido pela "força bruta", i.e., enumerando todas as possíveis soluções, calculando o custo de cada uma, e pegando a melhor. Isso levaria um tempo proporcional a $n!$ (existem $\frac{1}{2}(n-1)!$ alternativas a serem consideradas). Embora o PCV seja um dos problemas mais intensamente estudados, não se conhece até hoje algoritmos polinomiais que o resolvam.

3.1 Histórico

Em qualquer problema matemático, há três aspectos de sua história que devem ser considerados: como surgiu, como a sua pesquisa influenciou outros desenvolvimentos em matemática, e como o problema foi finalmente resolvido.

No caso do PCV, como o problema consiste em desenvolver um algoritmo que satisfaça padrões formais (ou informais) de eficiência, ele ainda não foi resolvido. Sem exagero, podemos afirmar que o *PCV* é o mais famoso dos problemas não resolvidos de otimização combinatória. Foi o primeiro problema descrito no livro "Computers and Intractability" de Garey e Johnson [15], considerado a referência fundamental em Complexidade Computacional. É também mais usado informalmente em conversas como padrão de comparação. Por exemplo: " O problema X é tão difícil como o PCV". Tudo isso faz com que o PCV continue a influenciar fortemente o desenvolvimento de conceitos de otimização, algoritmos e complexidade.

Sua história data desde 1930 por Menger [27] e [28] o precursor mais direto e o primeiro uso do termo "problema do caixeiro viajante" em círculos matemáticos pode ter surgido em 1931-32, em conversas de corredor em Princeton. Atribui-se seu uso a Hassler Whitney que na época fazia seu pós-doutorado lá, trabalhando em Teoria dos Grafos.

No entanto, esse termo é conhecido desde 1832 num livro editado na Alemanha sob o título: "O Caixeiro Viajante, como êle deve ser e o que deve fazer para ganhar Comissoẽs e ser bem sucedido nos seus Negócios, por um Caixeiro Viajante veterano" [38]. Embora, esse livro não seja um livro matemático, no último capítulo êle descreve o problema do PCV em sua essência, usando as seguintes expressões:

"por uma escolha apropiada e escalonamento do tour, muitas vezes pode-se ganhar muito tempo,... O aspecto mais importante é cobrir o maior número de locais possíveis, sem visitar o mesmo local duas vezes ... "

Mas, quem sem dúvida popularizou o termo na comunidade matemática e na comunidade de pesquisa operacional foi Merril Flood, que era um estudante de pós -

graduação em Princeton em 1932, e posteriormente trabalhou na Rand Corporation que se tornou na época o centro intelectual de Teoria de Pesquisa Operacional.

Outra razão para a rápida popularização do *PCV* pode ser atribuida a sua conexão íntima com alguns tópicos importantes em problemas combinatórios, que surgiram com a programação linear, como problemas de tarefas e mais geralmente, o problema de transporte [17] em 1941.

De três artigos fundamentais de Cook (1971), Karp (1972) e Levin (1973) ao redor de 1970, emergiu a idéia de que muitos dos problemas considerados inerentemente difíceis eram computacionalmente equivalentes, no sentido que se existisse um algoritmo polinomial para um deles, esse algoritmo poderia ser usado para resolver todos os outros em tempo polinomial. Claro que, a provável intratabilidade do *PCV* geral não exclui a possibilidade da existência de algoritmos polinomiais para casos especiais do problema, como o *PCV* euclideano onde as cidades são determinados por pontos no plano e as distâncias deles são calculadas de acordo com a métrica Euclideana. Varios métodos aproximativos foram propostos, alguns deles com boas garantias de desempenho como o de Christofides [10] que admite um erro de 50% no pior caso para o *PCV* euclideano. E Sahni e Gonzales [35] mostraram que garantir qualquer erro máximo fixo é tão difícil quanto achar o ótimo.

3.2 Formulação

O problema do caixeiro viajante está relacionado com o problema de determinar um ciclo hamiltoniano num grafo: Um caixeiro visitante deseja visitar um número n de cidades, a fim de vender seus produtos. Dados um conjunto de cidades $\{1, 2, \ldots, n\}$ e as distâncias $[d_{ij}]$ entre as cidades i e j, a nossa pergunta seria, em qual ordem ele as percorrerá, a fim de visitar cada cidade precisamente uma única vez, com o mínimo de quilometragem percorrida? Em outras palavras procuramos um ciclo hamiltoniano (ou tour) que tenha comprimento (ou custo) mínimo dentre todas as possíveis escolhas.

Por conveniência, considera-se uma métrica de distâncias $[d_{ij}]$ simétricas, i.e., uma matriz $n \times n$ de distâncias $[d_{ij}]$, com entradas reais positivas, onde

$$d_{ij} = \begin{cases} d_{ji} & i \neq j, \\ 0 & i = j \end{cases}$$

Dizemos que $[d_{ij}]$ satisfaz a desigualdade triangular se

$$d_{ij} + d_{jk} \geq d_{ik}$$

para todo $1 \leq i, j, k \leq n$

Observação: A restrição de desigualdade triangular diz essencialmente que ir da cidade i para a cidade k passando pela cidade j não pode ser mais barato que ir diretamente de i para k. Isto é muito razoável desde que a visita imposta a cidade j aparece como restrição adicional, o que pode somente aumentar o custo. A desigualdade triangular é automáticamente satisfeita, por exemplo, sempre que a matriz de distâncias é induzida por uma métrica, como é o caso especial de *matrizes de distâncias euclideanas*, no qual cada cidade representa um ponto p_j no plano com coordenadas (x_j, y_j), e

$$d_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}$$

Outra classe importante de matrizes de distância que automaticamente satisfazem a desigualdade triangular são as matrizes fecho. Dizemos que uma matriz $[\bar{d}_{ij}]$ é o fecho de $[d_{ij}]$ se $\bar{d}_{ij}$ é o comprimento de menor caminho de i para j no grafo completo de n vértices $\{1, 2, \ldots, n\}$, onde o comprimento da aresta $[i, j]$ é d_{ij}. O fecho de uma matriz de distâncias $n \times n$ pode ser calculado em $O(n^3)$ vezes pelo algoritmo Floyd- Warshall[31] que encontra o caminho mais curto entre dois pares de vértices (o método trabalha com uma matriz $n \times n$ de números d_{ij}).

Por exemplo:

A matriz em (a) na figura 3.1 não satisfaz a desigualdade triangular pois,

$d_{34} > d_{32} + d_{24}$

A matriz em (b) na figura 3.1 satisfaz a desigualdade triangular, pois, é o fecho, obtida de (a).

$$\begin{bmatrix} 0 & 3 & 6 & 4 & 10 \\ 3 & 0 & 4 & \textcircled{2} & 6 \\ 6 & \textcircled{4} & 0 & \textcircled{7} & 8 \\ 4 & 2 & 7 & 0 & 4 \\ 10 & 6 & 8 & 4 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 3 & 6 & 4 & 8 \\ 3 & 0 & 4 & 2 & 6 \\ 6 & 4 & 0 & 4 & 8 \\ 4 & 2 & 4 & 0 & 4 \\ 8 & 6 & 8 & 4 & 0 \end{bmatrix}$$

(a) (b)

Figura 3.1: (a)Matriz em (b) é a matriz fecho de (a)

O fecho $[\bar{d}_{ij}]$ de qualquer matriz de distância $[d_{ij}]$ satisfaz a desigualdade triangular porque, se $\bar{d}_{ik} > \bar{d}_{ij} + \bar{d}_{jk}$, então $\bar{d}_{ik}$ não é o comprimento de um caminho mínimo de i até k.

Apesar do problema do Caixeiro Viajante ser fácilmente formulado, a sua resolução computacional não é tão simples, uma vez que os métodos exatos demandam muito tempo para problemas de tamanho razoavelmente grandes e os métodos aproximados, em geral bons, porém nem sempre garantem o ótimo.

Supondo que seja dada uma lista com n cidades a um caixeiro viajante, onde cada par de cidades i e j são ligadas por um arco de custo c_{ij}, o problema é encontrar um tour de custo mínimo de modo que, ele passe em cada cidade apenas uma vez e depois retorne para o seu ponto de partida.

O problema do caixeiro viajante (PCV) é formulado a partir do Problema de Alocação, utilizando variaveis x_{ij}. Define-se $x_{ij} = 1$ quando o caixeiro viajante vai da cidade i para a cidade j, i.e., a aresta $(i, j) \in E$ está na solução e $x_{ij} = 0$ caso contrário.

Dado $G = (V, E)$ um grafo com $|V| = n > 2$ e $C = (c_{ij})$ uma matriz $n \times n$ onde cada elemento $c_{ij} > 0$, associado a cada uma das arestas $(i, j) \in E$ que representa a distância entre as cidades i e j $(i \neq j)$, para todo $i = 1, \ldots, n$. O modelo clássico proposto por Dantzig, Fulkerson e Johnson [14] do PCV em variáveis inteiras $0 - 1$ é:

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

sujeito a

$$\sum_{j=1}^{n} x_{ij} = 1, \text{ para todo } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \text{ para todo } j = 1, \ldots, n$$

$$\sum_{i \in S} \sum_{j \in N \setminus S} x_{ij} \geq 1, \ \emptyset \neq S \subset \{1, \ldots, n\}$$

$$x_{ij} \in \{0, 1\}, \text{ para todo } (i, j) \in E$$

Os tipos de problemas do Problema do Caixeiro Viajante de acordo quanto a natureza de seus elementos de custo podem-se classificar como:

1º *PCV* Simétrico - quando $c_{ij} = c_{ji}$, para todo $i, j \in V$

2º *PCV* Assimétrico - quando c_{ij} não é necessariamente igual a c_{ji}.

3º *PCV* Simétrico e satisfazendo a desigualdade triangular, i. e.,

$c_{ij} = c_{ji}$ e $c_{ik} \leq c_{ij} + c_{jk}$, para todo $i, j, k \in V$, serão chamados *Euclideanos*.

Os c_{ij}'s podem ser determinados aleatoriamente ou serem especificados a partir de alguma aplicação particular.

A seguir formulamos o *PCV* simétrico. Usa-se a variavel x_{ij} para representar a aresta entre as cidades i e j, $1 \leq i < j \leq n$, tomando $x_{ij} = 1$ quando a aresta $(i, j) \in E$ está no tour

$$\text{minimizar} \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij}\, x_{ij}$$

sujeito a

$$\sum_{i=k \text{ ou } j=k} x_{ij} = 2, \ 1 \leq k \leq n$$

$$\sum_{|S \cap \{i,j\}|=1} x_{ij} \geq 2, \text{ para todo }, S \subset \{1, \ldots, n\}$$

$$x_{ij} \in \{0, 1\}, \ 1 \leq i < j \leq n$$

3.3 Problema do Caixeiro Viajante é *NP*-completo

Dado que o PCV é polinomialmente equivalente ao PCV de decisão associado, nesta seção prova-se que o problema do caixeiro viajante (PCV) é NP-completo na sua versão de decisão, e vamos considerar o caso do problema do caixeiro viajante simétrico. Para este resultado consideramos dois problemas intermediários: Cobertura de Vértices (CV) e Ciclo Hamiltoniano (CH). Demonstra-se que

$$3 - SAT \propto CV \propto CH \propto PCV$$

para atingir o objetivo. Esta abordagem indireta é para provar simplesmente que

$$3 - SAT \propto PCV$$

diretamente. A idéia geral é utilizar a redução polinomial: $SAT \propto \Pi$, onde Π é um problema de decisão $NP - completo$. $SAT \propto 3 - SAT$ ($3 - SAT$ é NP-completo [15]). Começamos formulando os problemas de decisão do PCV, CH e CV.

Problema: CAIXEIRO VIAJANTE SIMÉTRICO

Instância: Uma matriz de distâncias d_{ij} e um inteiro B, as distâncias reais positivos satisfazendo a desigualdade triangular, i.e., $d_{ij} + d_{jk} \geq d_{ik}$, $\forall\ i, j$ e k

Questão: Existe um tour de n cidades cujo comprimento no máximo é B, i.e., uma ordem $(i_1, i_2, \ldots, i_n)$ das cidades $1, 2, \ldots, n$ de todas as possíveis maneiras tal que a soma $d_{i_1 i_2} + d_{i_2 i_3} + \ldots + d_{i_n i_1} \leq B$?

Problema: CICLO HAMILTONIANO

Instância: Grafo $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$

Questão: O grafo G admite um ciclo hamiltoniano? isto é, uma ordem $v_1, \ldots v_n$ de vértices, onde $n = |V|$, tal que $\{v_i, v_{i+1}\}$ esta em E, $1 \leq i \leq n$ e $\{v_n, v_1\}$ esta em E?

Problema: COBERTURA DE VÉRTICES

Instância: Grafo $G = (V, E)$ e um inteiro positivo k.

Questão: O grafo G admite uma cobertura de vértices, i.e., existe um subconjunto $V' \subseteq V$, $|V'| = k$ tal que para cada aresta $(u, v) \in E$ ou $u \in V'$ ou $v \in V'$ (ou ambos). O tamanho de uma cobertura é a cardinalidade de seu conjunto de vértices.

A classe de problemas $NP-completos$ tem a seguinte propriedade [12]: se existir algum problema $NP-completo$ que pode ser resolvido em tempo polinomial, então *todo* problema em NP tem solução em tempo polinomial, i.e., $P = NP$. O ciclo hamiltoniano é um problema $NP-completo$ [15] e [40], e damos a seguir a prova desse fato, considerando que o problema Cobertura de Vértices é $NP-completo$ (veja a prova em [15]).

Teorema 3.1. *O problema Ciclo Hamiltoniano é NP-completo*

Prova:

(i) O problema Ciclo Hamiltoniano pertence a classe NP.

De fato, um algoritmo que resolve um problema que pertence a classe NP precisa apenas dar uma ordenação dos vértices do grafo e verificar em tempo polinomial se todas as arestas correspondentes pertencem ao conjunto de arestas do grafo dado.

(ii) Cobertura de Vértices se reduz polinomialmente ao Ciclo Hamiltoniano.

Para provar o item (ii) considere uma instância arbitrária de Cobertura de Vértices dada (que é $NP-completo$ [1]) pelo grafo $G = (V, E)$ e um inteiro positivo $k \leq |V|$. Vamos construir um grafo $G' = (V', E')$ tal que G' tem um ciclo hamiltoniano se e somente se G tem uma cobertura de vértices de tamanho menor ou igual a k.

Para a construção de G' considera-se 2 tipos de subgrafos que serão conectados por arestas adicionais. O primeiro tipo de subgrafo é o de vértices simples s_i para $1 \leq i \leq k$, da figura 3.2(a), chamados *vértices seletores*, que será usado para selecionar k vértices do conjunto de vértices V de G. Para cada aresta $e = \{u, v\}$ em

E, constrói-se um segundo tipo de subgrafo, ilustrado na figura 3.2(b).Este subgrafo tem 12 vértices: 6 correspondendo a u e 6 correspondendo a v, e 14 arestas que serão usadas na transformação CV para o CH. Chamaremos a este segudo tipo de subgrafo de *componente cobertura-teste.*

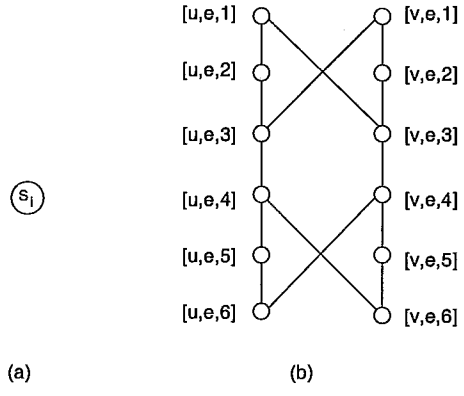


Figura 3.2: (a) Vértices seletores, $1 \leq i \leq k$. (b) O componente (subgrafo) cobertura-teste para a aresta $e = \{u, v\}$ usada na transformação cobertura de vértices para o ciclo hamiltoniano.

Estes vértices e arestas do componente (figura 3.2(b)) são especificados como segue:

$$V'_e = \{(u, e, i), (v, e, i) : 1 \leq i \leq 6\}$$

$$E'_e = \{\{(u, e, i), (u, e, i+1)\}, \{(v, e, i), (v, e, i+1)\} : 1 \leq i \leq 5\}$$

$$\cup\{\{(u, e, 3), (u, e, 1)\}, \{(v, e, 3), (v, e, 1)\}\}$$

$$\cup\{\{(u, e, 6), (u, e, 4)\}, \{(v, e, 6), (v, e, 4)\}\}$$

As arestas adicionais na construção final servirão para unir pares de componentes cobertura-teste ou para unir componentes cobertura-teste a um vértice seletor. Quaisquer das arestas adicionais somente envolvem os vértices extremos $[u, e, 1]$, $[v, e, 1]$, $[u, e, 6]$, $[v, e, 6]$. Adicionamos como segue: se assume que os vértices V de G são ordenados como $V = \{v_1, \ldots, v_n\}$. Para cada vértice v_i suas arestas podem ser ordenadas como $e_1, \ldots, e_{grau(i)}$ onde $grau(i)$ é o grau do vértice v_i, tal que para todo $e_j = \{v_i, v_p\}$ e $e_k = \{v_i, v_q\}$, $j < k$ se e somente se $p < q$. Conectamos os

lados v_i dos componentes cobertura-teste para $e_1, \ldots, e_{grau(i)}$ adicionando arestas $E_i' = \{\{[i, e_j, 6], [i, e_{j+1}, 1]\}, 1 \leq j \leq grau(i)\}$.

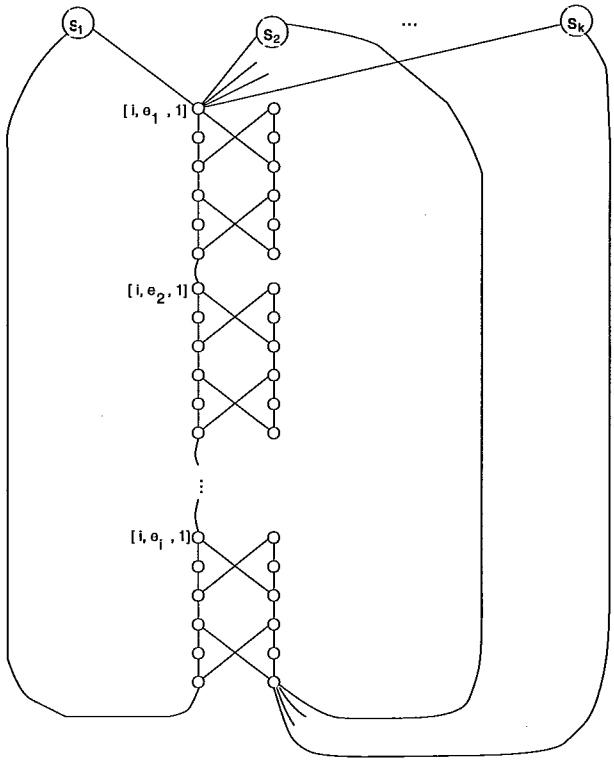


Figura 3.3: Caminho de todos os componentes cobertura-teste para as arestas de E tendo o vértice i como extremo.

Para completar o grafo $G' = (V', E')$ a ultima conexao de arestas em G' une o primeiro e último vértice de cada um desses caminhos para cada um dos vértices seletores $s_1, s_2, \ldots, s_k$.

Estas arestas são especificadas como segue:

$E'' = \{\{s_j, (i, e_1, 1)\}, \{s_j, (i, e_{grau(i)}, 6)\} : \forall j, 1 \leq j \leq k\}$.

Logo o grafo $G' = (V', E')$ tem

$$V' = \{s_j : 1 \leq i \leq k\} \cup (\bigcup_{e \in E} V_e')$$

e

$$E' = (\bigcup_{e \in E} E_e') \cup (\bigcup_{v \in V} E_i') \cup E''$$

Isto completa a construção de G', ilustrado na figura 3.3. (Observe que é fácil ver que G' pode ser construido a partir de G, e k em tempo polinomial).

Então, agora mostra-se que G tem uma cobertura de vértices V_1 com $|V_1| \leq k$, se e somente se G' tem um ciclo hamiltoniano. De fato:

Suponha que G tem uma cobertura de vértices V_1 de tamanho no máximo k então podemos assumir que $|V_1| = k$, (se $|V_1| < k$, adicionamos $k - |V_1|$ vértices de $V - V_1$ para V_1, estes vértices adicionais não afetam o estado dos V_1's como um conjunto de cobertura de vértices.) Seja $V_1 = \{v_{i(1)}, \ldots, v_{i(k)}\}$ onde $1 \leq i(1) <, \ldots, < i(k) \leq n$.

Constrói-se um ciclo hamiltoniano começando pelo vértice seletor s_1. Começando em s_1 nós "caminhamos" para $[i(1), e_1, 1]$, entrando no primeiro componente cobertura-teste de $v_{i(1)}$. Existem dois caminhos que atravessam este componente que sai em $[i(1), e_1, 6]$ e não visitam um vértice duas vezes, (estes são mostrados na figura 3.4, caminho (a) é tomado se o vértice v_i esta em V_1, enquanto que o caminho (b) é tomado se v_i não está em V_1) até atingir o vértice $[i(1), e_{grau(i(1))}, 6]$, o último vértice dos componentes cobertura-teste para $v_{i(1)}$. Portanto, caminhamos para s_2 ao longo da aresta $\{[i(1), e_{grau(i(1))}, 6], s_2\}$.

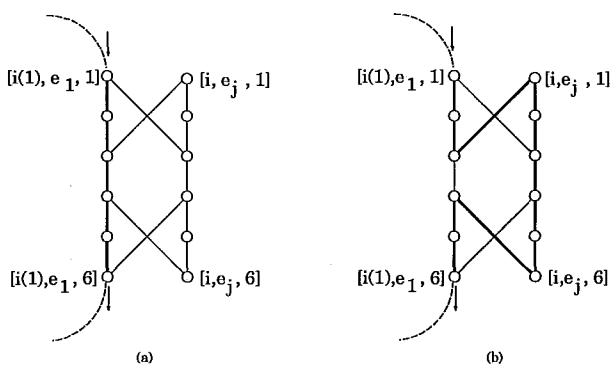


Figura 3.4: As duas possíveis configurações de um ciclo hamiltoniano dentro do componente cobertura-teste

Seguimos um caminho similar para $v_{i(2)}$, o qual nos leva a s_3. Isto se repete para $v_{i(3)}, \ldots, v_{i(k)}$, exceto que a aresta final para $v_{i(k)}$ é $\{[i(k), e_{grau(i(k))}, 6], s_1\}$ a qual completa o ciclo, com isto obtém-se um ciclo fechado que é um ciclo hamiltoniano. Primeiro porque o ciclo não visita qualquer vértice de V' duas vezes, que é válido para vértices seletores e para os vértices $[i(j), e_k, l]$ pela construção. Se $[i, e_k, l]$ é visitado durante o ciclo, então isto é devido a que o outro vértice em e_k esta em V_1. Isto significa que o ciclo contém um caminho da forma mostrada na figura 3.4(b).

Mas isto implica que todos os vértices $[i, e_k, l]$ $1 \leq l \leq 6$, são visitados. Somente as outras arestas unidas a eles são as arestas que se comunicam a v_i, o qual liga este componente a outros componentes v_i ou a vértices seletores. Pela escolha do ciclo isto implica que nenhum dos vértices $[i, e_k, l]$, $1 \leq l \leq 6$ são visitados duas vezes. Para os vértices que estão em V' obtidos dos vértices em $V - V_1$, considere o vértice $[i, e_k, l]$, para algum l, $1 \leq l \leq 6$, onde v_i não esta em V_1. Seguidamente, u outro vértice v_j em e_k deve estar em V_1. Isto implica que $[j, e_p, 1]$, onde $e_p = e_k$, é visitado pelo ciclo e, portanto, o ciclo visita $[i, e_k, l]$, $1 \leq l \leq 6$, usando o caminho da figura 3.4(b). Logo o ciclo visita cada vértice em V' e é, por conseguinte, um ciclo hamiltoniano.

Agora provamos o reverso. Suponha que G' tem um ciclo hamiltoniano. Particiona-se o ciclo em k caminhos. Cada caminho começa em um vértice seletor, e termina em um vértice seletor e não passa por um outro vértice seletor. Consideramos o caminho começando por s_1. O segundo vértice sobre o caminho é ou $[i, e_1, 1]$ para algum i, ou $[j, e_{grau(j)}, 6]$, para algum j. Assumimos que este é $[i, e_1, 1]$; o outro caso é simétrico.

A figura 3.5 mostra as escolhas para um outro caminho diferente que as duas mostradas na figura 3.4.

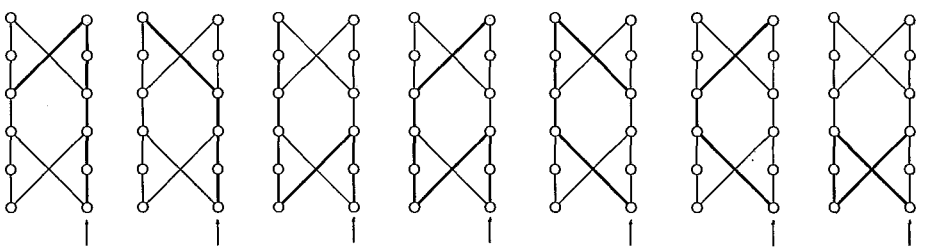


Figura 3.5: Escolhas diferentes dos componentes cobertura-teste

Cada um deles ocasiona que um vértice intermediário no componente possa ser omitido. Pois, como nós argumentamos acima, um vértice intermediário em um componente não tem outras arestas em E' que as únicas do componente. Mas este caminho é parte de um ciclo hamiltoniano; portanto, os vértices intermediarios sob o lado que entra do componente devem somente ser visitados e os caminhos que fazem isto são apenas os dois da figura 3.4. Assim o caminho entra em $[i, e_1, 1]$ e sai

em $[i, e_1, 6]$. Há somente uma aresta desde $[i, e_1, 6]$ a qual conduz ou para $[i, e_2, 1]$ ou para um vértice seletor. Se for um vértice seletor, esta completo o caminho. No caso em que conduz para $[i, e_2, 1]$ um argumento similar mostra que o caminho deixa o segundo componente no vértice $[i, e_2, 6]$. Finalmente o caminho deixa o grau(i)-ésimo componente de v_i no vértice $[i, e_{\text{grau}(i)}, 6]$ para algum vértice seletor.

Nós mostramos então que este caminho deveria passar através de todos os componentes correspondentes a algum vértice v_i em V. Além do mais, o caminho passa através de cada componente que tem uma das duas formas mostradas na figura 3.4.

Os k vértices de V obtidos desta forma formam uma cobertura de vértices de G, por argumentos similares usados na primeira parte da prova.

Como o ciclo passa por cada $[x, y, z]$ (tendo $x = v$) exatamente uma vez, cada aresta $(i, j) \in E$ é coberta especificamente por um dos k vértices do grafo G, os quais forman uma cobertura de tamanho k em G.

Concluimos que G tem uma cobertura de vértices se e somente se G' tem um ciclo hamiltoniano

Portanto, o problema do ciclo hamiltoniano pertence a NP e cobertura de vértices $\propto$ ciclo hamiltoniano, logo o problema do ciclo hamiltoniano é $NP -$ *completo.* ∎

O problema do ciclo hamiltoniano é um caso especial do *PCV.* Agora prova-se que o problema do caixeiro viajante com desigualdade triangular é NP-completo. Para este objetivo, dado qualquer grafo $G = (V, E)$, constrói-se uma instância de $|V|$ cidades para o *PCV* e temos que encontrar um tour de custo mímimo. Para isto mostra-se que o Problema do Ciclo Hamiltoniano(CH) se reduz ao Problema do Caixeiro Viajante (PCV), logo o PCV é $NP - completo$ [31]. O seguinte corolario prova o caso particular par instâncias que satisfazem a desigualdade triangular.

Corolário 3.2. *O* PCV *com desigualdade triangular (*Δ PCV*) é NP-completo.*

Prova: Temos que provar as duas seguintes condições:

(i) Dado uma instância do problema do caixeiro viajante , usamos como garantia a seqüência de $|V| = n$ vértices no tour. O algoritmo de verificação verifica se esta seqüência contém cada vértice exatamente uma vez, somando o custo das arestas, sendo a soma o máximo de L. Este processo é efetuado em tempo polinomial, logo o Δ PCV pertence a classe NP.

(ii)Provaremos que o Δ PCV é *NP-completo*, reduzindo o problema do Ciclo hamiltoniano para o Δ PCV, mostrando que as instâncias do CH podem ser transformadas em instâncias do PCV.

Seja $G = (V, E)$ um grafo que determina uma instância do CH. Construimos uma instância G' do PCV: os vértices de G correspondem a cidades, $V = \{1, 2, \ldots, n\}$, onde $n = |V|$, e define-se a função distância d_{ij} entre quaisquer duas cidades i e j como

$$d_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E, \\ 2 & \text{caso contrário} \end{cases}$$

e o limite B definimos como n.

Então, agora temos que provar que: O grafo G tem um ciclo hamiltoniano se e somente se G' tem uma solução. Se $H = [i_1, i_2, \ldots, i_n]$ é um ciclo hamiltoniano de G, e cada aresta em E tem um custo de 1 em G', então o custo deste tour H em G' tem custo de exatamente B. Portanto , G' tem uma solução.

Reciprocamente, suponha que G' tem um tour $H' = [i'_1, \ldots, i'_n]$ de custo no máximo B. Como há n arestas neste tour e $B = n$, cada aresta pode corresponder a 1 unidade de distância, o que implica que H' é um ciclo hamiltoniano em G contendo unicamente arestas em E.

Portanto, há uma tranformação que é polinomial nos vértices de G, de CH para ΔPCV. Logo o Δ PCV é $NP - completo$. ∎

Capítulo 4

Algoritmos Aproximativos para o Problema do Caixeiro Viajante

Neste capítulo apresenta-se o estudo e análise de dois algoritmos aproximativos para a solução do problema do caixeiro viajante euclideano, simétrico e no plano: o algoritmo da Árvore e o algoritmo de Christofides ([25] e [31]). O primeiro produz um tour cujo custo é no máximo duas vezes o custo do tour ótimo, num tempo de ordem $O(n^2)$, o de Christofides produz o tour com um custo de no máximo 3/2 vezes o custo do tour ótimo, e num tempo de ordem $O(n^3)$. Estes tipos de algoritmos constroem um grafo gerador euleriano em vez de um tour, mas com a hipótese da desigualdade triangular válida, essas soluções podem ser transformadas em tours sem peso (custo) extra.

Mostra-se a seguir o teorema fundamental da teoria de complexidade, devido a Sahni e Gonzales [35], o qual restringe o comportamento de quaisquer algoritmos aproximativos para o *PCV*.

Entretanto a tarefa de desenvolver algoritmos aproximativos para o problema geral do *PCV* é tão sem esperanças quanto a de achar soluções exatas para ele.

O teorema a seguir, devido a Sahni e Gonzales [35], prova esse fato.

Teorema 4.1. *[35] Para qualquer $\varepsilon > 0$, existirá um algoritmo ε-aproximativo em tempo polinomial para o Problema Simétrico do Caixeiro Viajante* (PCVS) *se e*

somente se $P = NP$.

Prova: A prova é por absurdo. Suponha que existe algum $\varepsilon > 0$ e um algoritmo $\varepsilon - aproximativo$ A em tempo polinomial para o $PCVS$, então

$$\frac{S - OPT}{OPT} \leq \varepsilon_0$$

ou

$$S \leq (1 + \varepsilon_0)OPT$$

onde S é a solução viável obtida ao rodar o algoritmo A e OPT é a solução ótima do PCV

Mostra-se que tal algoritmo A pode ser usado para construir um algoritmo em tempo polinomial para o problema do ciclo hamiltoniano, o qual é NP-completo (Teorema 3.1), isto implicaria que o problema do ciclo hamiltoniano pertence a P, então $P = NP$ [12] e [26] e o resultado do teorema se verifica.

Seja $G = (V, E)$ uma instância do problema do ciclo hamiltoniano com n vértices. Queremos determinar quando G contêm um ciclo hamiltoniano usando a hipótese da existência do algoritmo ε - aproximativo A. Constrói-se a partir de G, uma instância do PCV simétrico e aplica-se o algoritmo A. Seja $G' = (V, E')$ um grafo completo em V, i.e., $E' = \{(i, j)/i, j \in V \text{ e } i \neq j\}$.

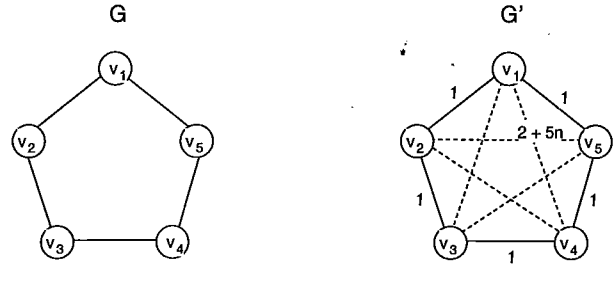


Figura 4.1: Grafo completo G' obtido de G

Seja o conjunto $V = \{1, 2, \ldots, n\}$ de vértices de G', com $n > 2$, onde esses vértices correspondem a cidades e define-se a função distância d_{ij} para cada aresta em E' como:

$$d_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E, \\ 2 + \varepsilon n & \text{caso contrário} \end{cases}$$

Agora considera-se o problema do caixeiro viajante (G', d). Observe que se o grafo original G tem um ciclo hamiltoniano H, então a função distância d associa para cada aresta em H uma distância de 1, e assim (G', d) contêm um tour de custo n que atravessa as cidades no ordem que os vértices correspondentes de G são atravessados por H, logo a solução ótima OPT é igual a n, então $S \leq (1 + \varepsilon_0)n$.

Por outro lado, se $S \leq (1 + \varepsilon_0)n$, então existe ciclo hamiltoniano em G, pois G tem pelo menos 3 vértices, então S usou pelo menos 3 arestas e se alguma delas tiver distância $(2 + \varepsilon_0 n)$ teremos uma contradição.

Assim, $S \leq (1 + \varepsilon_0)n \iff$ existe ciclo hamiltoniano em G

■

Impondo restrições essenciais sobre os tipos de instâncias permitidas, em particular, aplicações onde as distâncias possam obedecer a chamada desigualdade triangular, poderemos explorar a possibilidade de desenvolvimento de algoritmos aproximativos para o PCV.

A seguir apresentamos preliminares necessárias para os algoritmos da Árvore e de Christofides.

4.1 Multigrafos e passeios eulerianos

Lembra-se que o teorema 2.2 caracteriza a os multigrafos que são eulerianos: um multigrafo $G = (V, E)$ é euleriano se e somente se G é conexo e se todos os seus vértices e V têm grau par. Na prova construtiva dada, das condições de suficiência sugere-se o seguinte algoritmo recursivo para encontrar um passeio euleriano em qualquer multigrafo (V, E) satisfazendo os item (a) e (b) do teorema referido, em tempo $O(|E|)$. Devido a Euler, este recebe o nome de Procedimento de Euler, o

qual aceita como entrada o conjunto de vértices $V = \{1, 2, \ldots, n\}$ de um grafo conexo $G = (V, E)$ e retorna uma sequência de arestas que pertencem a um passeio euleriano da componente conexa de G contendo v_1, chamado o passeio Euler(v_1) (o passeio retornado começa com o vértice v_1).

Procedimento Euler(v_1)(retorna um passeio euleriano)

Início

Se v_1 não tem arestas **então retornar** $[v_1]$ (o passeio é vazio)

caso contrário

Inicio

começando de v_1 criar um passeio de G, nunca visitando a mesma aresta duas vezes, até que v_1 seja alcançado outra vez;

Seja $[v_1, v_2, \ldots, v_n, v_1]$ este passeio;

remova $(v_1, v_2), \ldots, (v_n, v_1)$ de G;

retornar $[Euler(v_1), Euler(v_2), \ldots, Euler(v_n), v_1]$

fim

fim

Como exemplo, considere-se o multigrafo G da figura 4.2 que é euleriano desde que existe um passeio fechado $[v_1, v_3, v_7, v_8, v_9, v_5, v_4, v_2, v_6, v_5, v_8, v_6, v_1, v_7, v_3, v_2, v_1]$ que atravessa todos os vértices ao menos uma vez, e cada aresta exatamente uma vez.

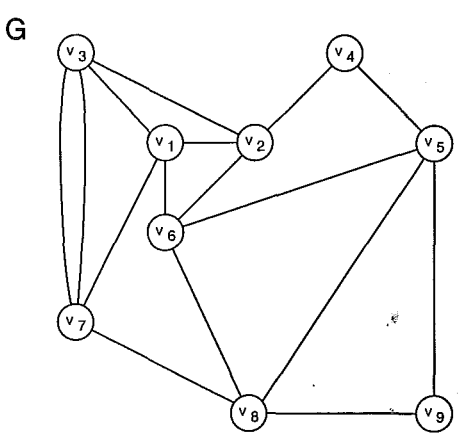


Figura 4.2: Multigrafo G

Neste multigrafo aplicamos o Procedimento Euler. Começamos com v_1 e criamos um passeio, que chamaremos de Euler(v_1), pode-se escolher o passeio [v_1, v_3, v_2, v_4, v_5, v_6, v_8, v_7, v_1] (este passeio é mostrado na figura 4.3 em linhas pontilhadas), e ao

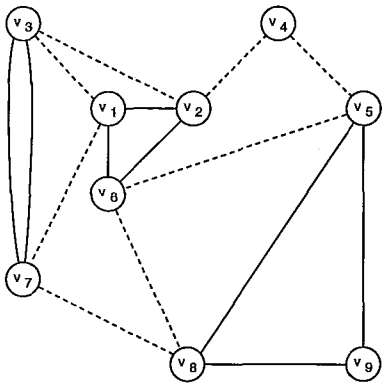


Figura 4.3: As linhas pontilhadas retornam o passeio Euler(v_1) começando por v_1

remover as arestas deste passeio o grafo resultante é o mostrado na figura 4.4. Seguindo com o procedimento, novamente pode-se criar Euler(v_1), que retorna o passeio $[v_1, v_2, v_6, v_1]$ como mostra a figura 4.4(b). Euler(v_3) retorna $[v_3, v_7, v_3]$. Euler(v_2) retorna $[v_2]$, Euler(v_4) retorna $[v_4]$, Euler(v_5) retorna $[v_5, v_9, v_8, v_5]$, Euler(v_6) retorna $[v_6]$, Euler(v_8) retorna $[v_8]$ e Euler(v_7) retorna $[v_7]$. Portanto o passeio Euleriano resultante do grafo mostrado na figura 4.2 é

$[v_1, v_2, v_6, v_1, v_3, v_7, v_3, v_2, v_4, v_5, v_9, v_8, v_5, v_6, v_8, v_7, v_1]$.

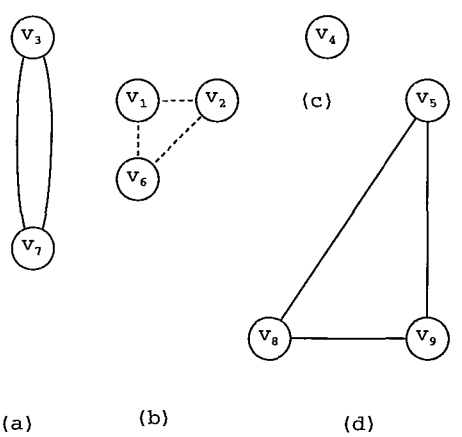


Figura 4.4: Grafo resultante da remoção de Euler(v_1). (a)Euler(v_3). (b)Euler(v_1). (c)Euler(v_4). (d)Euler(v_5)

A seguir será provado que sendo $G = (V, E)$ um multigrafo Euleriano é possivel

encontrar um tour em tempo $O(|E|)$. O problema de encontrar um tour mínimo é *equivalente* a encontrar um grafo gerador euleriano de custo mínimo. Para isto, será definida $[d_{ij}]$ como sendo uma matriz de distâncias $n \times n$ satisfazendo a desigualdade triangular, e dizemos que um *grafo gerador euleriano* é um multigrafo euleriano $G = (V, E)$ com $|V| = \{1, 2, \ldots, n\}$, onde o custo de G, é dado pela somatória das distâncias das arestas existentes em G, i.e., o custo de G é dado por:

$$c(G) = \sum_{(i,j) \in E} d_{ij} \tag{4.1}$$

Teorema 4.2. *Se $G = (V, E)$ é um grafo gerador euleriano, então podemos encontrar um tour τ de V com $c(\tau) \leq c(G)$ em tempo $O(|E|)$.*

Prova: Por hipótese, G tem um passeio euleriano. Considerando w este passeio euleriano, então w visita todos os vértices ao menos uma vez e cada aresta uma só vez, logo pode-se escrever $w = [\alpha_0 i_1 \alpha_1 i_2, \ldots, i_n \alpha_n]$ onde $\tau = (i_1, \ldots, i_n)$ é um tour, e $\alpha_0, \alpha_1, \ldots, \alpha_n$ são sequências (possivelmente vazias) de vértices de $\{1, \ldots, n\}$ (pode-se dizer que τ esta imerso em G), como se mostra no esquema da figura 4.5.

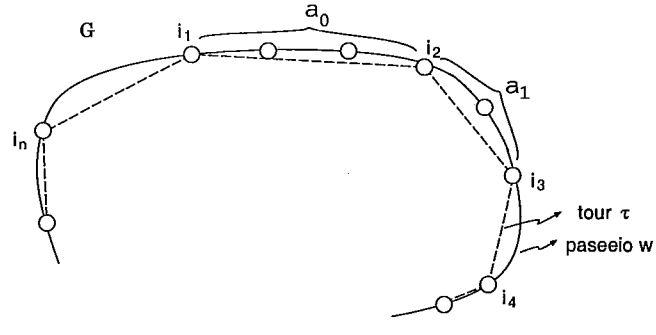


Figura 4.5: Grafo G

Agora a desigualdade triangular implica que

$d_{ik} \leq d_{ij_1} + d_{j_1 j_2} + \ldots + d_{j_{m-1} j_m} + d_{j_m k}$ para qualquer $m \geq 1$.

Por outro lado, o comprimento total de w que é exatamente $c(G)$ segundo a equação (4.1) não pode ser menor que $d_{i_1 i_2} + d_{i_2 i_3} + \ldots + d_{i_n i_1} = c(\tau)$. Portanto $c(G) \geq c(\tau)$. ∎

A seguir, será desenvolvido o algoritmo da árvore para obter um tour ou ciclo

hamiltoniano do problema do caixeiro viajante. Para isto suponha que seja dado um grafo euleriano, sendo o seus vértices, as cidades para o PCV, então pode-se usar o passeio euleriano deste grafo para obter um tour do caixeiro viajante tomando atalhos. A operação de *tomar atalho* tem sua interpretação imediata em grafos euclideanos, pois a desigualdade triangular nos assegura que a menor distância entre dois pontos é a linha reta.

Observação:

Observe que ao remover qualquer aresta de um tour obtém-se uma árvore geradora. Portanto, a árvore geradora mínima de um grafo constitui um limite inferior para o tour ótimo. É possivel então para problemas euclideanos, obter algoritmos aproximativos por uma transformação conveniente da árvore geradora mínima em um tour.

4.2 Algoritmo da Árvore

O algoritmo da árvore [9] e [31] constrói um tour a partir da árvore geradora mínima de G (em [1] encontra-se uma grande variedade de algoritmos para construir uma árvore geradora mínima). Este algoritmo sugere que se quer achar o melhor de todos os tour do caixeiro viajante, gerando um grafo euleriano que conecta todas as cidades e caracteriza-se por ter razão de desempenho no pior caso igual a 2, i.e., que o custo obtido pelo algoritmo da árvore é duas vezes o custo do tour ótimo.

O algoritmo pode-se ser esquematizado em 3 passos:

Passo 1. Obter a árvore geradora mínima

Passo 2. Criação do multigrafo euleriano

Passo 3. Obter um passeio euleriano e um tour imerso.

4.2.1 Algoritmo

Sejam dados um grafo completo $G = (V, E)$, um conjunto $V = \{1, 2, \ldots, n\}$ de vértices e as distâncias d_{ij} para todo $i, j \in V$ entre pares destes vértices[1].

Algoritmo da árvore $G = (V, E)$ [Tour obtido da árvore geradora mínima]

Início (Passo 1: Obtenção da arvore geradora mínima T)

Obter a árvore geradora mínima T do grafo G

$Y := \emptyset$; $H := \emptyset$;

Enquanto $T \neq \emptyset$ fazer

Início (Passo 2: Criação do multigrafo G')

Escolher $(v_i, v_j) \in T$;

$Y := Y \bigcup \{(v_i, v_j), (v_j, v_i)\}$;

$T := T \backslash \{(v_i, v_j)\}$;

fim;

(Passo 3: Obtenção do tour H imerso num passeio euleriano W)

Obter um passeio euleriano $W = \{v_1, v_2, \ldots, v_k, v_1\}$ de $G'(V, Y)$;

Enquanto $W \neq \emptyset$ fazer

Início

Escolher seqüencialmente $v_i \in W$;

Se $v_i \notin H$ então $H := H \cup \{v_i\}$;

$W := W \backslash \{v_i\}$;

fim;

$H := H \cup \{v_1\}$;(o tour H é construido partindo de W)

Escrever $\{H\}$;

fim.

O algoritmo da árvore é da ordem $O(n^2)$

- Achar a árvore geradora mínima pode ser feito em tempo $O(n^2)$.

- Achar um multigrafo euleriano onde há um passeio euleriano pode ser obtido por

[1] "distância" pode ser alternativamente custo, tempo ou alguma outra quantidade.

um procedimento de ordem $O(n)$.

- A operação realizada para obter um tour partindo de um passeio euleriano tomando atalhos é de ordem $O(n)$.

4.2.2 Ilustração do algoritmo da árvore

Ilustra-se mediante um exemplo como trabalha o *"Algoritmo da Árvore"* . Seja $G = (V, E)$ o grafo completo de $|V| = n = 9$ vértices representados na figura 4.6

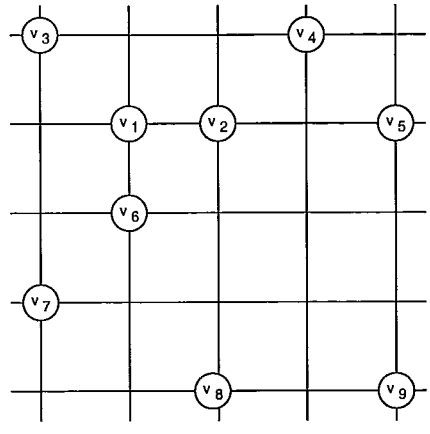


Figura 4.6: Instância

com a matriz de distâncias d_{ij} dados na figura 4.7 a seguir. E é usada a distância euclideana.

$$d_{ij} = \begin{bmatrix} 0 & 100 & 141 & 224 & 300 & 100 & 224 & 316 & 424 \\ 100 & 0 & 224 & 141 & 200 & 141 & 283 & 300 & 412 \\ 141 & 224 & 0 & 300 & 412 & 224 & 300 & 447 & 566 \\ 224 & 141 & 300 & 0 & 141 & 283 & 424 & 412 & 412 \\ 300 & 200 & 412 & 141 & 0 & 316 & 447 & 361 & 300 \\ 100 & 141 & 224 & 283 & 316 & 0 & 141 & 224 & 361 \\ 224 & 283 & 300 & 424 & 447 & 141 & 0 & 224 & 412 \\ 316 & 300 & 447 & 412 & 360 & 224 & 224 & 0 & 200 \\ 424 & 412 & 566 & 412 & 300 & 361 & 412 & 200 & 0 \end{bmatrix}$$

Figura 4.7: Matriz de distâncias d_{ij}

Se inicializa a construção da árvore usando o algoritmo de Prim [12]:

O algoritmo de Prim [33] começa com um vértice inicial v e consiste em efetuar repetidamente o seguinte passo $n-1$ vezes.

Passo (Prim): Seja T uma árvore que contém v. Escolhe-se uma aresta de custo mínimo incidente a T e adiciona-se esta aresta a T.

Seguindo a execução do algoritmo de Prim sobre o grafo completo G, o vértice v_1 é o vértice inicial e depois de 8 passos é completada a árvore geradora mínima T (veja figura 4.8) e seu custo é igual a 1188.

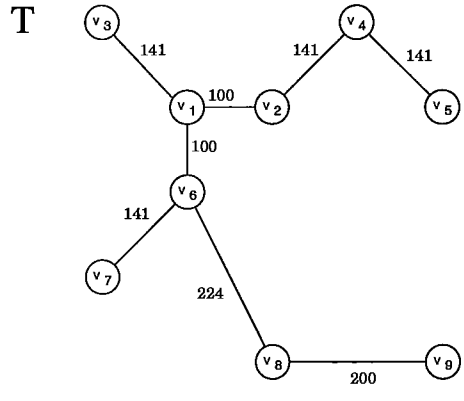


Figura 4.8: Árvore T

A seguir duplica-se cada aresta de T obtendo assím um multigrafo G' e por um procedimento de Euler começando com v_1, obtém-se um passeo euleriano que visita os vértices no ordem $v_1, v_2, v_4, v_5, v_4, v_2, v_1, v_6, v_8, v_9, v_8, v_6, v_7, v_6, v_1, v_3, v_1$, a este passeio chama-se de W (veja figura 4.9).

$W = [v_1, v_2, v_4, v_5, v_4, v_2, v_1, v_6, v_8, v_9, v_8, v_6, v_7, v_6, v_1, v_3, v_1]$ Seu custo total é de fato duas vezes o custo da árvore T, $c(W) = 2376$ como mostrado na figura 4.9.

Note bem que o passeio W tem mais de 9 vértices, pois visita algum vértice mais de uma vez, então pode-se aplicar a operação de tomar atalhos para obter um tour, por exemplo: o caminho $[v_2, v_1, v_6]$ pode-se ser substituído por $[v_2, v_6]$ e o caminho $[v_7, v_6, v_1, v_3]$ pode-se ser substituído por $[v_7, v_3]$ já que o grafo menor é um grafo euleriano visitando todos os vértices uma só vez, além do mais a desigualdade triangular garante que o custo total das arestas não é maior que antes (figura 4.10).

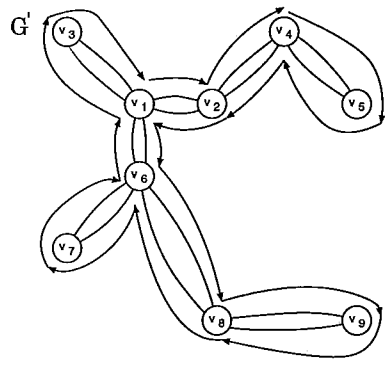


Figura 4.9: W:Multigrafo G' criado de T

E finalmente, um tour H obtido pelo algoritmo da árvore e mostrado a seguir na

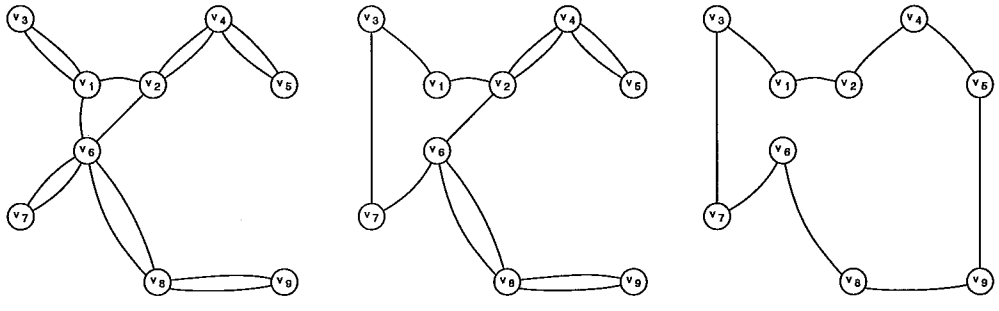


Figura 4.10: Tomando atalhos

figura 4.11 para o conjunto dado de vértices.

$H = [v_1, v_2, v_4, v_5, v_9, v_8, v_6, v_7, v_3, v_1]$

E ou custo total de H é aproximadamente 1688.

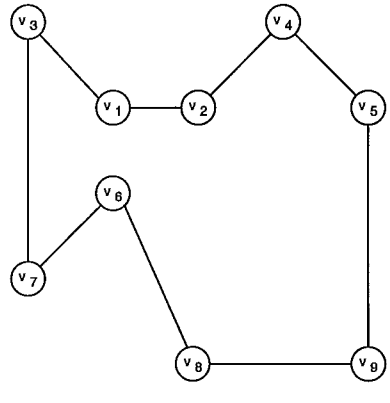


Figura 4.11: Tour H obtido pelo algoritmo da árvore

4.2.3 Caracterização do algoritmo da árvore

O seguinte teorema mostra o comportamento do pior caso do algoritmo da Árvore.

Teorema 4.3. *O algoritmo da Árvore é um algoritmo 1-aproximativo para o problema euclideano do caixeiro viajante.*

Prova: Assume-se primeiro que G é euleriano, pois só assim é possível encontrar um passeio euleriano. Como G contém uma árvore geradora T, ele é conexo. Além do mais, todos os vértices de G tem grau par (porque eles são duas vezes os graus correspondentes de T). A árvore geradora mínima T é um limite inferior para um tour ótimo H^*, porque qualquer tour pode ser tranformado em uma árvore simplesmente removendo uma aresta, portanto:

$$c(T) \leq c(H^*) \tag{4.2}$$

onde $c(T)$ é o custo da árvore geradora mínima. Por outro lado o passeio euleriano completo de T chamado de W lista todos os vértices quando eles são visitados, que pode ser mais de uma vez mas percorre toda aresta de T exatamente duas vezes, então temos

$$c(W) = 2c(T) \tag{4.3}$$

das equações (4.2) e (4.3) e dado que o tour é obtido por operações de atalho implica que

$$c(W) \leq 2c(H^*) \tag{4.4}$$

W geralmente não é um tour, pois alguns vértices são visitados mais de uma vez. Mas, pela desigualdade triangular pode-se remover os vértices intermediários para obter um caminho direto, não aumentando o custo. Aplicando repetidamente esta operação podemos remover de W os vértices já visitados obtendo assim um passeio de vértices distintos finalizando com o primeiro visitado para obter um tour H. Então temos que:

$$c(H) \leq c(W) \tag{4.5}$$

combinando as desigualdades (4.4) e (4.5) temos

$$c(H) \leq c(W) \leq 2c(H^*)$$

$$\Longrightarrow c(H) \leq 2c(H^*)$$

ou ainda

$$\frac{c(H)}{c(H^*)} \leq 2 \Leftrightarrow c(H) \leq c(H^*) + c(H^*) \Leftrightarrow \frac{c(H) - c(H^*)}{c(H^*)} \leq 1$$

Isto quer dizer que algoritmo da árvore é um algoritmo $1 -$ *aproximativo* e tem razão de desempenho no pior caso igual a 2. ∎

Observação:

O algoritmo pode ter um mal comportamento. O exemplo a seguir mostra uma instância euclideana que alcança 100% do limite superior.

Exemplo do pior caso:

Seja dada uma instância euclideana. A árvore geradora mínima T é mostrada na figura seguinte:

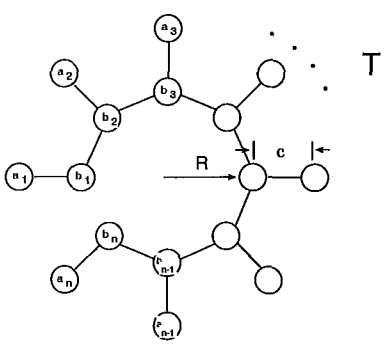


Figura 4.12: Árvore geradora mínima T

Agora criamos um multigrafo usando duas vezes cada aresta de T, como se mostra na figura 4.13:

Seguidamente aplicando o algoritmo da árvore, achamos um passeio euleriano do multigrafo, podemos obter um tour H (veja figura 4.15(a)) e fazendo um cálculo com ajuda da figura 4.14 obtemos seu custo:

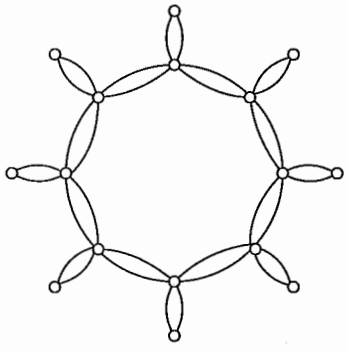


Figura 4.13: Multigrafo criado

$$c(H) = (n-1)2Rsen(\tfrac{\pi}{n}) + (n-1)2(R+c)sen(\tfrac{\pi}{n}) + 2c$$
$$= 2(n-1)sen(\tfrac{\pi}{n})(R+R+c) + 2c$$
$$= 2(n-1)(2R+c)sen(\tfrac{\pi}{n}) + 2c$$

E obtém-se também o tour ótimo que denotamos por H^*, mostrado na figura 4.15(b) a seguir, e seu custo é:

$$c(H^*) = (\tfrac{n}{2})2R\text{sen}(\tfrac{\pi}{n}) + (\tfrac{n}{2})2(R+c)sen(\tfrac{\pi}{n}) + cn$$
$$= n(2R+c)\text{sen}(\tfrac{\pi}{n}) + nc$$

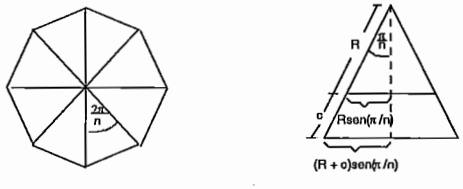


Figura 4.14: Esquema para a construção dos custos de H e H^*

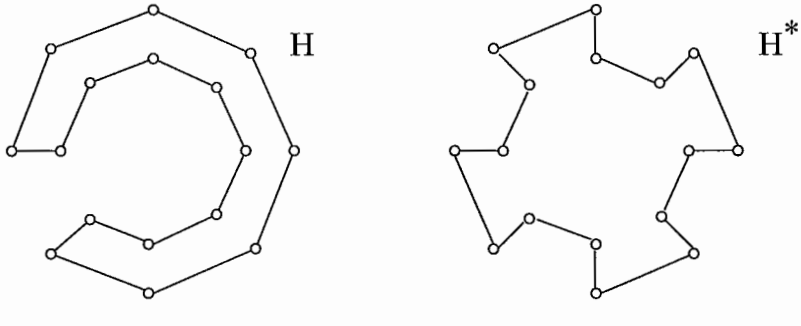


Figura 4.15: (a)Tour obtido pelo algoritmo da árvore. (b)Tour ótimo.

Então, ressumindo temos:

$c(H) = 2(n-1)(2R+c)sen(\frac{\pi}{n}) + 2c$, e

$c(H^*) = n(2R+c)sen(\frac{\pi}{n}) + nc$

Agora se $R = 1$, e $c = \frac{1}{n^2}$, sendo n arbitrariamente grande, vemos que:

$$\lim_{n\to\infty} c(H) = 4\pi$$

enquanto que

$$\lim_{n\to\infty} c(H^*) = 2\pi$$

Consequentemente o erro pode ser feito arbitrariamente perto de 100%.

O algoritmo da árvore foi conhecido por muitos anos como o algoritmo aproximado para o *PCV* com a melhor razão de desempenho no pior caso. Porém, para melhorar a razão de desempenho, Christofides[11] mostrou uma outra maneira de converter a árvore geradora mínima em um grafo euleriano, que envolve o conceito de um emparelhamento mínimo. Na seção seguinte mostraremos o algoritmo de Christofides que garante um tour de comprimento no máximo de 3/2 vezes o comprimento ótimo.

4.3 Algoritmo de Christofides

Este algoritmo foi desenvolvido por Christofides[11] em 1976, para obter um tour para o *PCV* euclideano. Encontra um tour aproximado de comprimento que não excede em mais de 50% o comprimento do tour ótimo; mas especificamente, o custo do tour aproximado é no máximo o 3/2 vezes do tour ótimo. Esta razão de desempenho caracteriza ao algoritmo de Christofides.

Um *emparelhamento* em G é um conjunto de arestas de G ($M \subseteq E$) tal que seus elementos não são adjacentes, i.e., duas arestas de M não têm vértice comum.

O algoritmo de Christofides pode ser esquematizado em três passos:

Passo 1. Obter a árvore geradora mínima

Passo 2. Obter emparelhamento mínimo com os vértices de graus ímpar da árvore geradora mínima, e o multigrafo construido terá tanto arestas da árvore geradora mínima quanto arestas do emparelhamento

Passo 3. Obter um passeio euleriano e um tour imerso

4.3.1 Algoritmo

O algoritmo é descrito a seguir: Seja dado um grafo $G = (V, E)$ onde o $V = \{1, 2, \ldots, n\}$ e d_{ij} a matriz distância para cada par de vértices.

Algoritmo Christofides($G = (V, E)$)[Tour da árvore geradora mínima]

Início (Passo 1: Obtenção da árvore geradora mínima T

Obter a árvore geradora mínima T de G;

(Passo 2: Obtenção do emparelhamento mínimo M)

Definir $G_0 = (V_0, M_0)$, onde $M_0 = \{(i, j) \in V / i, j \in V_0\}$ sendo V_0 o conjunto de vértices de T com grau ímpar;

Obter o emparelhamento perfeito mínimo M em G_0;

(Passo 3: Obtenção do tour imerso H dentro do passeio euleriano W)

Obter um passeio euleriano $W = \{v_1, v_2, \ldots, v_k, v_1\}$ a partir de

$G' = (V, T \cup M)$;

$H := \emptyset$;

Enquanto $W \neq \emptyset$ fazer

Início

Escolher seqüencialmente $v_i \in W$;

Se $v_i \notin H$ então $H = H \cup \{v_i\}$;

$W := W \backslash \{v_i\}$;

fim

$H := H \cup \{v_1\}$;

Escrever $\{H\}$;

Fim.

O algoritmo de Christofides é da ordem $O(n^3)$

- Achar a árvore geradora mínima pode ser feito em tempo $O(n^2)$.
- Achar um emparelhamento perfeito mínimo em um grafo completo (com um número par de vértices) pode ser feito em tempo $O(n^3)$.
- Achar um tour partindo de um passeio euleriano tomando atalhos pode ser feito em tempo linear.

4.3.2 Ilustração do Algoritmo de Christofides

Aplicamos o algoritmo de Christofides ao grafo completo G de 9 vértices (figura 4.6) satisfazendo a matriz de distâncias da figura 4.7.

Obtenção de uma árvore geradora mínima T de $G = (V, E)$ encontrada pelo algoritmo de Prim, onde $V = \{v_1, v_2, \ldots, v_9\}$ é o conjunto de vértices. A árvore é mostrada na figura 4.16

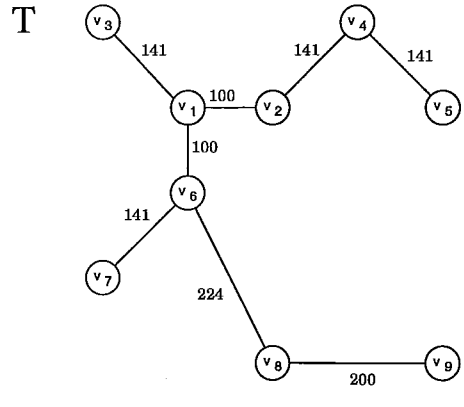


Figura 4.16: Árvore geradora mínima T

Obtenção do emparelhamento mínimo M.

Define-se $G_0 = (V_0, M_0)$ onde $V_0 = \{v_1, v_3, v_5, v_6, v_8, v_9\}$ é o conjunto de vértices de grau ímpar e $M_0 = \{(i, j) \in V / i, j \in V_0\}$, então um emparelhamento perfeito mínimo é (veja figura 4.17)

$$M = \{[v_3, v_7], [v_1, v_6], [v_5, v_9]\}$$

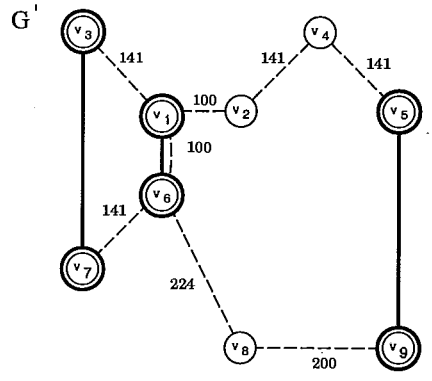


Figura 4.17: Grafo G'

Obtenção do passeio euleriano e um tour imerso.

Tendo o grafo $G' = (V, TUM)$. Um passeio euleriano W pode ser formado pela seguinte sequência de vértices

$$W = [v_6, v_1, v_2, v_4, v_5, v_9, v_8, v_6, v_7, v_3, v_1, v_6]$$

Note que o caminho $[v_6, v_1, v_2]$ pode ser substituído pelo caminho direto $[v_6, v_2]$, e o caminho $[v_8, v_6, v_7]$ pode ser substituído pelo caminho direto $[v_8, v_7]$. Veja figura 4.18(a) (pela desigualdade triangular o custo do caminho mais direto é menor), assím um tour ótimo construido pelo algoritmo de Christofides é

$$H = [v_6, v_2, v_4, v_5, v_9, v_8, v_7, v_3, v_1, v_6]$$

de custo $c(H) = 1688$ é mostrado na figura 4.18(b):

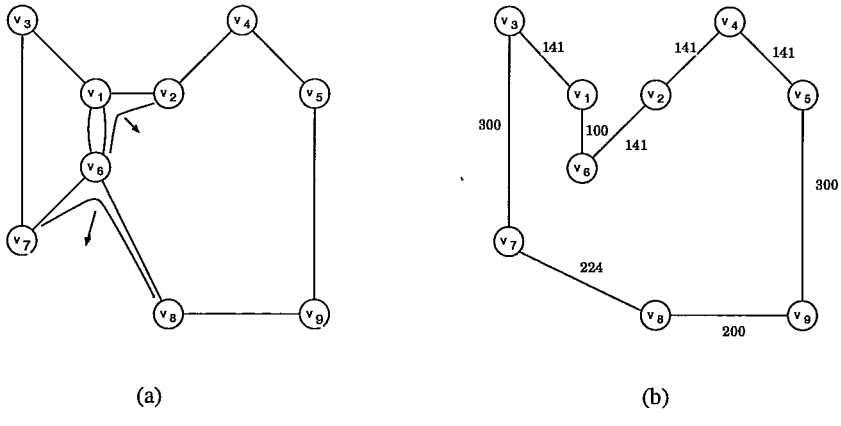


Figura 4.18: (a)Grafo G'. (b)Tour obtido pelo algoritmo de Christofides

4.3.3 Caracterização do algoritmo de Christofides

O seguinte teorema mostra o comportamento do pior caso do algoritmo de Christofides na construção do tour.

Teorema 4.4. *O algoritmo de Christofides é um algoritmo 1/2-aproximativo para* ΔPCV.

Prova: O grafo G construído no passo 2 é euleriano pois se um vértice tem grau par em T, enão esse vértice também tera grau par em G, pois o grafo G será formado pelas arestas de T incluindo as arestas do emparelhamento M , ou seja estaremos adicionando os graus unicamente aos vértices de grau ímpar, i.e., se um vértice tiver um grau ímpar em T, este terá uma aresta a mais incidente a ele proveniente do emparelhamento M. Além do mais, como G contém uma árvore geradora T como um subgrafo, ele é conexo.

Como em um grafo euleriano todo vértice tem grau par (Teorema 2.2), nos preocupamos com os vértices que possuem grau ímpar: o número de vértices de grau ímpar em T será sempre par, pois a soma dos graus de todos os vértices deve ser par (cada aresta é contada duas vezes).

Assim, uma forma de obter um grafo euleriano é acrescentar a T as arestas do emparelhamento mínimo M, calculado sob o grafo completo induzido pelos vértices de grau ímpar de T. Digamos então que M seja o emparelhamento perfeito mínimo.

Agora $G = (V, T \cup M)$ é um grafo euleriano de menor comprimento dentre todos os que contêm a árvore geradora T. De G constrói-se um ciclo euleriano e em seguida aplica-se a desigualdade triangular (que nos asegura que a menor distância entre dois pontos é uma linha reta (em grafos euclideanos)).

Para provar a razão de desempenho 1/2, lembremos que o grafo G consiste de T e M, ie, $G = (V, T \cup M)$ onde V é o número de vértices tanto de T e M, e as arestas de G estão formados pelas arestas de T mais as arestas de M, daí o custo de G é

$$c(G) = c(T) + c(M)$$

como G é euleriano então podemos encontrar um tour resultante H de V satisfazendo

$$c(H) \leq c(G),$$

$$\Longrightarrow c(H) \leq c(T) + c(M) \tag{4.6}$$

o custo da árvore geradora mínima T é um limite inferior para o valor ótimo $c(H^*)$

$$c(T) \leq c(H^*) \tag{4.7}$$

onde H^* é um tour mínimo. Além do mais, seja $\{i_1, i_2, \ldots, i_{2m}\}$ o conjunto de vértices de T com grau ímpar, no ordem que eles aparecem em H^*. Em outras palavras $H^* = [\alpha_0 i_1 \alpha_1 i_2 \ldots \alpha_{2m-1} i_{2m} \alpha_{2m}]$, onde as α's são (possivelmente vazias) sequências de vértices de $\{1, 2, \ldots, n\}$. Consideremos dois emparelhamentos de vértices de grau ímpar $M_1 = \{(i_1, i_2), (i_3, i_4), \ldots, (i_{2m-1}, i_{2m})\}$ e
$M_2 = \{(i_2, i_3), (i_4, i_5), \ldots, (i_{2m}, i_1)\}$.

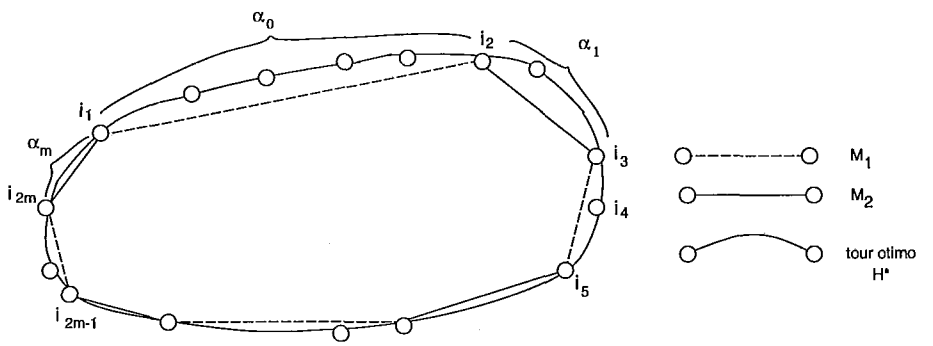


Figura 4.19: Os dois emparelhamentos induzidos por um tour ótimo

Os dois emparelhamentos induzem um tour ótimo. Os vértices nos emparelhamentos são os vértices de grau ímpar em uma árvore geradora mínima. Pela desigualdade triangular (ver figura 4.19), cada aresta no emparelhamento não é tão longa quanto que os correspondentes segmentos do tour ótimo, e assim

$$c(M_1) + c(M_2) \leq c(H^*)$$

Portanto um dos M_1 e M_2 poderia ter custo menor ou igual a $\frac{c(H^*)}{2}$.

Assim o custo de M poderia também ser no máximo $\frac{c(H^*)}{2}$, i.e.,

$$2c(M) \leq c(H^*)$$

$$c(M) \leq \frac{1}{2}c(H^*) \tag{4.8}$$

substituindo (4.7) e (4.8) em (4.6), conseguimos

$$c(H) = c(T) + c(M) \leq c(H^*) + \frac{1}{2}c(H^*) = \frac{3}{2}c(H^*)$$
$$\iff c(H) - c(H^*) \leq \frac{1}{2}c(H^*)$$
$$\iff \frac{c(H) - c(H^*)}{c(H^*)} \leq \frac{1}{2}.$$

Ou seja o algoritmo de Christofides é um algoritmo $1/2 - aproximativo$ e tem razão de desempenho no pior caso igual a 3/2. ■

Como o algoritmo da árvore, o algoritmo de Christofides pode alcançar assintoticamente o seu pior caso limite. O seguinte exemplo da figura 4.20 ilustra devido a Cornuéjols e Nemhauser[13], ilustra este caso.

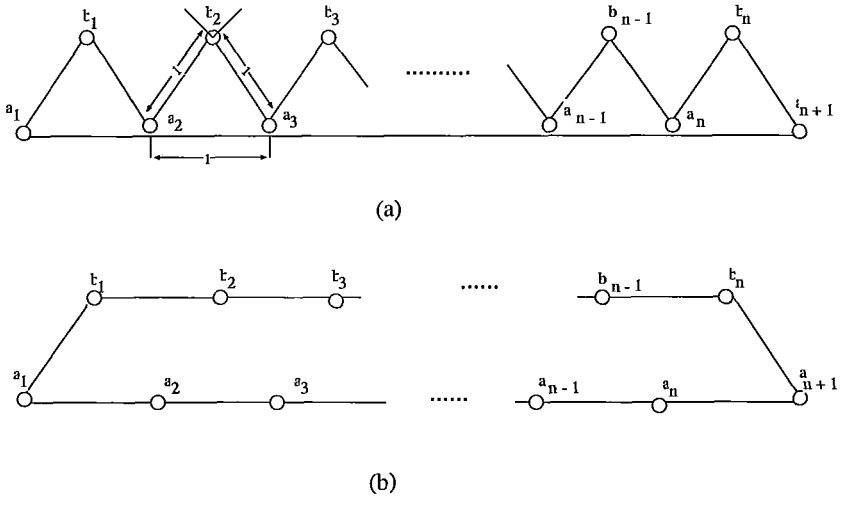


Figura 4.20: Exemplo do pior caso do algoritmo de Christofides. (a)Árvore geradora mínima de entrada. (b)O tour encontrado pelo algoritmo de Christofides pode ter comprimento 3/2 vezes do ótimo.

Existem somente 2 vértices de grau ímpar, e por consiguinte o emparelhamento ótimo é uma simples aresta. O grafo euleriano resultante é um tour. O tour assim construído tem comprimento total $3n$, enquanto que o tour mínimo (ver figu-

ra 4.20(b)) tem comprimento $2n + 1$. Assim o erro pode chegar arbitrariamente perto de $1/2$.

Capítulo 5

Esquema de Aproximação em Tempo Polinomial

Neste capítulo, apresenta-se um esquema de aproximação em tempo polinomial para o PCV euclideano no plano recentemente desenvolvido por Arora [2]. A idéia principal no algoritmo é realizar um particionamento geométrico (recursivo) de uma instância em instâncias menores. O conjunto de cidades é subdividido em grupos pequenos. Observamos que o melhor algoritmo conhecido até então para o problema encontrava uma 3/2-*aproximação* (veja algoritmo de Christofides seção 4.3).

Num PCV métrico os vértices estão sobre o espaço métrico (i.e., as distâncias satisfazem a desigualdade triangular). Num PCV Euclideano os vértices encontram-se no espaço $\Re^2$ e a distância é definida usando a norma ℓ_2. Note que o PCV Euclideano é um subcaso de PCV métrico. Também, recentemente Arora, Lund, Motwani, Sudan e Szegedy [5] mostraram que se $P \neq NP$, então o PCV métrico e muitos outros problemas não têm um esquema de aproximação em tempo polinomial. Esse trabalho se baseou sobre a teoria de *MAX-SNP-completitude* desenvolvida por Papadimitriou e Yannakakis [32].

Um esquema de aproximação em tempo polinomial ($EATP$) é um algoritmo em tempo polinomial ou uma familia de tais algoritmos de modo que, para cada $\varepsilon > 0$ fixo, podemos aproximar o problema dentro do fator $(1 + \varepsilon)$. (Observa-se que o

tempo de execução do algoritmo poderia depender de ε).

Até agora são poucos os problemas conhecidos que admitem $EATP$.

Nas seções seguintes mostra-se que o PCV Euclideano admite um $EATP$.

Dados n vértices no plano o $EATP$ encontra para cada $\varepsilon > 0$, uma $(1+\varepsilon)$-aproximação para o tour ótimo do caixeiro viajante em tempo $n^{O(1/\varepsilon)}$.

O $EATP$ é projetado de maneira que cada instância do PCV Euclideano no plano tem um tour $(1+\varepsilon)$-aproximado com a seguinte estrutura: existe uma forma de particionar recursivamente o plano de maneira que "muito poucas" arestas do tour atravessam cada linha da partição (veja Teorema Estrutural). Um tour com tal estrutura pode ser encontrado usando programação dinâmica. A idéia de particionar uma instância do PCV em pequenas instâncias havia sido usada anteriormente, em [22] e programação dinâmica também foi usada anteriormente em um esquema de aproximação para o PCV em grafos planares [23].

5.1 O algoritmo para o PCV

A seguir, descreve-se o algoritmo para o PCV Euclideano no plano

5.1.1 O PCV Euclideano no plano

Como foi mencionado anteriormente, o algoritmo executa um particionamento geométrico (recursivo) da instância e posteriormente aplica programação dinâmica ao conjunto de instâncias produzidas.

No que segue, quando nos referirmos a um retângulo, estaremos supondo um retângulo alinhado com os eixos. Consideraremos o *tamanho* do retângulo como o comprimento de sua aresta mais longa. A *caixa limitadora* de um conjunto de vértices é o menor retângulo que os contém.

Modifica-se ligeiramente as instâncias do PCV de maneira que as distâncias dos vértices internos não sejam muito diferentes uma da outra.

Proposição 5.1. *Sejam n, $\varepsilon > 0$ tal que $n > 10/\epsilon$. Então o problema de calcular uma $(1+\epsilon)$-aproximação para o comprimento do tour ótimo em uma instância com n-vértices pode ser reduzido em tempo polinomial de n para o problema de calcular uma $(1 + 9\varepsilon/10)$-aproximação em uma instância na qual a menor distância dos vértices internos é de 1 unidade e o tamanho da caixa limitadora é no máximo $1.5n^2$.*

Prova: A redução implica em perturbar ligeiramente as instâncias dos n-vértices. Seja OPT denota o custo de um tour ótimo do caixeiro viajante para o conjunto de vértices dado. Seja T o custo de uma árvore geradora mínima, um limite inferior para o tour OPT, dado que qualquer tour simplesmente pode ser transformado a uma árvore simples removendo uma aresta, então

$$T \leq OPT$$

Portanto o tour ótimo do caixeiro viajante tem custo de pelo menos T e no máximo $2T$, i.e.,

$$T \leq OPT \leq 2T.$$

Assim o tamanho da caixa limitadora é no máximo T. Construa uma nova instância colocando um grade de granuralidade $T/2n^2$ no plano e movendo cada vértice para seu ponto-grade mais próximo,veja figura 5.1. Este procedimento pode causar que alguns vértices se fundam, i.e., pode-se ter um vértice que mapeia o mesmo ponto grade).

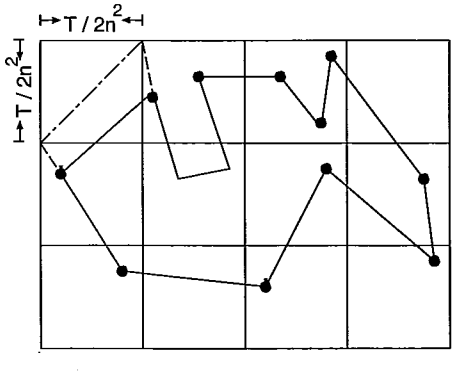


Figura 5.1: Instância nova sob a grade de granuralidade $T/2n^2$

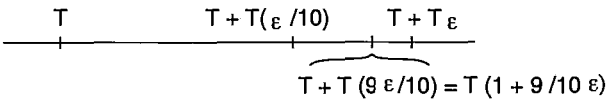


Figura 5.2: Esquema de redução em tempo polinomial de n

Como cada vértice pode ser movido em no máximo $\frac{T}{2n^2}$, e o tour do caixeiro viajante tem n arestas, então cada aresta pode ser aumentada ou diminuída no máximo

$$n \cdot (\frac{T}{2n^2} + \frac{T}{2n^2}) = \frac{2nT}{2n^2} = \frac{T}{n}$$

Logo, segue-se que a perturbação afeta o custo do tour ótimo no máximo em $\frac{T}{n}$.

Novamente, divida todas as distâncias em uma instância nova por $\frac{T}{2n^2}$, de maneira que a menor distância entre os vértices seja de pelo menos 1. Por outro lado, o tamanho da caixa limitadora da instância perturbada é no máximo de $T/(T/2n^2) < 2n^2$.

Agora como $\varepsilon/10 > 1/n$, basta calcular uma $(1+9\varepsilon/10)$-aproximação nesta nova instância (veja figura 5.2) em vez de uma $(1+\varepsilon)$-aproximação. ▪

5.2 Construção da árvore binária

Vamos definir agora uma partição recursiva do retângulo.

Uma *linha separadora* de um retângulo é um segmento de linha reta, paralelo a sua aresta menor e que particiona o retângulo em dois retângulos com pelo menos 1/3 da área total.

Exemplo:

Se a largura W de um retângulo é maior que sua altura H, então a linha separadora é qualquer linha vertical na parte central correspondente a $W/3$ do retângulo. Como ilustra a figura 5.3.

Em seguida definimos um *1/3 : 2/3 - ladrilhamento* para que possamos decompor

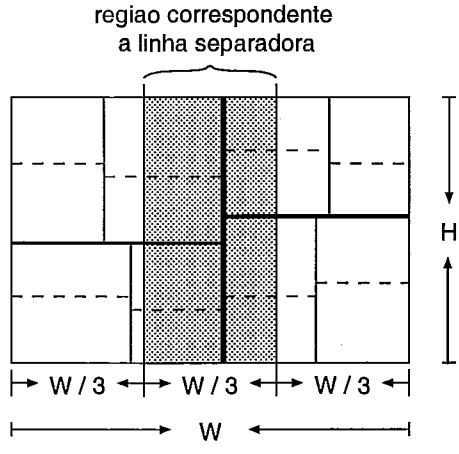


Figura 5.3: 1/3 : 2/3 - ladrilhamento

o problema em problemas menores.

Um *1/3 : 2/3 - ladrilhamento* de um retângulo R é uma árvore binária (i.e., uma hierarquia) de sub-retângulos de R. O retângulo R está na raiz. Se o tamanho de R é ≤ 1, então a construção da árvore binária termina em R. Caso contrário a raiz contém uma linha separadora para R, e tem duas subárvores que são 1/3 : 2/3 - ladrilhamentos dos dois retângulos na qual a linha separadora divide R. (Veja figura 5.4).

A profundidade do ladrilhamento é a profundidade máxima desta árvore.

Note que os retângulos que estão a profundidade d no ladrilhamento formam uma partição do retângulo raiz. O conjunto de todos os retângulos de profundidade d é um refinamento desta partição, obtido pela colocação de uma linha separadora em cada retângulo de profundidade $d-1$ de tamanho > 1. A área de qualquer retângulo de profundidade d é no máximo $(2/3)^d$ vezes da área total (a construção do ladrilhamento é mostrada na figura 5.4).

Como conseqüência damos a seguinte proposição.

Proposição 5.2. *Se um retângulo tem largura W e altura H, então cada um de seus 1/3 : 2/3 - ladrilhamentos tem profundidade de no máximo* $\log_{1.5} W + \log_{1.5} H + 2$.

Prova: Do esquema da figura 5.4 vemos que a construção do 1/3 : 2/3 −

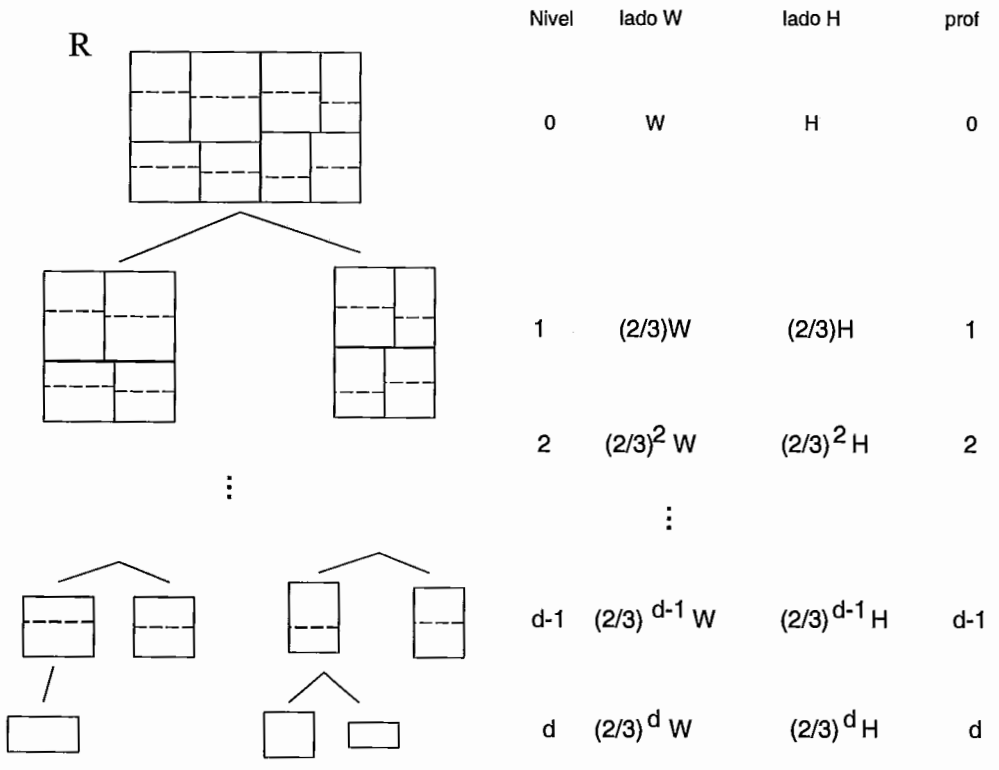


Figura 5.4: construção da árvore binária do ladrilhamento cuja raiz é o retângulo R.

ladrilhamento o retângulo de profundidade $d-1$ tem largura $(\frac{2}{3})^{d-1}W$, sendo este valor > 1, então que:

$$(\frac{2}{3})^{d-1}W > 1 \Rightarrow W > (\frac{3}{2})^{d-1}$$

e tomando $\log_{3/2}$ a ambos lados,tem-se

$$\log_{3/2} W > d-1$$

$$\Rightarrow \log_{3/2} W + 1 > d, \tag{5.1}$$

da mesma forma se o retângulo de profundidade $d-1$ tem altura $(\frac{2}{3})^{d-1}H$

$$\Rightarrow \log_{3/2} H + 1 > d, \tag{5.2}$$

logo das desigualdades (5.1) e (5.2) temos

$$\log W + 1 + \log H + 1 > 2d$$

$\Rightarrow d < \log W + \log H + 2$. ∎

Observamos que a profundidade do $1/3 : 2/3 -$ *ladrilhamentos* é calculada em

$O(\log n)$ vezes.

A seguir introduzimos o conceito de portal, que será muito utilizado no restante do desenvolvimento do algoritmo.

Um *portal* em um $1/3 : 2/3 - ladrilhamento$ é qualquer ponto que encontra-se na aresta de algum retângulo do ladrilhamento. Se m é qualquer inteiro positivo, então um conjunto de portais P é chamado $m - regular$ para o ladrilhamento se existem exatamente m portais equidistantes sobre a linha separadora de cada retângulo do ladrilhamento. (Supõe-se que os pontos extremos da linha separadora são também portais. Veja figura 5.5 Em outras palavras a linha separadora é particionada em exatamente $m - 1$ partes iguais por portais que estão sobre ela).

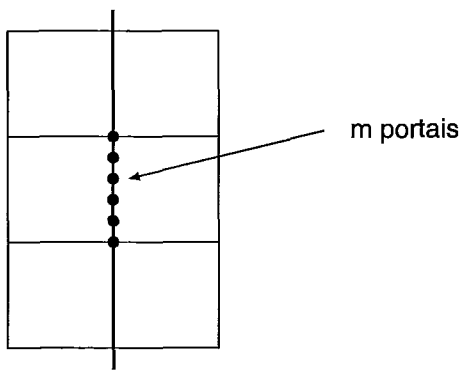


Figura 5.5: Ilustração de portais espaciados uniformemente

Agora mostraremos como as idéias dadas acima são usadas. Primeiro observamos que um tour ótimo do caixeiro viajante é sempre um polígono simples. Para simplificar, admitiremos tours com arestas "curvas". Estas arestas curvas aparecem da seguinte maneira. Nós introduzimos vértices adicionais ("steiner") no plano (estes vértices serão portais em algum $1/3 : 2/3 - ladrilhamento$) e exigimos que o tour visite estes vértices além dos vértices dados na entrada. Chamamos tal tour de um *caminho do caixeiro viajante.* É claro, que no fim do algoritmo, podemos mudar o caminho do caixeiro viajante em um tour poligonal, esticando as arestas curvas (i.e., removendo os pontos adicionais).

Para não perturbar demasiado o tour precisamos, limitar o número de vértices adicionais. Isto motiva a seguinte definição.

Seja $m \in Z^+$ e π um caminho do caixeiro viajante sobre algum conjunto de vértices. Seja S um $1/3 : 2/3 - ladrilhamento$ da caixa limitadora e P um conjunto m-regular de portais sobre este ladrilhamento. Então π é um caminho *m-leve* em relação a S se as condições a seguir se verificam:

(i) Em cada retângulo do ladrilhamento S, o caminho atravessa a linha separadora de tal retângulo no máximo em m vezes.

(ii) O caminho atravessa a linha separadora somente nos portais em P

(iii) O caminho não se auto-intercepta a si mesmo (exceto possivelmente num portal)

Observação:

Nós permitimos que o caminho se auto-intercepte a si mesmo passando por um portal k vezes, onde $k > 1$. Alternativamente, poderiamos insistir em tours que não se auto-interceptem, permitindo substituir o portal por uma matriz minúscula de k pontos ao redor dele e então o caminho pode usar um ponto diferente para cada cruzamento. Isto afeta o custo insignificantemente, assim os dois pontos de vista são equivalentes.

Proposição 5.3. *Se um caminho π do caixeiro viajante é m-leve em relação a um 1/3 : 2/3 - ladrilhamento da caixa limitadora, então o perímetro de cada retângulo em um ladrilhamento é atravessado pelo caminho no máximo $4m$ vezes.*

Prova: Cada lado do retângulo é uma linha separadora de algum retângulo que é seu antecessor na árvore ladrilhada. Portanto o caminho do caixeiro viajante atravessa esse lado no máximo m vezes, logo atravessa o perímetro no máximo $4m$ vezes. ■

O teorema seguinte, o qual será provado posteriormente, mostra que toda instância do PCV tem um caminho do caixeiro viajante quase-ótimo (e conseqüêntemente um tour do caixeiro viajante quase-ótimo) com uma estrutura muito simples.

Teorema 5.4. *(Teorema Estrutural) Existe um $c > 0$ tal que o seguinte é verdadeiro para cada $\varepsilon > 0$. Cada conjunto de vértices no plano tem uma $(1+\varepsilon)$ - aproximação do caminho π do caixeiro viajante, e um $1/3 : 2/3-$ladrilhamento associado à caixa limitadora tal que o tour é m-leve para este ladrilhamento, onde $m = c \log L/\varepsilon$, onde L é o tamanho da caixa limitadora.*

5.3 Descrição do EATP

Assumindo que o Teorema Estrutural é verdadeiro, vamos descrever o Esquema Aproximativo em tempo polinomial ($EATP$).

Como $\varepsilon > 0$ é arbitrário, usa-se primeiro a Proposição 5.1 e assumimos sem perda de generalidade que a menor distância entre os vértices internos é 1 e a caixa limitadora tem tamanho de no máximo $O(n^2)$. O Teorema Estrutural garante a existência de um $(1+\varepsilon)$ caminho ótimo π do caixeiro viajante e um ladrilhamento S em relação ao qual o caminho é m-leve, onde $m = O(\log n/\varepsilon)$.

De fato:

$$m = c \log L/\varepsilon = c \log \alpha n^2/\varepsilon = \underbrace{c \log \alpha/\varepsilon}_{\beta=cte} + c \log n^2/\varepsilon = \beta + 2c \log n/\varepsilon$$

Construção do ladrilhamento

Pela Proposição 5.2, a profundidade de qualquer ladrilhamento do tipo descrito é no máximo $O(logn)$.

Descreve-se a aplicação da técnica de programação dinâmica, proveniente da área de otimização combinatória [31] que acha ambos S e π com respeito ao ladrilhamento calculado em um tempo polinomial de n vezes $2^{O(m)} = n^{O(1/\varepsilon)}$

pois, $2^{O(m)} = 2^{\alpha m} = 2^{\alpha \log n/\varepsilon} = 2^{\frac{\alpha}{\varepsilon} \log n} = 2^{\log n^{\alpha/\varepsilon}} = n^{\frac{\alpha}{\varepsilon}}$,

logo polinomial de n vezes $n^{\frac{\alpha}{\varepsilon}} = n^{O(1/\varepsilon)}$.

A idéia da programação dinâmica permite que não se calcule duas vezes o mesmo

subproblema e que se utilize normalmente uma tabela de resultados que vai se completando a medida que se resolvem os subproblemas.

A programação dinâmica é um método *ascendente*, pois são resolvidos primeiro os subproblemas menores e portanto mais simples. Combinando suas soluções se obtêm as soluções de problemas sucessivamente maiores até chegar ao problema original.

A programação dinâmica usa a seguinte observação:

A caixa limitadora da instância certamente tem uma linha separadora que π atravessa somente $2k$ vezes, onde $2k \leq m$. Suponha que um "oráculo" anuncia esta linha separadora, os portais onde acontecem os cruzamentos, e a ordem em que estes cruzamentos acontecem no tour final. Então em cada lado da linha separadora estamos já separados com uma instância que chamaremos: o *problema do subtour*. Neste problema são dados um retângulo com alguns vértices nele, e um conjunto de k pares de portais em sua fronteira (um portal poderia aparecer em mais de um par). Se deseja achar caminhos disjuntos de vértices que conectam todos os pares de portais, de tal maneira que cada vértice que está dentro do retângulo, é visitado por um dos caminhos, e o comprimento total dos caminhos é minimizado.

Em geral, resolver o problema de subtour não é mais fácil que resolver o PCV em geral. Mas note que as duas instâncias têm a garantia de ter uma estrutura simples: cada uma tem uma linha separadora que é cruzada pelo subtour no máximo m vezes. Assuma que o "oráculo" continua ajudando e anuncia essas linhas separadoras e os portais sobre elas, onde os cruzamentos acontecem. Então pode-se quebrar cada uma das instâncias em outras duas instâncias do problema do subtour. Continuando deste modo obtém-se instâncias cada vez menores do problema do subtour, e paramos quando a instância é o suficientemente pequena para permitir uma solução que possa ser resolvida por força bruta, i.e., enumerando todas as possíveis soluções, calculando o custo de cada uma, e pegando a melhor.

Tudo isso assumindo a existência da ajuda de um "oráculo". Mas observe que m é tão pequeno que realmente não se precisa do oráculo. Ao invés, em cada passo

experimenta-se todas as linhas separadoras combinatorialmente distintas[2]. Existem $n-1$, e todas as possíveis ordens ($= m!$) nas quais o tour poderia atravessar esta linha separadora em $\leq m$ vezes. (O número de tais formas é de ordem $2^{O(m)}$ em $\Re^2$).

A tabela de procura do algoritmo terá polinomial de $n$vezes $2^{O(m)}$ entradas, e o algoritmo gastará um tempo da ordem de polinomial de $n$vezes $2^{O(m)}$ para construir cada entrada. Desde que $m = O(\log n/\epsilon)$, o tempo total de execução será $n^{O(1/\epsilon)}$.

Suponha R é um retângulo do ladrilhamento e o caminho do caixeiro viajante atravessa a fronteira de R em um total de $2k \leq 4m$ vezes. Seja $p_1, p_2, \ldots, p_{2k}$ sequência de portais onde esses cruzamentos ocorrem (os portais foram enumerados no ordem em que eles são atravessados pelo caminho do caixeiro viajante).

Então a parte do caminho do caixeiro viajante ótimo dentro do retângulo R é a sequência de k caminhos tais que:

(i) para $i = 1, \ldots, k$, o i-ésimo caminho conecta p_{2i-1} a p_{2i}

(ii) juntamente os caminhos visitam todos os vértices que estão dentro de R, e

(iii) a coleção de k caminhos atravessa cada aresta de cada retângulo no ladrilhamento em no máximo m vezes, e estes cruzamentos sempre acontecem em portais.

Desde que o caminho do caixeiro viajante é ótimo, a sequência de acima de k caminhos pode ser o conjunto de caminhos que tem o menor custo entre todos os caminhos que satisfazam (i), (ii), e (iii). Esta observação motiva a descrição da tabela. Uma entrada é indexada por uma tripla do seguinte tipo:

(a) Um retângulo não vazio dentro da caixa limitadora. Somente são considerados retângulos combinatorialmente distintos.

(b) Um multiconjunto que consiste de $2k$ portais no perímetro do retângulo, onde $2k \leq 4m$.

[2]duas linhas separadoras são combinatorialmente distintas se temos um vértice entre elas

(c) Uma partição dos $2k$ portais em k pares $(\{p_1, p_2\}, \{p_3, p_4\}, \ldots, \{p_{2k-1}, p_{2k}\})$. Esta partição representa o ordem no qual o caminho final percorrerá esses portais

Cada escolha em (a), (b) e (c) dá origem a uma instância do problema do subtour. A entrada na tabela de procura armazena o subtour de menor custo que é achado pelo algoritmo para esta instância.

Dado que (a), (b) e (c) são eventos independentes, e pelo principio multiplicativo [24] o tamanho da tabela de procura é no maximo o produto de:

$$(\#escolhas\ em(a)) \times (\#escolhas\ em(b)) \times (\#escolhas\ em(c))$$

 # escolhas em (a):

Para limitar superiormente isto, assume-se que toda linha separadora atravessa um vértice (encolhendo de maneira máxima os retângulos no $1/3 : 2/3$ - ladrilhamento). Conseqüêntemente o número de retângulos combinatorialmente distintos em (a) pode ser $\binom{n}{2} < \binom{n}{3} < \binom{n}{4}$, e no máximo é $\binom{n}{4}$, o número de 4-uplas de vértices.

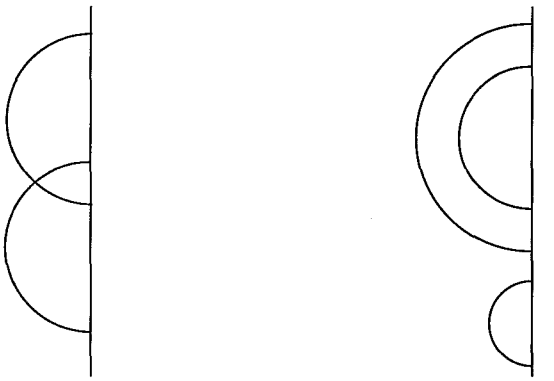
 # escolhas em (b):

O número de escolhas em (b) é $O(n^8) \times 2^{4m+2k}$. A razão para que seja $O(n^8)$ é que cada retângulo tem 4 lados, e cada um é parte da linha separadora de algum retângulo ancestral. Os m portais naquela linha separadora estão uniformemente espaçados, assim, eles são completamente determinados uma vez que se conheça a linha separadora. Mas o número de escolhas para uma linha separadora é no máximo o número de pares de vértices, o qual é $\binom{n}{2}$. Esta observação leva em consideração o fator $O((n^2)^4) = O(n^8)$. Além do mais, uma vez que tenhamos identificado o conjunto de $\leq 4m$ portais nos 4 lados, o número de modos de escolher um multiconjunto de $2k$ fora deles é no máximo 2^{4m+2k}.

 # escolhas em (c):

Finalmente, o número de possíveis pares em (c) é no máximo 2^{4m}. A razão é que em um caminho do viajante que não se auto-intersepte asi mesmo, um em-

parelhamento aceitável dos portais corresponde a um arranjo equilibrado de $2k$ parênteses.(Veja figura 5.6)



(a) (b)

Figura 5.6: Para tours que não se auto-interseptem asi mesmo, pares válidos de $2k$ portais correspondem a arranjos balanceados de k pares de parênteses. (a)mostra um par inválido e (b) mostra um par válido.

O número de tais arranjos é o k-ésimo *número de Catalan*, o qual é

$$\leq 2^{2k} \leq 2^{4m} = 2^{\log n^{4/\epsilon}} = n^{(4/\epsilon)} = n^{O(1/\epsilon)} \text{ (dado que 4 é constante)}$$

Conseqüêntemente pode-se limitar superiormente o tamanho da tabela procurada por (dado que $2k \leq 4m$)

$$\sum_{k=1}^{2m} n^4 \times n^8 \times 2^{4m+2k} \times 2^{4m} = n^{12} \times 2^{12m} = n^{12} \times 2^{\log n^{12/\epsilon}} = n^{12} \times n^{12/\epsilon}$$

o qual é $O(n^{12}2^{12m}) = n^{O(1/\epsilon)}$.

Agora passamos a descrever como construir as entradas da tabela de procura, usando de baixo para cima as folhas do ladrilhamento. As entradas correspondendo a retângulos que contêm $4m$ vértices no máximo, são calculadas usando um algoritmo de força-bruta em um tempo $2^{O(m)}$. Para calcular uma tabela-entrada que corresponde a qualquer outro retângulo, enumera-se todas as triplas nos tipos seguintes:

(i) todas as possíveis linhas separadoras $\leq n$ combinatorialmente distintas para o retângulo,

(ii) os portais $j \leq m$ nos quais o tour atravessa esta linha separadora, e

(iii) todas as possíveis ordens nas quais estes portais poderiam ser colocados no subtour, respeitando os pares existentes impostos pelos portais $\leq 4m$ no perimetro do retângulo.

Então verifica-se as entradas apropriadas para os retângulos em qualquer lado da linha separadora, para determinar o menor custo do subtour que usa essas arestas separadoras nessa ordem. E no final ficamos somente com o subtour de menor custo dentro de todas as triplas numeradas.

O tempo de execução da programação dinâmica é certamente limitado superiormente por
$2^{O(m)} \times$ polinomial de $n \times$ tamanho da tabela, o qual é $n^{O(1/\varepsilon)}$.

5.4 Resultados prévios

Nesta seção prova-se o teorema sobre separadores e o lema dos retalhos que são importantes para a prova do Teorema Estructural.

Antes de provar o Teorema Estructural precisamos de mas uma definição e de mais alguns resultados.

Uma *linha separadora aleatória* de um retângulo é um segmento de linha reta paralela a sua menor aresta colocada, aleatoriamente para particionar o retângulo em dois retângulos de pelo menos 1/3 da área total.

Teorema 5.5. *Seja C uma coleção qualquer de segmentos de linha reta cujo comprimento total é T, e que encontra-se completamente dentro de um retângulo de tamanho W. Então o número esperado de segmentos de C que atravessam uma linha separadora aleatória do retângulo é no máximo $3T/W$.*

Prova: Seja e um segmento de linha reta de comprimento l_e. Dado que a linha separadora encontra-se dentro de uma região de comprimento $W/3$ como máximo. A probabilidade da linha reta l_e atravessar a linha separadora S é

$$P(S) = \frac{\text{Proj-comprimento de } e}{\text{comprimento de } W/3} \leq l_e/(W/3)$$

(Note bem: se $l_e > W/3$, a probabilidade poderia ser 1, que é porem menor que $l_e/(W/3)$). Consequentemente o número esperado de arestas atravessando a linha separadora é no máximo

$$\frac{l_a}{W/3} + \frac{l_b}{W/3} + \ldots + \frac{l_f}{W/3} = \frac{1}{W/3}(l_a + l_b + \ldots + l_f) = \frac{1}{W/3}\sum_{e \in C} l_e = \frac{T}{W/3} = \frac{3T}{W} \blacksquare$$

Observações:

Em particular teorema 5.2 prova a existência de uma linha separadora que é atravessada por no máximo $3T/W$ arestas de C.

Cada vez que usarmos o teorema 5.5, em conexão com o PCV, a coleção de segmentos de linha C mencionada na hipótese, é parte de um tour ótimo ou quase-ótimo do caixeiro viajante.

Agora apresenta-se outro lema, o Lema dos Remendos, que esta implícito em trabalhos anteriores sobre o problema euclideano [6], [22]. Observamos ainda que provando o Teorema Estrutural estamos dizendo o seguinte: Se um tour do caixeiro viajante tem um conjunto grande de arestas que atravessam por uma faixa "estreita", então existe um tour de custo "não muito maior" no qual muito poucas arestas atravessam tal faixa. Também, este novo tour deveria poder ser obtido do tour original usando algumas trocas locais de arestas que não afetem as arestas longe da faixa. O lema apresentado a seguir nos permite provar exatamente este fato, que tal vez já fosse conhecido há longo tempo (por exemplo, isto é implícito nos cálculos de [22]).

Lema 5.6. *(Lema dos Remendos). Existe uma constante $b > 0$ tal que o seguinte é verdadeiro. Seja S qualquer segmento de linha de comprimento l e π um caminho fechado que atravessa S pelo menos três vezes. Então existem segmentos de linha em S cujo comprimento total é $b \cdot l$ e cuja adição a π o transforma num caminho fechado π' que atravessa S no máximo duas vezes.*

Prova: Suponha que π atravessa S um total de t vezes. Seja $2k$ o maior número par menor ou igual à t. Identifique $2k$ pontos $M_1, \ldots, M_{2k}$ esses pontos onde π atravessa S (veja esquema na figura 5.7 para $k = 4$).

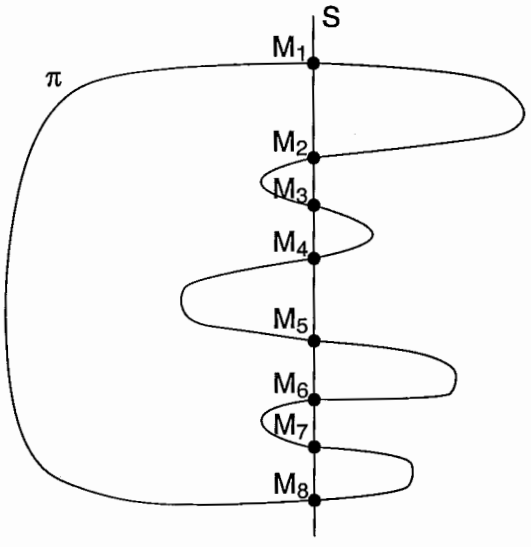


Figura 5.7: Esquema de um tour atravessado por um segmento de linha l

Divida π nesses pontos, dando origem a $2k-1$ caminhos $P_1, P_2, \ldots, P_{2k-1}$. Precisaremos de duas cópias de cada M_i um para cada lado de S. Denote por M_i' e M_i'' estas cópias. (O esquema é mostrado na figura 5.8).

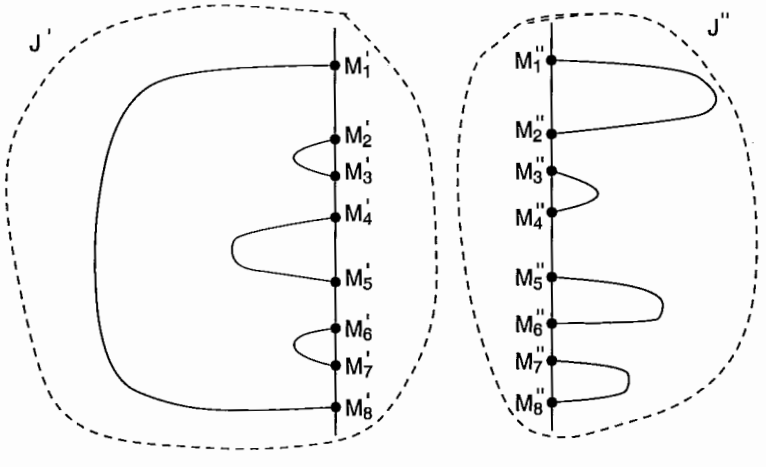


Figura 5.8: Esquema das duas copias M' e M''

Seja J o multiconjunto de segmentos de linha que consistem do seguinte:

(i) Um tour π^* do caixeiro-viajante de custo mínimo passando por $M_1, \ldots, M_{2k}$,

(ii) Um emparelhamento E quase-perfeito de custo mínimo entre os $M_1, \ldots, M_{2k}$.

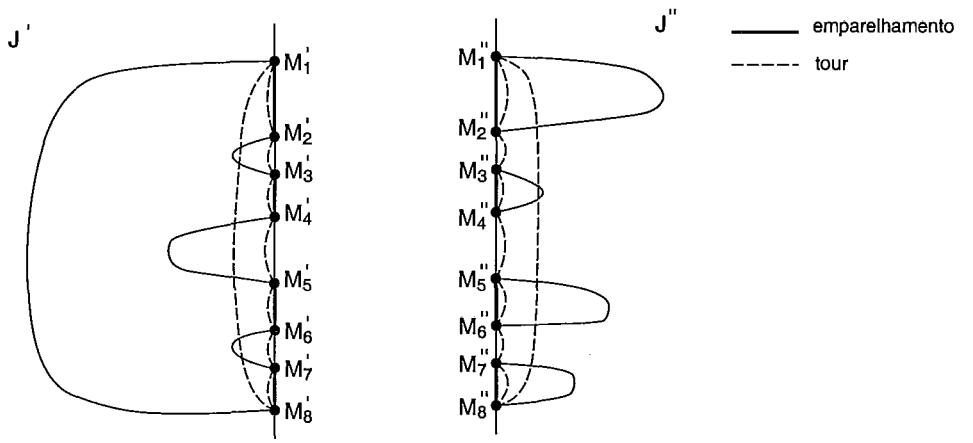


Figura 5.9: Construção dos multiconjuntos J' e J''

Note que os segmentos de linha de J encontram-se em S (veja figura 5.9) e seu comprimento total é no máximo $3l$, pois

$$|E_{c/M_i's}| \leq l$$

e

$$|\pi^*{}_{c/M_i's}| \leq 2l$$

Toma-se duas copias J' e J'' de J e as adicionamos a π. Consideramos J' como posicionado à esquerda de S e J'' posicionado à direita de S.

Agora para juntar estes caminhos formados e obter um tour π', adiciona-se arestas (considerando que tem comprimento zero) como segue:

sendo que $2k$ é o maior número par menor do que t, então adiciona-se uma aresta entre M'_{2k-1} e M''_{2k-1} e uma aresta entre M'_{2k} e M''_{2k}

Juntamente com os caminhos $P_1, \ldots, P_{2k-1}$, esses segmentos adicionados e arestas definem um grafo conexo, 4-regular nessa união

$$\{M'_1, \ldots, M'_{2k}\} \cup \{M''_1, \ldots, M''_{2k}\}.$$

Um passeio euleriano deste grafo (figura 5.10) é um caminho fechado que contém $P_1, \ldots, P_{2k-1}$ e atravessa S no máximo duas vezes. Logo provamos o teorema para $b = 6$. ■

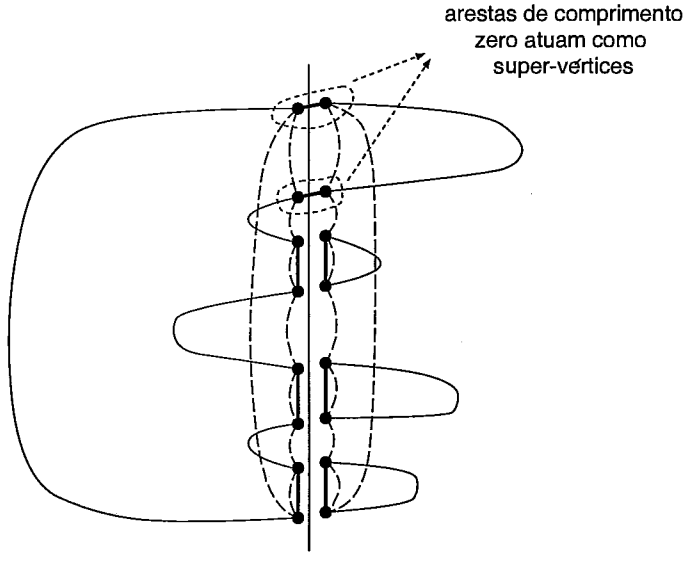


Figura 5.10: caminho fechado π'

Observação:

O argumento que usa a técnica de Christofides mostra para $b = 3$.

Prova (Teorema Estrutural): A idéia principal é começar com um tour ótimo no conjunto original de vértices e um ladrilhamento vazio, e refinar o ladrilhamento (i.e., aumentar a sua profundidade) um nível de cada vez, sempre assegurando que o tour atual é m-leve em relação aos retângulos que já estão no ladrilhamento. O refinamento a cada nível:

i) determina uma linha separadora em todos os retângulos de nível anterior que tenham tamanho > 1;

ii) identifica alguns portais nesta linha separadora para adicionar ao conjunto corrente de vértices; e,

iii) modifica o tour corrente para atravessar a linha separadora usando somente estes portais.

Se mostrará que tal modificação para o tour corrente aumenta o seu custo de um fator multiplicativo $(1 + \frac{3b}{m})$, onde b é uma constante que aparece no enunciado do *Lema dos Remendos*. Como o refinamento do ladrilhamento detêm-se depois de $2log_{1.5}L$ níveis, o custo do tour final está dentro de um fator $(1+\frac{3b}{m})^{2log_{1.5}L}$ do ótimo.

Este fator é menor que $(1+\frac{\varepsilon}{2})$ quando $m = 12b\log_{1.5} L/\varepsilon$.

Agora descrevemos como refinar um ladrilhamento de profundidade d em um ladrilhamento de profundidade $d+1$. Seja R um retângulo de tamanho W que está a profundidade d e seja T o comprimento das arestas do caminho corrente do caixeiro viajante que se encontram no interior de R. (Sabe-se que o tour entra em R somente pelos portais). Agora se mostra como particionar R em dois retângulos escolhendo uma linha separadora apropriada.

Caso 1: $T \leq mW/3$. Seja $\mu = 3T/W \leq m$. O teorema 5.5 garante a existência de uma linha separadora S que é atravessada por no máximo μ arestas do caminho corrente. Colocamos m portais em S a intervalos de $\frac{W'}{m-1}$ (dado que $W' \leq W$). A seguir, para cada ponto em S que é atravessado pelo caminho vamos identificar o portal mais próximo. Desloca-se a aresta em questão de no máximo $\frac{W'}{2(m-1)}$, de maneira que passe por esse portal (veja figura 5.11). Então adicionamos esse portal ao conjunto corrente de pontos.

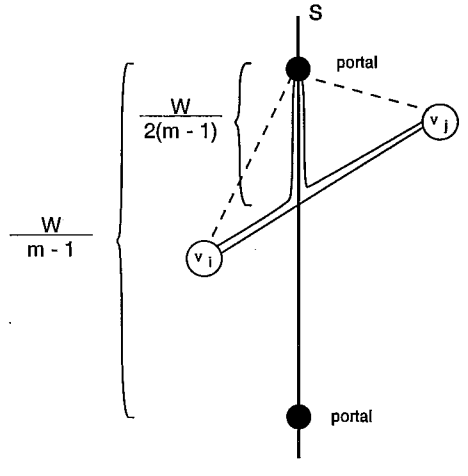


Figura 5.11: (v_i, v_j), aresta deslocada

No final, se assegura que todos os cruzamentos $\leq \mu$ acontecem em um dos m portais. Para assegurar isso, nós tivemos que incrementar o custo incorrido no retángulo R, de T para no máximo $(1+\frac{3}{m-1})T$. De fato:

$$T + 2\mu\frac{W'}{2(m-1)} = T + \mu\frac{W'}{m-1} = T + \frac{3T}{W'}\cdot\frac{W'}{m-1} = T + \frac{3}{m-1}T.$$

Observe que:

$$\frac{3}{m-1} \leq \frac{3b}{m} \Longleftrightarrow m \leq b(m-1).$$

Agora se $b \geq 2 \Longrightarrow b(m-1) \geq 2(m-1) = m + \underbrace{m-2}_{>0} > m$ para $m > 2$

Logo $T + \frac{3}{m-1}T \leq T + \frac{3b}{m}T = (1 + \frac{3b}{m})T$ para $b \geq 2$.

Caso 2: $T > mW/3$. Neste caso, tomamos *qualquer* linha separadora S dentro do retângulo R. O caminho corrente pode estar atravessado S muitas vezes, mas usa-se o *Lema dos Remendos* para modificar o caminho de forma que atravesse S no máximo em 2 lugares.

Como $W' \leq W < \frac{3T}{m}$ implica $W' \leq \frac{3T}{m}$. Então o custo do remendo é no máximo $bW \leq 3bT/m$, onde b é a constante mencionada no Lema dos Remendos. Assím o custo do tour corrente dentro de R aumentou de T para $T(1 + \frac{3b}{m})$. (O custo de mover os dois cruzamentos para os portais mais próximos é $2W/m < T/m^2$, o qual é muito pequeno).

Finalmente, note bem que em ambos casos as modificações para o tour envolveram arestas que estão dentro de R, assim a propriedade de m-leveza não é afetada em nenhum lugar, no ladrilhamento.

A cada refinamento, para que o caminho do caixeiro viajante não se auto intercepte substituimos as partes do caminho corrente dentro de cada retângulo pelo subtour ótimo que visitam todos os vértices que estão dentro do retângulo, enquanto continuamos entrando e saindo do retângulo atravez dos portais já identificados na fronteira. ∎

5.5 Ilustração do exemplo

Nesta seção ilustra-se com um exemplo o Esquema Aproximativo desenvolvido por Arora. Considere o seguinte roteiro:

1. Começamos com um tour ótimo no conjunto original de vértices

2. Encontra-se um $1/3 - 2/3$ - ladrilhamento,

3. Para cada nível, a profundidade do ladrilhamento determinará:

   (i) Uma linha separadora dentro da caixa limitadora,
   (ii) Se identifica alguns portais nesta linha separadora
   (iii) Modifica-se o tour corrente para atravessar a linha separadora usando somente estes portais.

4. Encontramos a parte do caminho do caixeiro viajante sobre ambos lados da linha separadora.

Ao decompor uma entrada em 3 escolhas (i), (ii) e (iii); primeiro resolve-se as escolhas em (i), logo as escolhas em (ii) e finalmente as escolhas em (iii).

Suponha que as escolhas em (i) ocorrem em m formas, e não têm relação no sentido de que as escolhas em (ii) ou (iii) ocorrem em n ou q formas respectivamente apesar dos resultados das escolhas em (i). Então cada entrada da tabela ocorre em $m \times n \times q$ formas

A programação dinâmica como descrita acima, somente calcula o custo do caminho ótimo m-leve, e não o caminho em si. Mas o caminho pode ser reconstruído da tabela de procura no final, analizando as decisões feitas em cada passo da programação dinâmica.

Agora, porque insistir que o caminho do caixeiro viajante entra e saia das regiões do retângulo somente em portais? Porque isto ajuda na programação dinâmica enumerando todas as possíveis maneiras no qual o caminho podería entrar e sair da região.

O algoritmo poderia retornar um caminho do caixeiro viajante que é auto-atravessado.

Seja dada o conjunto de cidades $\{1, 2, 3, 4\}$ especificados pelos vértices v_1, v_2, v_3 e v_4 respectivamente e a matriz correspondente d_{ij} para cada duas cidades i e j (figura 5.12) e o tour ótimo no conjunto original de vértices mostrado na figura 5.19(a)

cujo custo é 8.714.

Agora será feita a construção do $1/3 - 2/3$ -ladrilhamento.

$$d_{ij} = \begin{bmatrix} 0 & 1.41 & 3 & 2.24 \\ 1.41 & 0 & 2.24 & 3 \\ 3 & 2.24 & 0 & 2.82 \\ 2.24 & 3 & 2.82 & 0 \end{bmatrix}$$

Figura 5.12: Instância

Observe que o comprimento de sua aresta mais longa é 3, assim todos as vértices se encontram dentro do menor retângulo de tamanho da caixa limitadora de 3, neste caso a caixa limitadora resulta ser um quadrado (veja figura 5.13)

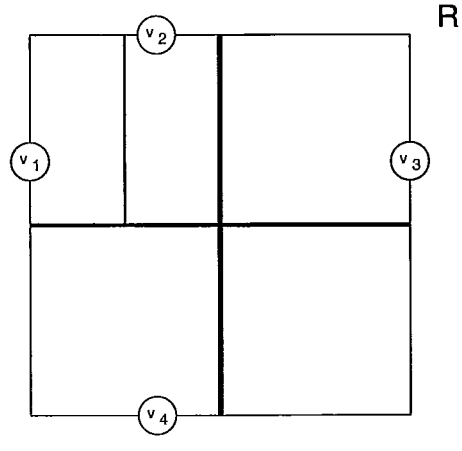


Figura 5.13: Vértices dentro do retângulo R

Começa-se agora a refinar o ladrilhamento, um nível de cada vez. Em particular, toma-se a linha separadora na metade, i.e., o retângulo será dividido em dois subretângulos de igual área (veja figura 5.14), no primeiro nível do refinamento se identifica os portais p_1, p_2 e p_3.

Esta divisão será feita até que os subretângulos tenham tamanho ≤ 1 ou contenham um só vértice, no segundo nível particiona-se os retângulos R_1 criando-se os retângulos R_3 e R_4 (figura 5.15(a)),

e R_2 criando-se os retângulos R_5 e R_6 mostrados na figura 5.15(b).

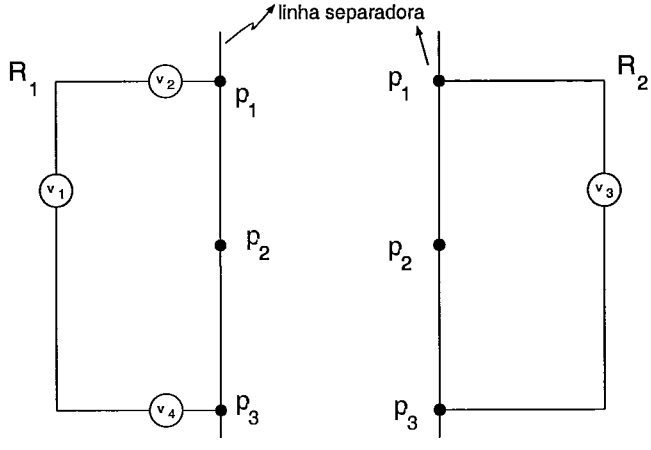


Figura 5.14: Primeiro nível

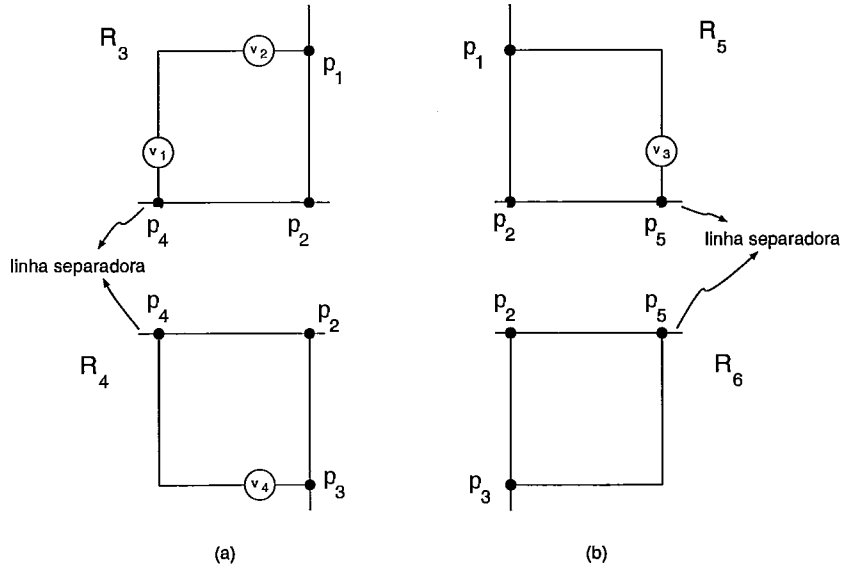


Figura 5.15: Segundo nível

Finalmente no terceiro e último nível particiona-se o retângulo R_3 em dois retângulos R_7 e R_8 (veja figura 5.18).

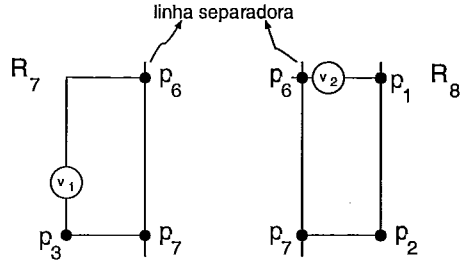


Figura 5.16: Terceiro nível

Desta forma pode-se construir a árvore binária de retângulos R_i, $i = 1, 2, \ldots, 8$ (figura 5.17), e cada retângulo representa um subproblema que será resolvido usando a técnica da programação dinâmica. Então a sequência para resolver os subproblemas criados começará pelas folhas da árvore binária de retângulos (veja figura 5.17). Neste exemplo começamos pelo retângulo R_8.

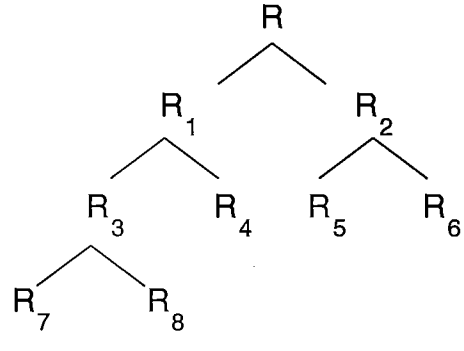


Figura 5.17: 1/3 : 2/3 -ladrilhamento

Para obter os resultados dos subproblemas, será construido uma tabela mostrada a seguir:

Tabela 5.1: (para o subproblema R_8)

mostra as distâncias de todas as possíveis ordens nas quais estes portais p_1, p_2, p_6 e p_7 e sempre passando pela cidade $2 = v_2$, poderiam ser colocados no subtour, por exemplo o custo do caminho dos portais p_1 a p_2 passando por v_2 é $d_{p_1 v_2} + d_{v_2 p_2} = 0.5 + 1.58 = 2.08$.

	p_1	p_2	p_6	p_7
p_1		2.08	0.75	2.02
p_2			1.83	3.1
p_6				1.75

Tabela 5.1: Para o retângulo R_8

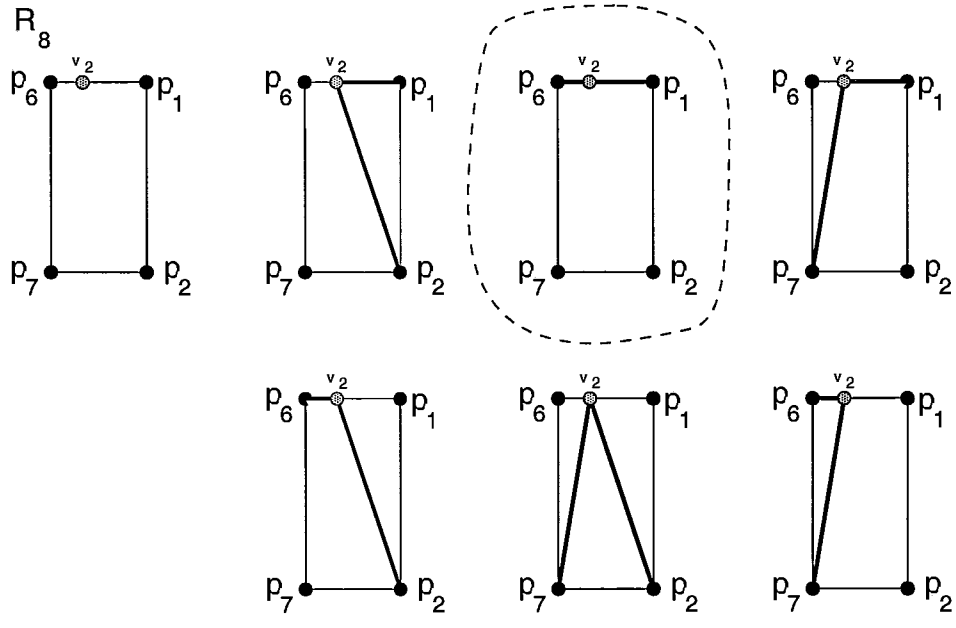


Figura 5.18: Solução para o subproblema R_8.

A seguinte Tabela 5.2 para o subproblema R_7 inclui a cidade $1 = v_1$ e os portais p_3, p_6 e p_7.

	p_4	p_6	p_7
p_4		1.75	1.40
p_6			2.15

Tabela 5.2: Para o retângulo R_7

Tabela 5.3 (Para o subproblema R_3) onde existe caminho passando pela partição dos portais usados na tabela 5.1 e tabela 5.2, então as distâncias respectivas são dados a seguir na tabela 5.3:

	p_4	p_6	p_7
p_1	2.5	4.17	2.9
p_2	3.58	5.25	3.98
p_4		3.15	3.5

Tabela 5.3: Para o retângulo R_3

Agora agrupa-se os retângulos R_3 e R_4 para obter o retângulo R_1 e os resultados são mostrados na Tabela 5.4:

	p_2	p_3	p_4	p_6	p_7
p_1	5.88	4.8			
p_2		5.88		6.88	6.88
p_3			5.66	5.45	5.8
p_4				8.63	7.36

Tabela 5.4: Para o retângulo R_1

A seguir, são dados os resultados para os subproblemas dos retângulos R_5 e R_6 nas Tabela 5.5 e Tabela 5.6 respectivamente.

	p_2	p_5
p_1	3.38	2.3
p_2		2.08

Tabela 5.5: Para o retângulo R_5

Agora agrupando os subproblemas anteriores, R_5 e R_6, obtém-se o subproblema R_2 cujos resultados são dados na Tabela 5.7.

Finalmente é resolvido o problema, juntando os resultados dos subproblemas dos retângulos R_1 e R_2 para obter os resultados do retângulo principal R cujos dados são dados a seguir na Tabela 5.8.

	p_3	p_5
p_2	1.5	1.5
p_3		2.12

Tabela 5.6: Para o retângulo R_6

	p_2	p_3
p_1	3.8	4.42
p_2		4.42

Tabela 5.7: Para o retângulo R_2

Destes resultados, obtivemos um custo de um tour pasando por todas as cidades aproximado de 9.22. Agora o caminho pode ser reconstruido das tabelas analizando as decisões feitas em cada passo da programação dinâmica

$[p_1, v_2, p_6, v_1, p_4, v_4, p_3, p_5, v_3]$ (veja figura 5.19)

	p_2	p_3
p_1	9.22	9.22
p_2		10.30

Tabela 5.8: Para o retângulo R

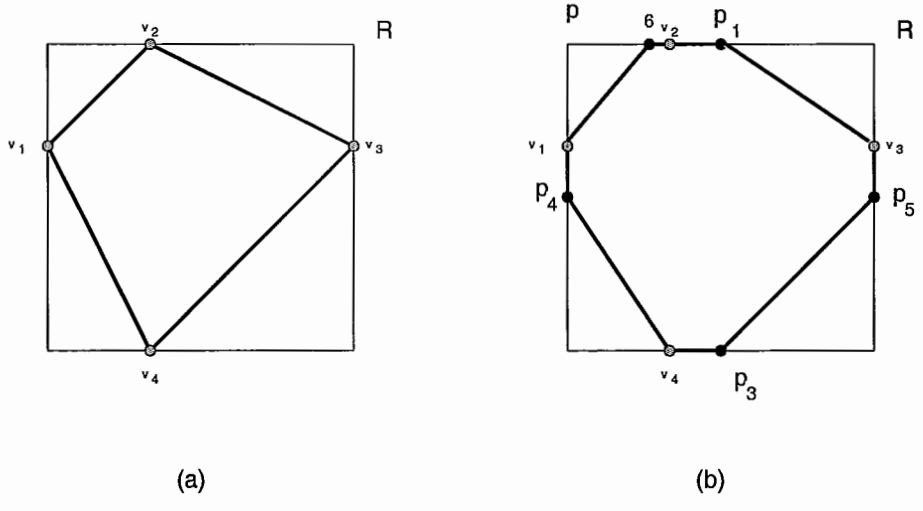


Figura 5.19: (a)Tour ótimo. (b)Tour aproximado

Capítulo 6

Conclusões

O objetivo dessa tese foi estudar os algoritmos aproximativos mais importantes existentes na literatura para o Problema do Caixeiro Viajante Euclideano (Δ PCV). Apresentamos com detalhes o algoritmo da Árvore e o algoritmo de Christofides. E estudamos o esquema aproximativo para o *PCV* euclideano, recentemente desenvolvido por Arora.

Com relação ao Esquema Aproximativo em tempo polinomial, mostramos algumas definições novas, e resultados para mostrar que o tour obtido para o *PCV* euclideano é feito em tempo polinomial, efetuando partiçã geométrica recursiva e seguidamente aplica-se a técnica de programação dinâmica.

Nossa maior contribuição consistiu na tentativa de estudar em detalhes o esquema de aproximação de Arora [2]. Este algoritmo é reconhecidamente difícil, tendo o próprio autor produzido três versões [3] e [4] do mesmo artigo.

Observamos que ainda permanecem em aberto dois problemas importantes relacionados ao *PCV*:

(i) desenvolver um algoritmo aproximativo melhor que o de Christofides para espaços métricos, em geral.

(ii) desenvolver uma versão ainda mais eficiente do esquema aproximativo de Arora.

Referências Bibliográficas

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, (1974).

[2] S. Arora. Polynomial time approximation shemes for euclideam TSP and other geometric problems, 2–11. *Published in proc 37th. IEEE Symposium on Foundations of Computer Science*, 1996.

[3] S. Arora. Nearly time approximation shemes for euclideam TSP and other geometric problems, 20–22. *In 38th Annual Symposium on Foundations of Computer Science, Miami Beach, florida*, (1997).

[4] S. Arora. Polinomial time approximation shemes for euclideam TSP and other geometric problems. *Published in journal of ACM,vol 45,* **5**, *753–782*, (1998).

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Published in journal of ACM, vol 45,* **3**, *501–555*, (1998).

[6] J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Soc,* **5**, *299–327*, (1959).

[7] R.E. Bellman. *Dynamic programming.* Princeton University Press, Princeton, (1957).

[8] J.A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* North-Holland, New Jork, (1976).

[9] R.E. Campello and N. Maculan. *Algoritmos e Heurísticas: Desemvolvimento e avaliação de performance.* EDUFF, Niteroi, (1994).

[10] N. Christofides. *Graph theory: an algorithmic approach.* Academic Press, New York, (1975).

[11] N. Christofides. *Worst-Case Analisys of an Algorithms for the Traveling Salesman Problem.* In J. F. Traub, editor, Sympos. on New Directions and Recent Result in Algorithms and Complexity, New York, NY,Academic Press, 441, (1976).

[12] T. H. Cormen, Ch. E. Leiserson, and R. L. Rivest. *Introdution to Algorithms.* The MIT Press, Cambridge, Massachusetts, (1990).

[13] G. Cornuéjols and G. L. Newhauser. Tight bounds for christofides TSP heuristic. *J. Math. Prog.* **8**,*163*, (1978).

[14] G.B. Dantzig, S.M. Fulkerson, and S.M. Johnson. Solution of large scale traveling salesman problem. *Oper. Res.* **2**,*393–40*, (1954).

[15] M. R. Garey and D. S. Johnson. Computers and Intractability*: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, (1979).

[16] M. Grigni, D. Karger, P. Klein, A. Woloszyn, and S. Arora. A polynomial-time approximation scheme for weighted planar graph tsp. *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 33–41*, (1998).

[17] F.L. Hitchcock. The distribution of a product from several sources to nume rous localities. *J. Math e Phys,* **20** *[1:4],224–230*, (1941).

[18] D.S. Hochbaum. *ed. Approximation Algorithms for NP−Hard problems.* PWS Publishing Boston, (1996).

[19] A.J. Hoffman and P. Wolfe. *History. In:*E. L. Lawler et alii, eds., *The Traveling Salesman Problem: a guided tour of combinatorial optimization.* John Wiley, Chichester, 1–15, 1985.

[20] J.E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languajes and Computation.* Addison-Wesley, Reading, Massachusetts, (1979).

[21] E. Horowitz and S. Sahni. *Fundamentals of computer Algorithms.* Springer Verlag, (1978).

[22] R.M. Karp. *Probabilistic analysis of partitioning algorithms for the TSP in the plane.* Math. Oper. Res. **2**, 209–224, (1977).

[23] E. Koutsopias, M. Grigni, and C. H. Papadimitriou. An approximation scheme for planar graph TSP. *Published in Proc IEEE symposium on Foundations of computer science, 640–645*, 1995.

[24] H.L. Larson. *Introduction to probability theory and statistical inference.* by Jonh Wiley, Inc, Canada, (1974).

[25] E.L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley, Chichester, (1985).

[26] H.R. Lewis and C.H. Papadimitriou. *Elements of the theory of computation.* Prentice hall, New Jersey, (1998).

[27] K. Menger. Das botenproblem. *Menger 1932, 9. Kolloquium (5.II), 12. [1:2, 7]*, (1930).

[28] K. Menger. Ergegnisse eines mathematischen kolloquiums 2. *Teubner, Leipzip [Bibliography]*, (1932).

[29] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: Part II - a simple PTAS for geometric K-MST, TSP and related problems. *Preliminary manuscript*, (1996).

[30] C.H. Papadimitriou. Euclidean TSP is NP-complete. *Published in TCS,***4**, *237–244*, (1977).

[31] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall Englewood Cliffs, NJ, (1982).

[32] C.H. Papadimitriou and Yannakakis. Optimization, approximation and complexity classes. *J. Computer and Systems Sci. 425–440*, (1991).

[33] R.C. Prim. Shortest connection networks and some generalizations. *Arch. Math. Phys.* **27**, *742-744*, 1957.

[34] G. Reinelt. TSPLIB-A travelling salesman problem library. *ORSA J. Computing,* **3**, *376-384*, (1991).

[35] S. Sahni and T. Gonzales. P-complete approximation problems. *Published in J. ACM,* **23**, *555–565*, (1976).

[36] J.L. Szwarcfiter. *Grafos e Algoritmos Computacionais.* Campus, Rio de Janeiro, (1986).

[37] L. Trevisan. When hamming meets euclid: *The approximability of geometric* TSP and MST. *In Proc. 29th ACM STOC,21–39*, 1997.

[38] B. F. Voigt. *Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Auftrage zu erhalten und eines glucklichen Erfolgs in seinen Geschaften gewiss zu sein, Von einem alten Commis - Voyageur, Ilmenau.* (Republished Verlag Bernd Schramm Kiel.) [1 : 3], (1981).

[39] R. J. Wilson and J. Watkins. *Graph: An introdutory approach a first course in discrete mathematics.* USA, (1990).

[40] D. Wood. *Theory of computation.* John Wiley, New York, 471–488, (1987).