ESTUDO DE ALGORITMOS PARA A RESOLUÇÃO COM

PUTACIONAL DO PROBLEMA DO CAIXEIRO VIAJANTE

Aurora Akiko Kawahara

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 OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

Prof. Nelson Maculan Filho

Prof. Cláudio T. Bornstein

Profa. Dina Feigenbaum Cleiman

RIO DE JANEIRO, RJ - BRASIL

AGOSTO DE 1978

KAWAHARA, AURORA AKIKO

Estudo de Algorítmos para a Resolução Computacional do Problema do Caixeiro Viajante |Rio de Janeiro| 1978.

VI , 98 p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e COmputação, 1978)

Tese - Univ. Federal do Rio de Janeiro. Fac. Engenharia

1. Assunto: Formulação e Métodos utilizando Programação Inteira (0-1) do Problema do Caixeiro Viajante. I. COPPE/UFRJ II. Título: Estudo de Algoritmos para a Resolução Computacional do Problema do Caixeiro Viajante (série).

A G R A D E C I M E N T O S

Agradeço ao Prof. Nelson Maculan Filho pela confiança e orientação durante o curso de mestrado.

Agradeço também a Profa. Maria Terezinha T. Torneros pelo incentivo inicial ao Curso de Mestrado.

Ao CNPq e a CAPES pela ajuda financeira.

R E S U M O

Neste trabalho são estudadas as principais formulações utilizando Programação Inteira (0-1) do problema do Caixeiro Viajante, bem como alguns métodos exatos e aproximados para sua resolução e generalizações.

É também incluído um Programa em FORTRAN do Método Heurístico de Karg - Thompson e seus resultados.

A B S T R A C T

In this work the principal formulations using Integer programming (0-1) of the Traveling Salesman Problem, were studied, and also approximate and exat methods for its resolution and generalizations.

A Program in Fortran of the Heuristic Method of Karg-Thompson and its results has also been included.

I N D I C E

C A P I T U L O I

I.1. INTRODUÇÃO

O problema do caixeiro viajante (P.C.V.) pode ser formulado pelo seguinte: Dada uma lista com n cidades, encontrar a ordem em que um caixeiro viajante deve visitar cada uma delas apenas uma vez e voltar para o ponto de partida, tal que a rota seja de custo mínimo. (O custo pode ser expresso em dinheiro, tempo, distância, etc).

Na terminologia da Teoria dos Grafos, pode ser definido como: encontrar o menor circuito Hamiltoniano de um grafo. (circuito Hamiltoniano: é uma linha passando por todos os vértices do grafo, exatamente uma vez, com um arco ligando os dois vértices extremos).

Este problema foi proposto formalmente, pela primeira vez, em 1934 por Hassler Whitney num seminário na Universidade de Princeton.

Apesar do P.C.V. ser facilmente 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.

Alguma das Aplicações mais comuns do P.C.V.:

- distribuição de alguns produtos (bebida, gás, correio, etc) para uma extensa rede de fregueses, através de uma frota de veículos. O que se quer é estabelecer uma rota para cada veículo, de modo a minimizar o custo total da viagem (gastos de combustível, mão-de-obra, etc).

- esquematização de um conjunto de tarefas que devem ser efetivadas por uma máquina, onde a mudança de uma tarefa para outra demora um certo tempo variável e o problema é escolher a sequência das tarefas que minimize o tempo para o conjunto das tarefas.

Este trabalho propõe o seguinte: estudar as principais formulações do P.C.V., alguns métodos exatos e aproximados para sua resolução e generalizações.

Dentre os métodos exatos destacamos o de Dantzig [5] que foi um dos primeiros a resolver o P.C.V. usando a idéia de dualidade e recentemente Held e Karp [10] e [11] e Bazaraa [1], definiram o problema dual do P.C.V. e resolveram o problema usando o método ascendente num procedimento de branch-and-bound.

Little [9] descreve um método branch-and-bound de construção de rotas, com um limite inferior no conjunto de todas as rotas, calculado pelo procedimento de redução da matriz de distância.

Bellmore e Malone [3] usam a idéia de branch

-and-bound, mas para eliminar subrotas na solução de um Problema de Alocação e seu valor é o limite inferior para o P.C.V.

Quanto aos métodos aproximados destacamos: O método de Lin e Kernighan [14] que é um procedimento de melhoramento de uma rota, com a troca de k-arcos por outros k-não pertencentes a esta rota e ver se ocorre uma melhora na solução.

O método de Karg-Thompson [11] é um método de construção de uma rota e para n grande também de melhoramento.

Foi programado este último método em FORTRAN IV, com exemplos de 33 e 42 cidades e com soluções bem perto do ótimo.

Finalmente, no último capítulo referimos a duas generalizações do P.C.V., sendo a primeira o problema de sequenciamento de n tarefas em uma máquina com o custo de passar de uma tarefa a outra, dado por funções especiais e a segunda o P.C.V. dependente do tempo.

C A P Í T U L O II

II.1. ALGUMAS FORMULAÇÕES PARA O PROBLEMA DO CAIXEIRO VIAJANTE (P.C.V.)

Supondo que seja dada uma lista com n cidades a um caixeiro viajante, onde cada para de cidades i e j são ligadas por um arco de custo c_{ij}.

O problema é encontrar a rota de custo mínimo, de modo que, ele passe em cada cidade apenas uma vez e depois retorne para o seu ponto de partida.

II.2. FORMULAÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE EM PROGRAMAÇÃO INTEIRA

O problema do caixeiro viajante (P.C.V.) é formulado a partir do Problema de Alocação, representado pelo seguinte:

Dado $C = (c_{ij})$ a matriz de custo da cidade i para a cidade j ($i \neq j$)

$$\text{minimizar} \quad \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij}\, x_{ij} \quad \text{e } c_{ii} = \infty \quad \forall i=1,\ldots,n$$

$$\text{sujeito a:} \quad \text{a)} \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1,\ldots,n$$

$$\text{b)} \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1,\ldots,n$$

$$\text{c)} \; x_{ij} \in \{0,1\}$$

Apesar de todo x_{ij} de uma subrota satisfazer as condições acima, nem sempre um solução viável deste problema é uma rota, podendo ter como solução várias subrotas. Para evitar que isto aconteça, é necessário acrescentar mais algumas restrições ao Problema de Alocação.

Antes de formular estas restrições, vai ser definido os tipos de problemas que são classificados quanto a natureza dos elementos da matriz de custo.

1º) P.C.V. SIMÉTRICO - quando $c_{ij} = c_{ji}$, $\forall i,j$.

2º) P.C.V. ASSIMÉTRICO - quando c_{ij} não é necessaria - mente igual a c_{ji}.

3º) P.C.V. SIMÉTRICO E SATISFAZENDO A DESIGUALDADE TRIANGULAR - $c_{ij} = c_{ji}$ e $c_{ik} \leq c_{ij} + c_{jk}$ para todo i, j e k.

Os c_{ij}'s podem ser determinado aleatoriamen-

te ou serem especificados a partir de alguma aplicação particular.

II.3. TIPOS DE RESTRIÇÕES

Existem vários tipos de restrições que acrescentadas ao Problema de Alocação, evitam ou eliminam as subrotas.

Serão mencionados aqui três delas:

RESTRIÇÃO DO TIPO I

Wagner [20], acrescenta ao Problema de Alocação as variáveis u_i, $i = 2, 3, \ldots, n$ e $(n-1)^2 - (n-1)$ restrições do tipo:

$$u_i - u_j + n\, x_{ij} \leq n - 1 \qquad \text{para } i = 2,3,\ldots,n$$

$$u_i,\ u_j \geq 0 \text{ e inteiros} \qquad \text{para } j = 2,3,\ldots,n\ (i \neq j)$$

A formulação completa do P.C.V. consiste de $n^2 - n + 2$ restrições lineares e $n^2 - n + 1$ variáveis inteiras.

$$\text{Minimizar } \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij}\, x_{ij} \text{ e } c_{ii} = \infty \quad \text{para } \forall i=1,2,\ldots,n$$

$$\text{sujeito a: a) } \sum_{\substack{j=1 \\ j\neq 1}}^{n} x_{ij} = 1 \qquad \forall i = 1,\ldots,n$$

b) $\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1 \qquad \forall j = 1,\ldots,n$

c) $x_{ij} = \begin{cases} 1 \text{ se o par } (i,j) \text{ pertence a rota} \\ 0 \text{ caso contrário} \end{cases}$

d) $u_i - u_j + n\, x_{ij} \leq n-1$ para $i=2,3,\ldots,n$ $j=2,3,\ldots,n$ $(i \neq j)$

e) $u_i, u_j \geq 0$ e inteiros.

As restrições de a) e d): 1 - evitam as subrotas e 2 - garante que nenhuma rota seja excluída.

Demonstração: 1) evitam as subrotas.

Supondo que possa existir uma subrota $(r_1, r_2,\ldots,r_j)$ com $j < n$ e $r_i \neq 1$ $i = 1,\ldots,j$.

$xr_1,r_2 = xr_2,r_3 = \ldots = xr_{j-1}, r_j = 1$ por c).

Substituindo em d)

$ur_i - ur_{i+1} + n\, xr_i, r_i + 1 \leq n - 1 \quad i = 1,\ldots,j-1$

$ur_i - ur_{i+1} + n \leq n - 1$

$ur_i - ur_i + 1 \leq - 1$ para $i = 1,\ldots j-1$

Somando-se todas as desigualdades, para i de 1 a j - 1, temos que:

$$
\begin{aligned}
i \qquad & ur_1 - ur_2 \leq -1 \\
& ur_1 - ur_3 \leq -1 \\
& \vdots \\
& ur_{j-1} - ur_j \leq -1 \\
& ur_j - ur_1 \leq -1
\end{aligned}
$$

$j \leq 0$, contradição pois $1 \leq j < n$.

2) Garante que nunhuma rota seja excluída:

É suficiente mostrar que existem valores para u_i, que satisfazem a última restrição, para qualquer rota dada.

Considerando uma rota, que por definição passa por cada uma das cidades i = 2,3,...,n apenas uma vez.

Seja t_i - a posição na rota, em que a cidade i é visitada e $t_1 = 1$ para a cidade 1.

Então $u_i = t_i$ para i = 2,3,...,n é viável.

Quando $x_{ij} = 1$, $t_j = t_{i+1}$ a restrição e) para este x_{ij} é satisfeita desde que:

$$t_i - t_{i+1} + n \leq n - 1$$

$$t_i - t_{i+1} = -1$$

Quando $x_{ij} = 0$

$$u_i - u_j \leq n - 1 \quad , \quad \text{para} \quad u_i \leq n$$

$$u_j \geq 1$$

RESTRIÇÃO DO TIPO II

Bellmore e Malone [3] dão outra restrição , que acrescentada ao Problema de Alocação, elimina as subrotas.

Seja S, $\bar{S}$ uma partição de inteiros i=1,...,n, isto é, $S \cap \bar{S} = \phi$ e $S \cup \bar{S} = \{1,2,...,n\}$.

A restrição é do tipo:

$$\sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geq 1 \quad \text{para toda partição } (S,\bar{S}) \ ,$$

obriga a existência de pelo menos um arco orientado ligando duas subrotas.

O P.C.V. é formulado por:

$$\text{Minimizar} \quad \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} \quad c_{ii} = \infty \quad \forall i = 1,...,n$$

Sujeito a: a) $\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n$

b) $\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n$

c) $\sum_{i\in S} \sum_{j\in \bar{S}} x_{ij} \geq 1$ para toda partição$(S,\bar{S})$

e) $x_{ij} = \begin{cases} 1 & \text{se o para } (i,j) \text{ pertence a rota} \\ 0 & \text{caso contrário} \end{cases}$

No caso das distâncias serem simétricas:

c) $\sum_{i\in S} \sum_{j\in \bar{S}} x_{ij} \geq 2$

para todas as partições $(S,\bar{S})$ não vazias e temos que se $(S,\bar{S})$ é considerada $(\bar{S},S)$ não o é. Existem $2^{n-1} - 1$ restrições deste tipo num problema com n cidades.

RESTRIÇÃO DO TIPO III

Outra restrição que evita as subrotas é:

$$\sum_{(i,j)\in S'} x_{ij} \leq k - 1$$

onde S' é a representação de pares ordenados das cidades de uma dada subrota S, de uma solução do Problema de Alocação, e k é o número de cidades de S.

O P.C.V. é formulado por:

$$\text{Minimizar} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \quad c_{ii} = \infty \qquad \forall i = 1,\ldots,n$$

$$\text{sujeito a:} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n$$

$$\sum_{(i,j)\in S'} x_{ij} \leq k - 1$$

$$x_{ij} = \begin{cases} 1 \text{ se o par } (i,j) \text{ pertence a rota} \\ 0 \text{ caso contrário.} \end{cases}$$

O número de restrições deste tipo depende do tamanho da subrota.

A redução no número de soluções inteiras viáveis impondo esta restrição III a um problema de aloca - ção é de $[k!\ e^{-1} + 0,5]$ soluções [3].

No problema com a restrição do tipo II são eliminadas $[k!\ e^{-1} + 0,5]\ [(n-k)!\ e^{-1} + 0,5]$ soluções inteiras viáveis do Problema de Alocação.

Isto porque a restrição do tipo II elimina todas as soluções possíveis de um subproblema com k-cidades, que são $[k!\ e^{-1} + 0,5]$ soluções e as soluções possíve-

is de outro subproblema com as (n-k) cidades restantes, $[(n-k)! \; e^{-1} + 0,5]$ soluções.

Como estas soluções podem aparecer no Problema de Alocação em todas as combinações possíveis, existem $[k! \; e^{-1} + 0,5] \; [(n-k)! \; e^{-1} + 0,5]$ soluções eliminadas.

Isto indica que a restrição do tipo II é mais eficiente, se $k \geq 3$, do que uma restrição do tipo III (TEOREMAS IV e V, pags. 283, de [3]).

A formulação do P.C.V. com a restrição do tipo I, usa-se um pouco menos restrições do que a restrição III, mas esta vai ser melhor, por excluir as soluções fracionárias.

Isto pode ser visto em um exemplo simples:

Supondo que temos uma solução com as subro - tas (2,3,4,2) e (5,...,n,1,5).

Com a restrição III, no caso simétrico:

$$\sum_{(i,j)\in S'} x_{ij} \leq 2 \qquad S = \{2,3,4\}$$

$$x_{23} + x_{34} + x_{42} \leq 2$$

Com a restrição I em S:

$$u_i - u_j + n \, x_{ij} \leq n - 1 \qquad \text{para} \quad i = 2,3,\ldots,n$$
$$j = 2,3,\ldots,n \; (i \neq j)$$

$$u_2 - u_3 + n\, x_{23} \leq n - 1$$

$$u_3 - u_4 + n\, x_{34} \leq n - 1$$

$$u_4 - u_2 + n\, x_{42} \leq n - 1$$

Somando-se, obtemos: $x_{23} + x_{34} + x_{42} \leq 3 - 3/n$

que é suficiente para evitar uma subrota, mas é mais fraca do que $x_{23} + x_{34} + x_{42} \leq 2$, pois admite mais soluções viáveis fracionárias.

II.4. FORMULAÇÃO DO DUAL DO PROBLEMA DO CAIXEIRO VIAJANTE

O problema dual do P.C.V. será formulado a partir do P.C.V., sob um enfoque diferente dos até agora vistos.

Seja o P.C.V.

$$\text{Minimizar} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \quad c_{ii} = \infty \quad \forall i = 1,\ldots,n$$

$$\text{a)} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1 \qquad i = 1,2,\ldots,n$$

$$\text{b)} \sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n$$

$$c)\ x_{ij} = \begin{cases} 0 & i = 1,\ldots,n \\ 1 & j = 1,\ldots,n \end{cases}$$

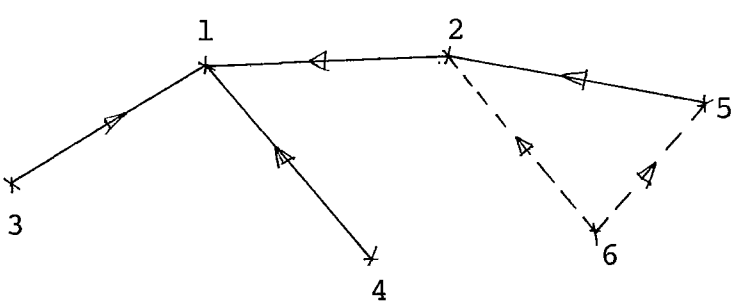
não subrotas.

Para evitar as subrotas é adotado o método de Held e Karp [11]. É o método da relaxação lagrangeana estudado por Geoffrion [8] e o dual é formulado a partir de novas definições [1].

Definição:

ÁRVORE-1 de um grafo composto de n cidades e seus arcos é uma árvore com n-1 cidades, e mais dois arcos distintos ligando as cidades restantes com qualquer duas das n-1 cidades da Árvore.

Exemplo:



Uma árvore-1 composta pela árvore com os nós 1,2,3,4, 5 e mais dois arcos a partir do vértice 6.

Uma árvore-1 corresponde a uma rota se e somente se, o ciclo formado na árvore 1 tem n arcos e é orienta

do.

O P.C.V. pode ser reformulado da seguinte maneira:

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

$$\text{sujeito a:} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n$$

$$x \in X$$

onde X = conjunto das árvores-1 do grafo completo no conjunto de nós $\{1,2,\ldots,n\}$

ou $X = (x_{11},\ldots,x_{1n},\ldots,x_{nn}) : x_{ij}=0$ ou $x_{ij}=1 (i\neq j)$ e os arcos (i,j) com $x_{ij} = 1$ formam uma árvore-1.

II.5. UM PROCEDIMENTO PARA ENCONTRAR UMA ÁRVORE-1 DE CUSTO MÍNIMO

Colocando fora um dos n nós, chamado nó inicial então:

a) encontrar a árvore de custo mínimo

que liga os (n-1) nós restantes.

b) encontrar dois arcos de custos mínimos que ligam o nó inicial com a árvore dos n-1 nós restantes.

É um problema difícil trabalhar com as duas primeiras restrições e a árvore-1 de custo mínimo. Se trabalharmos com as restrições a) e b) e com os multiplicadores lagrangeanos (variáveis duais) e colocando-os na função objetivo, obtém-se a formulação dual do P.C.V.. Este é o procedimento de relaxação lagrangeana estudada por Geoffrion.

II.6. PROBLEMA DUAL

Maximizar $\theta(u,v)$

$$\theta(u,v) = \text{mínimo}\{\sum\sum c_{ij}\, x_{ij} + \sum_i u_i \left(\sum_j x_{ij} - 1\right) + \sum_j v_j \left(\sum_i x_{ij} - 1\right)\}$$

ou

$$\theta(u,v) = -\sum_i u_i - \sum_j v_j + \text{mínimo}_{x \in X}\{\sum_i \sum_j (c_{ij} + u_i + v_j) x_{ij}\}$$

u,v não restritos.

Avaliar $\theta(u,v)$ para valores fixos de u e v - é um trabalho relativamente simples pois o problema se resume em encontrar uma árvore-1 mínima, com o custo c_{ij} do arco (i,j) substituído por $(c_{ij} + u_i + v_j)$.

As restrições (a) e (b) forçam automaticamente um ciclo a ser orientado, portanto não há a preocupação a respeito da orientação dos arcos, na procura da árvore-1 mínima.

Interpretação de u e v - podem ser vistos como penalidades por violar as orientações próprias dos arcos. Quando estiver calculando a árvore-1 mínima e muitos arcos "deixarem" o nó i, tenta-se "penalizar" esta deficiência na próxima iteração fazendo-se crescer os u_i's. Também do mesmo modo, se muitos arcos "entram" no nó j, tenta-se penalizar esta deficiência fazendo-se crescer os v_j's.

II.7. DUAL DO P.C.V. SIMÉTRICO

No caso simétrico, quando o vetor x, representa uma árvore-1 e como a orientação dos arcos não é importante, não será necessário que o número de arcos deixando o nó i seja 1 e o número de arcos entrando no nó i seja 1, é suficiente dizer que o número de arcos incidentes no nó i é 2. Há uma simplificação na formulação do primal e do dual do P.C.V..

O P.C.V. é reformulado:

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

$$\text{sujeito a: } \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} + \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ji} = 2 \quad \text{para } i=1,2,\ldots,n$$

$$x \in X$$

onde X é o conjunto das árvores-1.

O problema dual é dado abaixo, com apenas um conjunto de variáveis duais, u.

Maximizar $\theta(u)$

$$\text{sujeito a: } \theta(u) = -2\sum_{i=1}^{n} u_i + \min_{x\in X} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} (c_{ij}+u_i+u_j)x_{ij}$$

C A P Í T U L O III

III.1. ALGUNS MÉTODOS EXATOS

Existem vários métodos exatos para resolver o P.C.V. (problema do caixeiro viajante), isto é, métodos que garantem que a solução encontrada é a solução ótima.

Entre eles são citados aqui o método de Dantzig, o método de Branch-and-Bound de Little e de Bellmore e o método de Held-Karp.

III.2. MÉTODO DE DANTZIG, FULKERSON E JOHNSON

Uma das dificuldades em encontrar uma rota ótima resolvendo como um problema de programação inteira o problema de Alocação e mais as restrições para evitar as subrotas é o número muito grande de restrições e variáveis.

Porém Dantzig, Fulkerson e Johnson [7], encontraram soluções ótimas para problemas com até 49 cidades, usando quatro estratégias adicionais:

1a.) só trabalham com rotas não-orientadas, o que simplifica suas caracterizações quando n é grande e reduz o tempo de cálculo.

2a.) não caracterizam as rotas pelo conjunto completo das restrições lineares e adicionam novas restrições somente quando é necessário bloquear subrotas e tal que assegure que alguma rota satisfaça $\sum \sum c_{ij} x_{ij}$ e também para excluir soluções envolvendo x_{ij}'s fracionários.

3a.) nos exemplos dados com até 49 cidades, o cálculo foi feito sem o uso do computador e em parte isto é possível porque usam um simbolismo simples que permite a representação direta das relações algébricas a partir de um mapa das cidades em questão. Assim o processo iterativo é mais rápido e é fácil de acompanhá-lo nos seus passos e sugere novas restrições lineares que não seriam obtidas por outros métodos menos visuais.

4a.) uma vez obtida uma rota que esteja perto do ótimo, um procedimento combinatório é feito, usando o mapa e listando as possibilidades que ainda não foram eliminadas pelas condições impostas no problema.

O método de Dantzig, Fulkerson e Johnson é ilustrado no seguinte exemplo:

Considerando-se um mapa de 5 cidades, em forma de um pentágono regular, de lados unitários e portanto as diagonais de comprimento $1/2\ (\sqrt{5} + 1) = 1,7$

$$D(x) = \sum_{i>j} c_{ij} x_{ij} - \sum_{i=1}^{5} \Pi_i \left(\sum_{j=1}^{5} x_{ij} - 2 \right)$$

$$= \sum_{\substack{i=1,5 \\ i>j}} c_{ij} x_{ij} - \sum_{i=1}^{5} \Pi_i \left(\sum_{j=1}^{5} x_{ij} \right) + 2 \sum_{i=1}^{5} \Pi_i$$

$$= \sum_{\substack{i=1 \\ i>j}}^{5} c_{ij} x_{ij} - \sum_{\substack{i>j \\ j=1}}^{5} (\Pi_i + \Pi_j) x_{ij} + 2 \sum_{i=1}^{5} \Pi_i$$

$$= - \sum_{\substack{i=1 \\ i>j}}^{5} (\Pi_i + \Pi_j - c_{ij}) x_{ij} + 2 \sum_{i=1}^{5} \Pi_i$$

Chamando de $\delta_{ij} = \Pi_i + \Pi_j - c_{ij}$, temos (3)

$$D(x) = - \sum_{\substack{i=1 \\ i>j}}^{5} \delta_{ij} x_{ij} + 2 \sum_{i=1}^{5} \Pi_i \qquad (3)$$

Determinação dos Π_i's

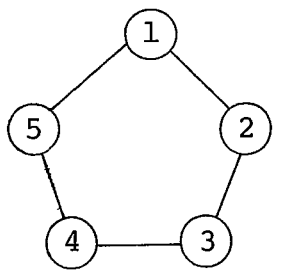
$\delta_{ij} = 0$ para $x_{ij} = 1$

$\delta_{21} = \Pi_2 + \Pi_1 - 1 = 0$ $\quad$ $\delta_{32} = \Pi_3 + \Pi_2 - 1 = 0$

$\delta_{54} = \Pi_5 + \Pi_4 - 1 = 0$ $\quad$ $\delta_{51} = \Pi_5 + \Pi_1 - 1 = 0$

Resolvendo o sistema,

$\Pi_1 = \Pi_2 = \Pi_3 = \Pi_4 = \Pi_5 = 1/2$



A matriz de distância $C = \lceil c_{ij} \rceil$

1	∞				
2	1	∞			
3	1,7	1	∞		
4	1,7	1,7	1	∞	
5	1	1,7	1,7	1	∞

O problema é:

$$\text{Minimizar} \quad C(x) = \sum_{i>j} c_{ij}\, x_{ij} \qquad i = 1,\ldots,5 \qquad (1)$$

$$\text{sujeito a:} \quad \sum_{j=1} x_{ij} = 2 \qquad \forall i \neq j \qquad (2)$$

$$x_{ij} = x_{ij}$$

$$x_{ij} = \begin{cases} 1 & \text{se o par } (i,j) \text{ pertence a rota} \\ 0 & \text{caso contrário} \end{cases}$$

Resolução:

Iniciando com a rota viável (1,2,3,4,5), então $\bar{x}_{12} = \bar{x}_{23} = \bar{x}_{34} = \bar{x}_{45} = 1$.

O comprimento da rota, quando $x = \bar{x}$ em (1) é $c(\bar{x}) = 5$.

Multiplicando cada restrição (2) por Π_i, um parâmetro a ser determinado e subtraindo de (1),

$$\text{para cada } i \neq j, \ \Pi_i \ \left(\sum_{j=1}^{5} x_{ij} - 2 \right) = 0 \ (x_{ij} = x_{ji})$$

Fazendo-se $x_{ij} = \bar{x}_{ij}$ em (3), $\bar{x}_{ij}\ \delta_{ij} = 0$

$\forall(i,j)$

$$D(\bar{x}) = 2 \sum_{i=1}^{5} \Pi_i = 5$$

Subtraindo $D(\bar{x})$ de $D(x)$:

$$D(x) - D(\bar{x}) = - \sum_{\substack{i=1 \\ i>j}}^{5} \delta_{ij}\ x_{ij} \qquad (4)$$

Nas diagonais do pentágono, tem-se:

$$\delta_{31} = \Pi_3 + \Pi_1 - c_{31} = -0,7$$

$$\delta_{41} = \Pi_4 + \Pi_1 - c_{41} = -0,7$$

$$\delta_{42} = \Pi_4 + \Pi_2 - c_{42} = -0,7$$

$$\delta_{52} = \Pi_5 + \Pi_2 - c_{52} = -0,7$$

$$\delta_{53} = \Pi_5 + \Pi_3 - c_{53} = -0,7$$

e para todo par (i,j), $\delta_{ij} \leq 0$, então a diferença (4) é sempre maior ou igual a zero.

$$D(x) - D(\bar{x}) \geq 0$$

Qualquer outra solução x, satisfazendo a relação (2) não vai ser melhor do que a solução inicial $\bar{x}$ por

tanto a solução ótima.

III.3. MÉTODO DE "BRANCH-AND-BOUND"

III.3.1. Método de Little, Murty, Sweeney e Kaul

O procedimento básico do método a Little [9] consiste no seguinte: particionar o conjunto de todas as rotas viáveis com n cidades em um número crescente de subconjuntos cada vez menores, calculando para cada um deles um limite inferior no custo das rotas. Esses limites é que determinam o particionamento dos subconjuntos.

Uma rota ótima é determinada quando se encontra um desses subconjuntos com uma única rota, cujo custo total é menor ou igual do que qualquer limite inferior dos outros subconjuntos.

Os subconjuntos de rotas são representados convenientemente como os nós de uma árvore e o processo de particionamento como uma ramificação dessa árvore, daí o nome "branch-and-bound".

Em geral este método trabalha bem com problemas não simétricos, sendo possível também aplicá-lo a problemas simétricos mas com algumas modificações importantes.

Representação:

$C = [c_{ij}]$ matriz de custo

t = rota

$z(t) = \sum_{(i,j)\varepsilon t} c_{ij}$ custo da rota

$X, Y, \bar{Y}$ = representam nós de uma árvore

$\omega(X)$ = limite inferior em todas as rotas de um conjunto X e portanto $z(t) > \omega(x)$, $t \varepsilon X$

z_o = limite superior no custo de todas as rotas.

III.4. UM PROCESSO DE REDUÇÃO

Para se determinar os limites inferiores nos subconjuntos é necessário o conceito de redução.

Se por exemplo, uma constante h é subtraída de uma linha ou coluna da matriz C, então qualquer rota sob esta nova matriz vai ser a mesma sob a matriz anterior, menos h, por que uma rota tem somente um elemento em cada linha e coluna.

Um processo de redução é quando subtrai-se o menor elemento de uma linha (coluna) de uma matriz, dos demais elementos desta linha (coluna).

Uma matriz reduzida - é uma matriz com pelo menos um elemento zero em cada linha e coluna e os demais elemen

tos positivos.

Se z(t) é o custo de uma rota t antes da redução e $z_1(t)$ depois e h é a soma das constantes reduzidas, então: $z(t) = z_1(t) + h$ e h constitui um limite inferior no conjunto de todas as rotas, sob a matriz original.

Exemplo:

i \ j	1	2	3	4	
1	∞	18	5	2	2
2	1	∞	9	3	1
3	16	18	∞	18	16
4	23	9	7	∞	7

Reduzindo linhas

i \ j	1	2	3	4	
1	∞	16	3	0	2
2	0	∞	8	2	1
3	0	2	∞	2	16
4	16	2	0	∞	7
	0	2	0	0	

Reduzindo Colunas

i \ j	1	2	3	4
1	∞	14	3	0
2	0	∞	8	2
3	0	0	∞	2
4	16	0	0	∞

Matriz Reduzida

h = 28

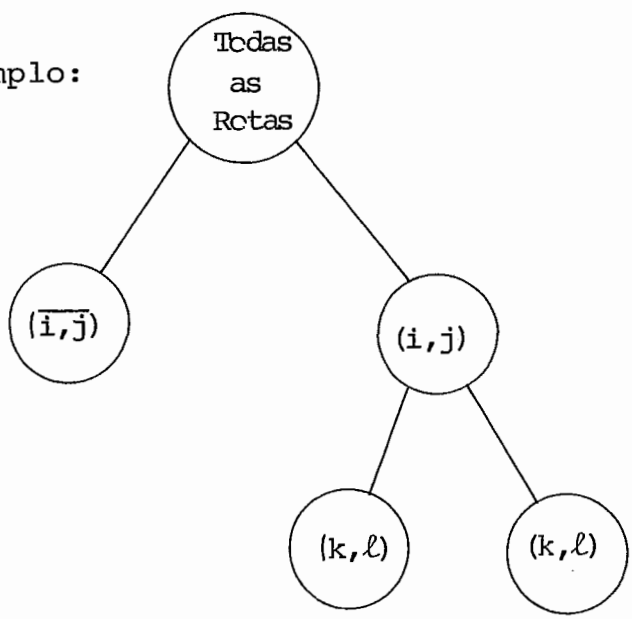
III.5. PROCESSO DE RAMIFICAÇÃO

O particionamento do conjunto de rotas em subconjuntos é representado como ramificação de uma árvore.

O nó inicial representa o conjunto de todas as rotas. O nó contendo o par (i,j) gerado a partir do nó inicial representa o conjunto de todas as rotas que incluem o par de cidades (i,j) enquanto que o nó $(\overline{i,j})$ representa o conjunto de todas as rotas que não incluem o par (i,j).

Continuando a ramificação a partir do nó (i,j), temos mais dois nós; um representando o conjunto de todas as rotas que incluem um novo par de cidade (k,ℓ), dentro do conjunto de todas as rotas que incluem o par (i,j), enquanto o outro, representa o conjunto não excluindo o par de cidades (k, ℓ).

Exemplo:



Algorítmo:

Passo 1: Fazer $z_o = \infty$, z_o é o limite superior no custo de todas as rotas.

- reduzir C.
- X = 1 no inicial representando todas as rotas
- $\omega(x)$ = soma das constantes reduzidas de C

Passo 2: Selecionar o par de cidades (k, ℓ), que vai determinar a próxima ramificação, pelo seguinte:

$\theta\ (k, \ell)$ = máximo $\theta\ (i,j)$

onde $\theta\ (i,j)$ é determinado nos pares onde $c_{ij} = 0$ da matriz reduzida C, como a soma do menor custo na linha i, excluindo c_{ij}, com o menor custo na coluna, excluindo c_{ij}.

Passo 3: Fazer uma ramificação do nó X para $\overline{Y} = (k, \ell)$ e rotular $\overline{Y}$ por $W(\overline{Y}) = w(X) + \theta\ (k, \ell)$.

- Fazer outra ramificação de X para Y onde $Y = (k, \ell)$ e eliminar a linha k e a coluna na matriz C.
- Encontrar p = cidade de partida
  m = cidade de chegada
  de um caminho gerado pelos pares de cidades escolhidas para estarem nas rotas, até Y e fazer $c_{mp} = \infty$.
- Reduzir C
- Rotular Y por $w(X) = w(X)$ + soma das constantes reduzidas
- Verificar se C é uma matriz 2 x 2
  Caso positivo, ir para o Passo 7.

Passo 4: Selecionar o próximo X, o nó com menor valor w(X), para a próxima ramificação.
Verificar se $z_o \leq w(X)$ - então pare !
Caso contrário, continue.

Passo 5: Verificar se a ramificação continua a direita, isto é, se X = Y do Passo 3, então volte ao Passo 2.

Passo 6: - Tomando a matriz C - original, ler os pares (i,j) escolhidos para estarem nas rotas até o nó X e calcular $g = \sum c_{ij}$ desses pares.

- Para cada desses pares (i,j) eliminar a linha i e a coluna j e para cada caminho contendo o par (i,j), encontre a cidade p e a cidade m e faça $c_{mp} = \infty$.

- Para cada par (k, ℓ) que não pode ocorrer nas rotas de X, faça $c_{k\ell} = \infty$.

- Reduzir C e rotule X com w(X) = g + soma das constantes reduzidas.

- Voltar ao Passo 2.

Passo 7: Verificar se $w(Y) \geq z_o$, então volte ao Passo 4.

Senão, faça $z_o = w(Y)$ e volte ao passo 4.

Observações: O processo de redução é um caminho eficiente de construir limites inferiores e também para escolher os pares de cidades a serem colocados na rota.

- A ramificação é feita sempre de modo a maximizar o limite inferior para o nó (k, ℓ), de modo que rotas não ótimas sejam eliminadas mais rapidamente.

Exemplo

Dada a matriz de distância C

	1	2	3	4	5
1	∞	10	25	25	10
2	1	∞	10	15	2
3	8	9	∞	20	10
4	14	10	24	∞	15
5	10	8	25	27	∞

Q - 1

	1	2	3	4	5
1	∞	0	6	3	0
2	0	∞	0	2	1
3	0	1	∞	0	2
4	4	0	5	∞	5
5	2	0	8	7	∞

Q - 2

Passo 1:

- $z_o = \infty$
- Redução de C (quadro 2)
- X = 1
- $\overset{\circ}{w}(X) = 58.$

Passo 2:

Determinar $\theta(k,\ell) = \max \theta(i,j)$

$\theta(1,2) = 0 \quad \theta(3,4) = 2 \quad \theta(1,5) = 0 \quad \theta(2,1) = 0$

$\theta(2,3) = 5 \quad \theta(3,1) = 0 \quad \theta(4,2) = 4 \quad \theta(5,2) = 2$

$\theta(k,\ell) = 5 \quad e \quad (k,\ell) = (2,3)$

Passo 3: Ramificação

- $\overline{Y} = (\overline{2,3})$

  $w(\overline{y}) = 63$
- Y = (2,3)

  Eliminar a linha 2 e a coluna 3 em C reduzida.

	1	2	4	5
1	∞	0	3	0
3	0	1	0	2
4	4	0	∞	5
5	2	0	7	∞

Q - 3

	1	2	4	5
1	∞	0	3	0
3	0	∞	0	2
4	4	0	∞	5
5	2	0	7	∞

Q - 4

- p = 2
  m = 3 $\quad C_{32} = \infty$ em Q-4
- Reduzir C
- w(Y) = 58

Passo 4: X = (2,3) e w(X) = 58 < z_o

Passo 5: X = Y no Passo 3.

Passo 2: Determinação do próximo $\theta(k,\ell)$

$\theta(1,2) = 0 \quad \theta(1,4) = 0 \quad \theta(1,5) = 2 \quad \theta(3,1) = 3$

$\theta(3,4) = 0 \quad \theta(4,2) = 4 \quad \theta(5,2) = 2$

$\theta(k,\ell) = 4 \quad$ e $\quad (k,\ell) = (4,2)$

Passo 3: - $\overline{Y} = (\overline{4,2})$ e $w(\overline{Y}) = 62$

- Y = (4,2) e eliminação da linha 4 e coluna 2

	1	4	5
1	∞	3	0
3	0	0	2
5	2	7	∞

Q - 5

	1	4	5
1	∞	3	0
3	0	∞	2
5	2	7	∞

Q - 6

	1	4	5
1	∞	0	0
3	0	∞	2
5	0	2	∞

Q - 7

- Um caminho: (4,2)(2,3) e portanto $c_{34} = \infty$ em Q-6
- Reduzir C do Q-6
- w(Y) = 63.

Passo 4: Somando X, o de menor valor

$X = (\overline{4,2})$ e $z_o > w(x)$

Passo 5: $X \neq Y$ do Passo 3.

Passo 6: - Somar a matriz C original
- $g = c_{23} = 10$
- Eliminar a linha 2 e a coluna 3
- $c_{42} = \infty$

	1	2	4	5
1	∞	10	25	10
3	8	9	20	10
4	14	∞	∞	15
5	10	8	27	∞

Q - 8

	1	2	4	5
1	∞	0	3	0
3	0	1	0	2
4	0	∞	∞	1
5	2	0	7	∞

Q - 9

	1	2	5
1	∞	0	0
4	0	∞	1
5	2	0	7

Q - 10

- Reduzir C do Q-8 em Q-9
- w(X) = 62

Passo 2: Determinação de $\theta(k,\ell)$

$\theta(1,2) = 0 \quad \theta(1,5) = 1 \quad \theta(3,1) = 0 \quad \theta(4,1) = 1$

$\theta(5,2) = 3 \quad \theta(3,4) = 3$

$\theta(k,\ell) = 3 \quad$ e $\quad (k,\ell) = (3,4)$

Passo 3: $Y = (\overline{3,4})$ e $w(\overline{Y}) = 65$

- $Y = (3,4)$
- Eliminar a linha 3 e a coluna 4 - Q -10
- Um caminho $(2,3)(3,4)$ e $c_{42} = \infty$
- Reduzir C
- $w(Y) = 62$

Passo 4: $X = (3,4)$ e $z_o > w(X)$

Passo 5: X = Y do Passo 3

Passo 2: Determinação de $\theta(k,\ell)$

$\theta(1,2) = 0$ $\theta(1,5) = 1$ $\theta(4,1) = 3$ $\theta(5,2) = 2$

$\theta(k,\ell) = 3$ e $(k, \ell) = (4,1)$

Passo 3: - $\overline{Y} = (\overline{4,1})$ e $w(\overline{Y}) = 65$

- $Y = (4,1)$
- Eliminar a linha 4 e a coluna 1 de Q-10.

	2	5
1	0	0
5	0	7

Q - 11

	2	5
1	∞	0
5	0	7

Q - 12

- Um caminho: $(2,3)(3,4)(4,1)$, então $c_{12} = \infty$
- Reduzir C
- $w(Y) = 62$
- C é uma matriz 2 x 2

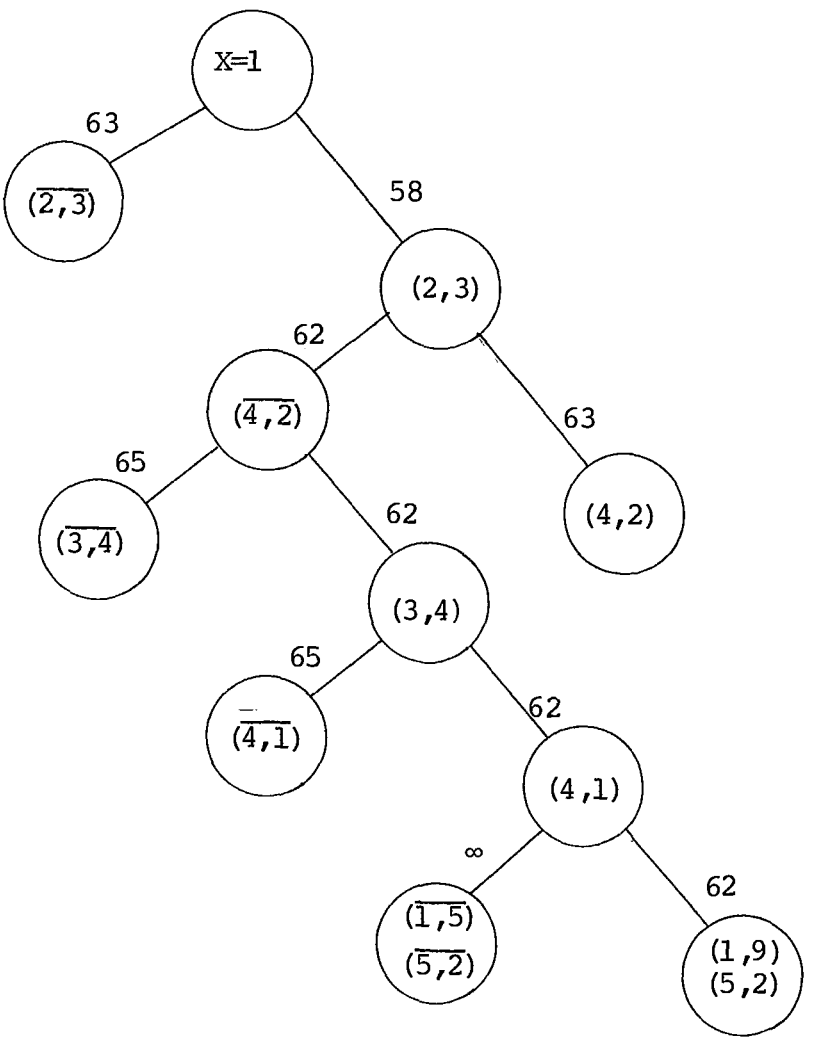
Passo 7: $w(X) < z_o$

$z_o = 62$

Passo 4: $X = (4,1)$

$w(X) = 62$

$z_o = w(X)$ - pare.



A solução ótima é: (2,3)(3,4)(4,1)(1,5)(5.2)

e $z_o = 62$.

III.6. MÉTODO DE BELLMORE E MALONE - OU MÉTODO DE ELIMINAÇÃO DE SUBROTAS

A restrição do tipo II, do Capítulo II, isto é, $\sum_{S}\sum_{\bar{S}} x_{ij} \geq 1$, onde $(S,\bar{S})$ é uma partição de inteiros, i= 1,...,n, acrescentada ao Problema de Alocação é bastante eficiente devido a sua rápida convergência e eliminando mais soluções viáveis.

Bellmore e Malone [2] encontraram uma técnica bastante eficaz para aplicar a restrição, sem se envolver em uma formulação inteira, associada com suas dificuldades computacionais em problemas assimétricos.

Seja a solução de um Problema de Alocação com uma subrota S de k cidades.

Se todas as cidades são particionadas em dois subconjuntos S e $\bar{S}$, então qualquer rota viável deve envolver pelo menos um arco ligando uma cidade de S a uma outra de $\bar{S}$, e de cada cidade t_i, i = 1,...,k pertencente a S não pode partir um arco para ligar a uma outra de S. Para que tal aconteça, forçamos o custo de cada $t_i \in S$ para qualquer outra cidade de S, ser arbitrariamente grande.

Então a solução S é "quebrada" em k subconjuntos onde cada um deles é caracterizado por cada cidade t_i, i = 1,...,k.

FORMALIZAÇÃO

Teorema 1: Dada uma solução viável ao Problema de Alocação:

$$\text{Min} \sum_i \sum_j c_{ij} x_{ij}$$

$$\text{sujeito a:} \sum_i x_{ij} = 1 \qquad \forall i = 1,\ldots,n$$

$$\sum_i x_{ij} = 1 \qquad \forall j = 1,\ldots,n$$

$$x_{ij} = 0 \text{ ou } 1 \quad \forall i \neq j$$

com uma subrota de comprimento k.

Se S contém as cidades da subrota e se S é particionado em k subconjuntos T_i, i = 1,...,k, onde cada T_i é caracterizado por cada arco saindo da i-ésima cidade de S indo diretamente para uma cidade $\overline{S}$, então a solução com a subrota será eliminada, mas não as rotas viáveis. (Demonstração, pag, 294 de [2]).

Teorema 2: Selecionar a subrota contendo o menor número de cidades para particionar o conjunto de solução, significa uma redução máxima no número de soluções que não correspondem a soluções viáveis. (Demonstração, pag, 294 de [2]).

Teorema 3: As soluções ótimas dos k subconjuntos originários da quebra de uma subrota tem os valores maiores do que o va

lor da solução inicial. O valor mínimo sobre todos os subconjuntos constitui um limite inferior em todas as rotas viáveis. (Demonstração, pag, 295 de [2]).

III.7. UM ALGORITMO PARA RESOLVER O PROBLEMA DE ALOCAÇÃO

Existem muitos algoritmos para resolver o Problema de Alocação e o mais eficiente é o Algoritmo de Hitchcock.

Considerando os problemas primal e dual do Problema de Alocação contínuo.

Primal	Dual
$\text{Min} \quad \sum_i \sum_j c_{ij} x_{ij}$	$\text{Max} \quad -\sum_i \alpha_i + \sum_j \beta_j$
sujeito a: $\sum_i x_{ij} \leq 1 \quad \forall i$	sujeito a: $\beta_j - \alpha_i \leq c_{ij}$
$\sum x_{ij} \geq 1 \quad \forall j$	$\alpha_i, \beta_j \geq 0$
$x_{ij} \geq 0$	

As condições de otimalidade são $x_{ij}(\beta_j - \alpha_i - c_{ij}) = 0$ para todo i e j.

A solução técnica do Algoritmo de Hitchcock

consiste em manter a viabilidade do dual, ajustando as variáveis primais de acordo com as condições de otimalidade primal -dual.

INICIO DO ALGORITMO

$\beta_j = \min_i c_{ij}$ para $\forall j$

$\alpha_i = 0$ para $\forall i$

$x_{ij} = 0$ para $\forall (i,j)$

Um par de (i,j) é ADMISSÍVEL se $\alpha_i - \beta_j + c_{ij} = 0$ e as mudanças nas variáveis primais só podem ocorrer nos pares admissíveis.

Passo 1: ROTULAÇÃO

Rotular cada linha i, para o qual $\sum_j x_{ij} = 0$

Das linhas rotuladas, rotule cada coluna j para o qual o par (i,j) é admissível. De cada coluna rotulada j, rotule cada linha i, para o qual $x_{ij} = 1$. Se uma coluna é rotulada tal que $\sum_i x_{ij} = 0$, vá para o Passo 2. Se não for possível mais rotular, vá para o Passo 3, senão continue o procedimento acima.

Passo 2: MUDANÇA DE FLUXO

Seguindo o caminho deixado pelo Passo 1, faça $x_{ij} = 1$ para cada passo rotulado da linha para coluna e $x_{ij} = 0$ para cada passo rotulado da coluna para a li

nha. Se $\sum_j x_{ij} = 1$, para todo i, termine. Caso contrário, volte ao Passo 1.

Passo 3: MUDANÇA NAS VARÍAVEIS DUAIS

Seja S - conjunto dos índices das linhas rotuladas

T - conjunto dos índices das colunas rotuladas

Calcular $\delta = \min_{S\overline{T}} (c_{ij} + \alpha_i - \beta_j)$ $i \in S$

$j \in \overline{T}$ (complementar de T)

e

$$\alpha_i = \begin{cases} \alpha_i & \text{se } i \in S \\ \alpha_i + \delta & \text{se } i \in \overline{S} \end{cases} \qquad \beta_j = \begin{cases} \beta_j & \text{se } j \in T \\ \beta_j + \delta & \text{se } j \in \overline{T} \end{cases}$$

Voltar ao Passo 1.

III. 8. PROGRAMA PARAMÉTRICO E O ALGORITMO DE HITCHCOCK

O algoritmo para o P.C.V. a ser desenvolvido vai depender de encontrar soluções do Problema de Alocação, com uma pequena modificação na função objetivo.

Dada a solução do Problema de Alocação x_o, α_o, β_o. Se um arco específico (i,j) precisa ser eliminado por pertencer a uma subrota, então faça $x_{ij} = 0$ e $c_{ij} = \infty$. As variáveis duais α_o, β_o continuam viáveis.

As condições de otimalidade são satisfeitas, menos aquela em que $\sum_i x_{ik} < 1$ para $k = j$.

O algoritmo pode ser continuado a partir deste ponto e a nova solução ótima será encontrada na primeira incidência à rotulação da coluna j. Com isto, o número de mudanças de fluxo são reduzidos, a partir da solução inicial.

O MÉTODO

Passo 1: Resolver o Problema de Alocação associado. Se sua solução for uma rota, então pare - a solução é ótima para o P.C.V.

Se em sua solução existirem subrotas, vá para o Passo 2.

Passo 2: Selecionar o conjunto de solução com o menor valor (da função objetivo) e tomar a subrota com o menor número de cidades, k.

Particionar o conjunto de solução em k subconjuntos, como no Teorema 2, e resolver o Problema de Alocação parametricamente, para cada desses subconjuntos.

Passo 3: As soluções sem subrotas são comparadas com a melhor solução existente, e se custar menos, fazer Z igual ao menor valor (da função objetivo) e a solução correspondente torna-se a melhor solução.

Encontrar entre as soluções restantes, o conjunto de solução com valor mínimo e compare com Z.

Se for maior que Z, então a melhor solução é ótima .

Senão, considerar todas as soluções com subrotas, cujos valores sejam menores que Z e volte ao Passo 2.

O particionamento de uma subrota em outros problemas pode ser visto como um processo de ramificação de uma árvore.

O nó inicial representa a solução do Problema de Alocação resolvido inicialmente. Se a solução tem subrotas, com a subrota com maior número de cidades k, fazer ramificações para cada problema de alocação paramétrico e o próximo nó a ser ramificado é aquele que tem o menor valor , com subrotas.

Exemplo

Dada a matriz de distância

	1	2	3	4	5
1	∞	10	25	25	10
2	1	∞	10	15	2
3	8	9	∞	20	10
4	14	10	24	∞	15
5	10	8	25	27	∞

α_i \ β_j	1	8	10	15	2	
0						1(-,1)
0	◯		◯	◯	(1)	0
0						1(-,1)
0						1(-,1)
0		(1)				0
	1	0	1	1	0	

Q - 1

Passo 1: Resolver o Problema de Alocação associado.

Pelo Algoritmo de Hitchcock:

Solução Inicial

$\beta_j = \min_i [c_{ij}]$

$\alpha_i = 0 \quad \forall i = 1,\ldots,n$

$x_{ij} = 0 \ \forall i,j$

pares admissíveis (i,j) onde $\alpha_i - \beta_j + c_{ij} = 0$

Passo 1: ROTULAÇÃO

Passo 2: MUDANÇA DE FLUXO

Passo 1: ROTULAÇÃO

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS (Q-1)

$S = \{1,3,4\}$

$T = \phi \quad \overline{T} = \{1,2,3,4,5\}$

$\delta = \min_{S\overline{T}} (c_{ij} + \alpha_i - \beta_j)$

$\delta = 1$ e (3,2) é o novo para admissível - Q-2

$$\alpha_i = \begin{cases} \alpha_i & \text{se } i \in S \\ \alpha_i + \delta & \text{se } i \in \overline{S} \end{cases} \qquad \beta_j = \begin{cases} \beta_j & \text{se } j \in T \\ \beta_j + \delta & \text{se } j \in \overline{T} \end{cases}$$

α_i \ β_j	2	9	11	16	3	
0						1 (-)
1	◯		◯	◯	(1)	0
0		◯				1 (-)
0						1 (-)
1		(1)				0 (2)
	1	0	1	1	0	

(3)

Q - 2

α_i \ β_i	6	9	15	20	7	
0						1 (-)
5	◯		◯	◯	(1)	0
0		◯		(1$^+$)		1$^-$ (-)
0						1 (-)
1		(1)				0 (2,1)
	1	0	1	1$^-$	0	

(3,1)

Q - 3

Passo 1: ROTULAÇÃO

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

$S = \{1,3,4,5\}$

$T = \{2\}$ e $\overline{T} = \{1,3,4,5\}$

$\delta = 4$ e o novo par admissível é (3,4) em Q-3

Passo 2: MUDANÇA DE FLUXO, DESCARTAR TODOS OS RÓTULOS

Passo 1: ROTULAÇÃO

α_i \ β_j	6	9	15	20	7	
0						1 (-)
5	◯		◯	◯	(1)	0
0		◯		(1)		0
0						1 (-)
1		(1)				0
	1	0	1	0	0	

Q - 4

α_i \ β_j	7	10	16	21	8	
0		◯				1 (-)
6	◯		◯	◯	(1)	0
1		◯				0
0		◯				1 (-)
		(1)				0 (2)
	1	0	1	0	0	

(4)

Q - 5

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

$S = \{1,4\}$

$T = \phi$ $\quad \overline{T} = \{1,2,3,4,5\}$

$\delta = 1$ e novos pares admissíveis (1,2) e (4,2) em Q-5

Passo 1:

Passo 3:

$S = \{1,4,5\}$

$T = \{\ \}$ e $\overline{T} = \{1,3,4,5\}$

$\delta = 2$ e o novo par admissível é (1,5) em Q-6

α_i \ β_j	9	10	18	23	10	
0		◯			(1^+)	1^- (-)
8	◯		(1^+)	◯	(1^-)	0 (5)
3		◯		(1)		0
0		◯				1 (-)
2		(1)				0 (2)
	1	0	1^-	0	0	

(4) (2) (1)

Q - 6

α_i \ β_j	9	10	18	23	10	
0		◯			(1)	0
8	◯		(1)	◯	◯	0
3		◯		(1)		0
0		◯				1 (-)
2		(1)				0 (2)
	1	0	0	0	0	

(4)

Q - 7

Passo 2: MUDAR O FLUXO E DESCARTAR TODOS OS RÓTULOS

Passo 1: ROTULAÇÃO - Q-7

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

$S = \{4,5\}$

$T = \{2\}$ $\quad \overline{T} = \{1,3,4,5\}$

$\delta = 3$ e o novo par admissível é (5,1) em Q-8.

α_i \ β_j	12	10	21	26	13	
3		◯			1	0
11	◯		1	◯	◯	0
6		◯		1		0
0		1^+				5 (-)
2	1^+	1^-				0 (2)
	1^-	0 (4)	0	0	0	

Q - 8

α_i \ β_j	12	10	21	26	13	
3					1	0
11			1			0
6				1		0
0		1				0
2	1					0
	0	0	0	0	0	

Q - 9

Passo 2: MUDANÇA NO FLUXO

TÉRMINO DO ALGORITMO DE HITCHCOCK

Solução: (1,5) (5,1)

(2,3) (3,4) (4,2)

Valor: 60

Passo 2: Somando a subrota com menor número de cidades (1,5) e (5,1) resolver os dois problemas de alocação para métrica;

1º) com $c_{15} = \infty$

2º) com $c_{51} = \infty$

1º) Resolvendo o primeiro problema a partir do quadro 5, pelo Algoritmo de Hitchcock

α_i \ β_j	7	10	16	21	8	
0		◯				1 (-)
6	◯		◯	◯	(1)	0
1		◯		(1)		0
0		◯				1 (-)
2		(1)				0 (2)
	1	0	1	0	0	

Q - 5

α_i \ β_j	11	10	20	25	12	
0		◯		◯		1 (-)
10	◯		◯	◯	(1)	0
5		◯		(1)		0
0		◯				1 (-)
2		(1)				0
	1	0	1	0	0	
		(4)		(1)		

Q - 6

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

$S = \{1,4,5\}$

$T = \{2\}$ $\quad \overline{T} = \{1,3,4,5\}$

$\delta = 4$ e o novo par admissível é (1,4) em Q-6.

Passo 1: ROTULAÇÃO

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

$S = \{1,3,4,5\}$

$T = \{2,4\}$ $\quad \overline{T} = \{1,3,5\}$

$\delta = 1$ e o novo par admissível é (5,1) em Q-7.

α_i \ β_j	12	10	26	25	13	
0		◯		◯		1 (-)
11	◯		◯	◯	(1)	0
6		◯		(1)		0 (4)
0		(1$^+$)				1$^-$ (-)
2	(1$^+$)	(1$^-$)				0 (2)
	1$^-$	0	1	0	0	
		(4)		(1)		

Q - 7

α_i \ β_j	12	10	26	25	13	
0		◯		◯		1 (-)
11	◯		◯	◯	(1)	0
6		◯		(1)		0 (4)
0		(1)				0 (2)
2	(1)	◯				0
	0	0	1	0	1	
		(3)		(1)		

Q - 8

Passo 2: MUDANÇA DE FLUXO

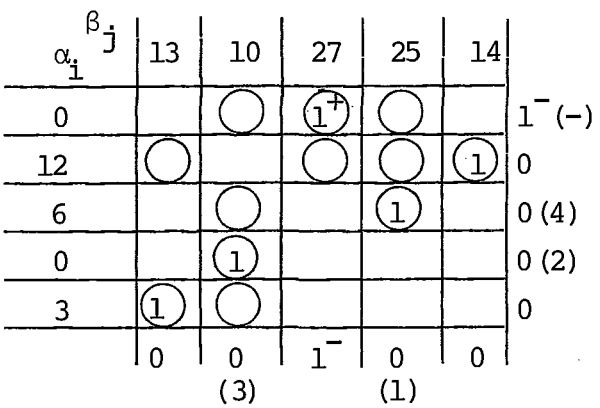
Passo 1: ROTULAÇÃO

Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

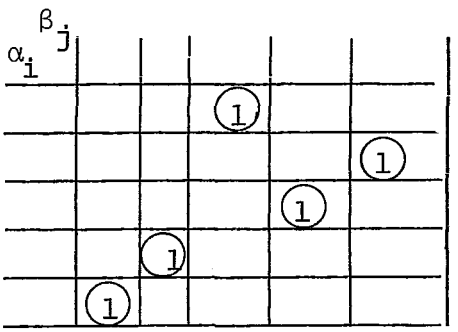
S = {1,3,4}

T = {2,4} $\bar{T}$ = {1,3,5}

δ = 1 e o novo par admissível é (1,3) em Q-9.



Q - 9

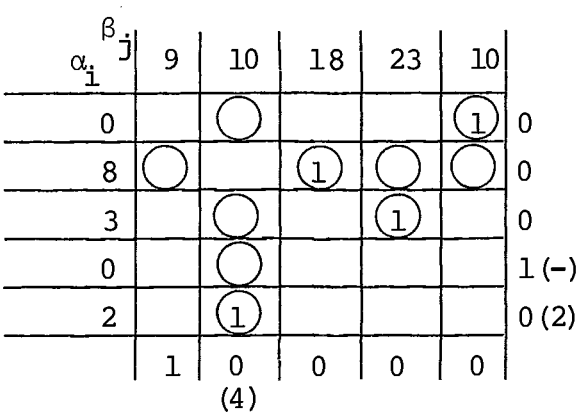


Q - 10

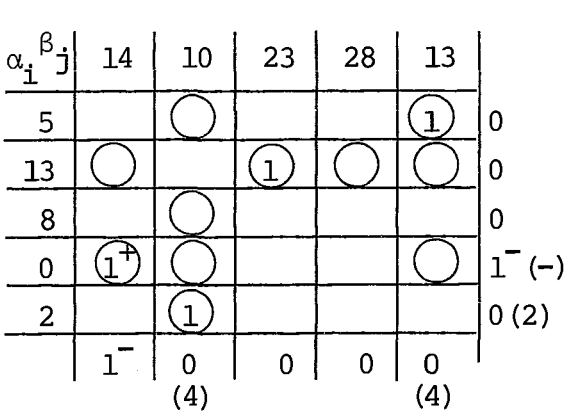
Solução: (1,3)(3,4)(4,2)(2,5)(5,1)

Valor: 67

2º) Problema - a partir do Q - 7



Q - 7



Q - 8

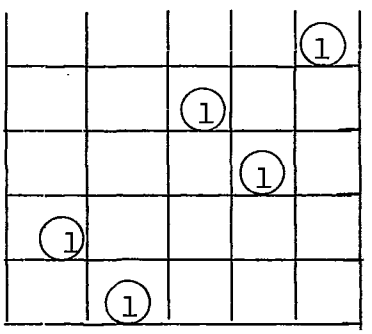
Passo 3: MUDANÇA DAS VARIÁVEIS DUAIS

S = {4,5}

T = {2} $\overline{T}$ = {1,3,4,5}

δ = 5 e os novos pares admissíveis (4,1) e (4,5) em Q - 8.

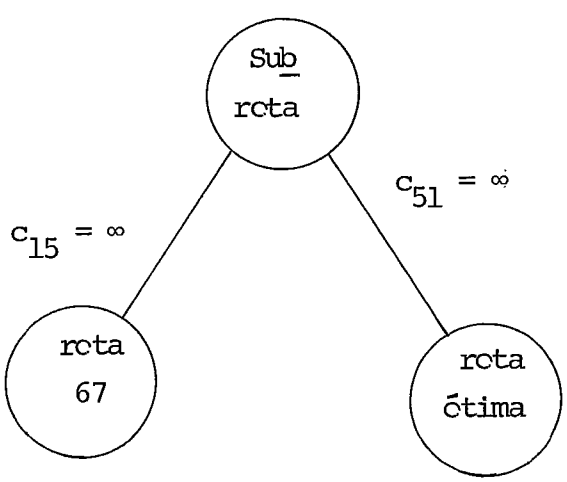
Passo 2: MUDANÇA DE FLUXO



Solução: (1,5)(5,2)(2,3)(3,4)(4,1)

Valor: 62

Passo 3: A melhor solução entre os dois problemas é: (1,5)(5,2)(2,3)(3,4)(4,1) com valor 62, que é portanto a solução ótima.



III.9. MÉTODOS DE HELD E KARP

Um novo enfoque para a resolução do problema do caixeiro viajante (P.C.V.) é apresentado por Held e Karp em [10] e [11] onde é explorada a relação entre o P.C.V. simétrico e o problema da árvore geradora mínima.

DEFINIÇÕES

K_n - grafo completo não orientado no conjunto de vértices {1,2,...,n}

(i,j) - arco ligando os vetores i e j

c_{ij} - o custo associado a (i,j)

árvore - um grafo conexo sem ciclos

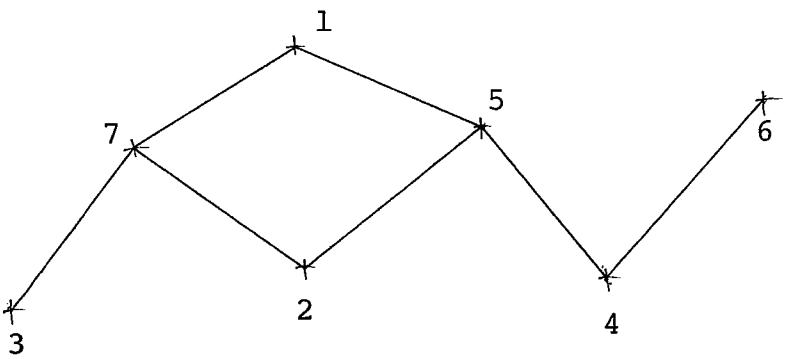
árvore geradora de um grafo G - é um subgrafo de G, tal que, todo nó de G está na árvore.

O P.C.V. procura a rota de custo mínimo enquanto que o problema da árvore geradora mínima procura uma árvore de custo mínimo no conjunto de vértices {1,2,...,n}.

É definido ainda a árvore-1, que consiste de uma árvore no conjunto de vértices {2,3,...,n}, justamente com dois arcos distintos, no vértice 1.

Então a árvore-1 tem um único ciclo e este contém o vértice 1. Sendo o grau em cada vértice o número de arcos incidentes nele, na árvore-1 o vértice 1 tem sempre grau 2.

Exemplo de uma árvore-1 num grafo de 7 nós:



A árvore-1 de custo mínimo pode ser encontrada construíndo-se uma árvore geradora mínima no conjunto de vértices {2,...,n} e depois associando-se dois arcos de menores custos ao vértice 1.

Toda rota de um P.C.V. é uma árvore-1 e uma árvore-1 é uma rota se e somente se cada um de seus vértices tem grau 2.

Teorema 1: Seja $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ um vetor do $\mathbb{R}^n$. Se C* é uma rota de custo mínimo em relação aos custos c_{ij}, então também é uma rota de custo mínimo em relação aos custos c_{ij} + $+ \Pi_i + \Pi_j$. (Demonstração, pag, 1140 de [10]).

Por outro lado a transformação de c_{ij} para $c_{ij} + \Pi_i + \Pi_j$ altera a identidade da árvore-1 mínima. Possivelmente, então uma estratégia para resolver o P.C.V. seja procurar um vetor Π, temos que, uma árvore-1 mínima em relação aos custos $c_{ij} + \Pi_i + \Pi_j$ seja uma rota.

Introduz-se uma função "gap" $f(\Pi)$ - igual a diferença do custo de uma rota com custo $c_{ij} + \Pi_i + \Pi_j$ com

o custo de uma árvore-1 mínima em relação aos mesmos custos.

O problema pode ser expresso com um problema de programação linear.

Sejam as árvores-1 de K_n, indexadas por 1 , 2,...,q com k um índice genérico.

Por definição:

$$f(\Pi) = w + 2 \sum_{i=1}^{n} \Pi_i - \min_k \left[ck + \sum_{i=1}^{n} \Pi_i \, d_{ik}\right]$$

onde:

w - custo da rota mínima em relação aos c_{ij}'s

c_k - custo da k-ésima árvore-1 em relação aos c_{ij}'s

d_{ik} - é o grau do vértice i, na k-ésima árvore-1

Ainda,

$$f(\Pi) = w - \min_k \left[ck + \sum_{i=1}^{n} \Pi_i \, (d_{ik} - 2)\right]$$

e determinar o mínimo de f (Π) em relação a Π, é equivalente a:

$$\text{maximizar}_{\Pi} \; w(\Pi)$$

$$(I) \qquad w(\Pi) = \min_k \left[ck + \sum_{i=1}^{n} \Pi_i \, v_i k\right] \quad k = 1,\ldots,q$$

$$v_i k = d_i k - 2$$

Este problema é equivalente ao problema de programação linear abaixo:

maximizar w

(I-a)

$$\text{sujeito a: } w \leq ck + \sum_{i=1}^{n} \Pi_i (v_i k) \quad k=1,2,\ldots,q$$

MÉTODOS DE RESOLUÇÃO

I) TÉCNICA DE GERAÇÃO DE COLUNAS

Tomando-se o dual do problema (I-a)

	w	Π_1	Π_2	.....	Π_n	
y_1	1	$-v_{11}$	$-v_{21}$	.....	$-v_{n1}$	c_1
y_2	1	$-v_{12}$	$-v_{22}$	.....	$-v_{n1}$	c_2
⋮	⋮					
y_q	1	$-v_{1q}$	$-v_{2q}$	.....	$-v_{nq}$	c_q
	1	0	0	.....	0	

$$\text{minimizar } \sum_{k=1}^{q} yk\ ck$$

II $$\text{sujeito a: } \sum_{k=1}^{q} yk = 1$$

$$\sum_{k=1}^{q} (-v_{ik})\ yk = 0 \quad \forall i=1,2,\ldots,n-1$$

onde: ck - é o outro da k-ésima árvore-1 em relação ao custo c_{ij}

yk - corresponde a uma árvore-1, A^k, no conjunto de vértices {1,2,...,n}

Aplicando no problema acima a técnica de geração de coluna, vai ser mostrado que a coluna a entrar na base é determinada pela árvore-1 mínima calculada.

Cada coluna da matriz de restrição tem a forma $(1, -v_2k, -v_3k, \ldots, v_{n-1}k)^T$, onde $v_{ik} = d_{ik} - 2$ e d_{ik} sendo o grau do vértice i na A^k.

A cada iteração do método simplex revisado, a variável que entra na base é determinada minimizando, sobre todos os k, a quantidade ck - zk, onde zk é o produto interno da coluna k por um vetor de custos marginais.

Seja o vetor de custos marginais $(\theta_1, \Pi_2, \Pi_3, \ldots, \Pi_{n-1})$ e a matriz de restrições:

$$
\begin{array}{c|cccc|c}
 & y_1 & y_2 & \cdots\cdots & y_q & 1 \\
\theta & 1 & 1 & \cdots\cdots & 1 & 0 \\
\Pi_2 & -v_{21} & -v_{22} & \cdots\cdots & -v_q & 0 \\
\vdots & \vdots & & & & \\
\Pi_{n-1} & -v_{n-1,1} & v_{n-1,2} & \cdots\cdots & -v_{n-1,q} & 0 \\
\hline
 & c_1 & c_2 & \cdots\cdots & c_q &
\end{array}
$$

O que se quer é encontrar uma árvore-1 A^k tal que $ck - \theta + \sum_{j=2}^{n-1} \Pi_j v_j k$ seja mínimo (se o mínimo é não negativo, então a solução básica presente é otima).

Nota-se que a cada arco (i,j) é associado , um custo $c_{ij} + \Pi_i + \Pi_j$ com $\Pi_i = \Pi_n = 0$, então o custo de A^k é:

$$ck + \sum_{i=2}^{n-1} \Pi_j d_j k = c_k + \sum_{j=2}^{n-1} \Pi_j (v_j k + 2) =$$

$$= ck + \sum_{j=2}^{n-1} \Pi_j v_j k + 2 \sum_{j=2}^{n-1} \Pi_j$$

Então a coluna a entrar na base pode ser determinada pelo cálculo da árvore-1 mínima em relação aos custos $c_{ij} + \Pi_i + \Pi_j$.

Apesar do grande número de variáveis do problema II, o método simplex é o mais conveniente.

Como este método converge muito lentamente, são considerados outros métodos para minimizar $f(\Pi)$ e portanto maximizar $w(\Pi)$.

MÉTODO ASCENDENTE

Em [11] Held e Karp propõem um método eficiente para aproximar o limite inferior, (a árvore-1 de custo mínimo) do custo da rota ótima e que dá a solução exata do P.C.V

simétrico.

O método consiste em um método ascendente iterativo relacionado como método de relaxação usado para a solução de sistemas de desigualdades lineares, encorporado a um procedimento branch-and-bound.

As iterações para calcular ou aproximar $\max_{\Pi} w(\Pi)$, resultam numa sequência de vetores $\{\Pi^m\}$ n-dimensionais a partir da fórmula.

$$\Pi^{m+1} = \Pi^m + t_m \cdot v_{k(\Pi^m)} \qquad \text{(III)}$$

onde: $k(\Pi)$ - é o índice da árvore-1 de custo mínimo no ponto Π.

$\{t_m\}$ - é uma sequência de escalares

Lema 1: Seja $\overline{\Pi}$ e Π tal que $w(\overline{\Pi}) \geq w(\Pi)$. Então

$$(\overline{\Pi} - \Pi)\, v_{k(\Pi)} \geq w(\overline{\Pi}) - w(\Pi) \geq 0$$

(Demonstração, pag, 9 de [11]).

O hiperplano passando pelo ponto Π e tendo o vetor $v_{k(\Pi)}$ como sua normal, determina um semi-espaço fechado contendo todos os pontos $\overline{\Pi}$, tal que $w(\overline{\Pi}) - w(\Pi) \geq 0$ e a cada passo da iteração está se movendo para dentro do semi-espaço que inclui qualquer ponto onde $w(.)$ assume seu valor máximo e basta, um pequeno passo para produzir um ponto mais próximo do ponto

maximo, do que Π. Isto é, dado pelo Lema seguinte:

Lema 2: Se $0 < t < \dfrac{2(w(\bar{\Pi}) - w(\Pi))}{\| v_{k(\Pi)} \|^2}$ então

$$\| \bar{\Pi} - (\Pi + t.\ v_{k(\Pi)}) \| \leq \| \bar{\Pi} - \Pi \|$$

O ponto $[\Pi + t.\ v_{k(\Pi)}]$ está mais próximo de $\bar{\Pi}$ do que Π está, quando t está no intervalo.

III.10. MÉTODO DE RELAXAÇÃO E O MÉTODO ASCENDENTE

Dado um sistema de desigualdades

$$\sum_{i=1}^{q} a_{ij}\ x_j \geq b\ i \qquad i = 1,2,\ldots,h$$

o método de relaxação constrói uma sequência de vetores $\{x^s\}$ on de x^{s+1} é obtido a partir de x^s, selecionando uma desigualdade violada por x^s e depois tomando um passo em direção ao hiper plano correspondente, ao longo de sua normal, no ponto x^s.

A relação do método acima e o problema de maximizar$_{\Pi}$ $w(\Pi)$ é o seguinte:

Nota-se que $\max_{\Pi} w(\Pi)$ é o valor ótimo do programa linear.

$$\max w$$

$$\text{sujeito a: } w \leq c_k + \Pi \cdot v_k \quad \text{para } \forall k \qquad \text{(IV)}$$

Dado um valor objetivo $\bar{w}$, encontrar um ponto $\bar{\Pi}$, tal que $w(\bar{\Pi}) \geq \bar{w}$ é equivalente a resolver o sistema de desigualdes

$$\bar{w} \leq c_k + \Pi \cdot v_k \quad \text{para } \forall k \qquad \text{(V)}$$

Uma versão do método de relaxação em que a desigualdade violada por Π é selecionada tal que minimize $c_k + \Pi \cdot v_k$ e portanto maximize o "resíduo" $\bar{w} - (c_k + \Pi \cdot v_k)$, leva a um esquema iterativo da forma (III).

INTERPRETAÇÃO GEOMÉTRICA

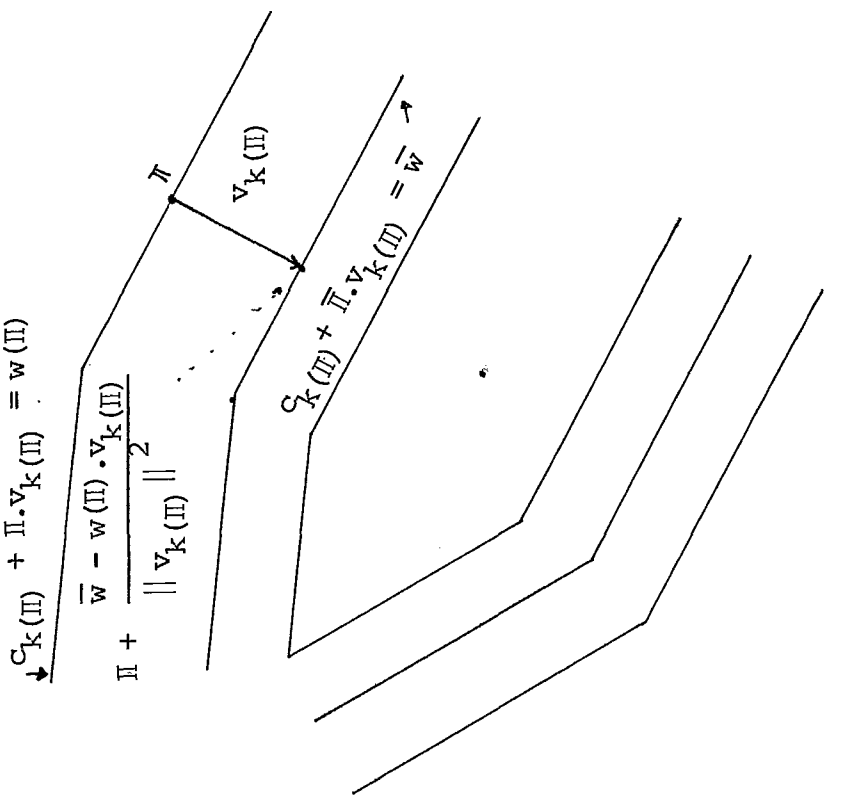
Seja $P_{\bar{w}}$ o poliedro das soluções viáveis para (V), nota-se que:

a) se $w_1 < w_2$ $\quad P_{w_1} \supset P_{w_2}$

b) se $\bar{w} > w(\Pi)$ então $P_{\bar{w}}$ está contido no semi-espaço determinado pelo hiperplano que passa por Π com o vetor normal $v_{k(\Pi)}$.

c) o raio $\{\Pi + t \cdot v_{k(\Pi)} \mid t > 0\}$ é perpendicular a face de $P_{\bar{w}}$ cuja equação é $\bar{w} = c_{k(\Pi)} + \bar{\Pi} \cdot v_{k(\Pi)}$.

O raio intercepta a face no ponto correspondente a $t = \dfrac{\bar{w} - w(\Pi)}{\|v_{k(\Pi)}\|^2}$.



III.11 . UM PROCEDIMENTO BRANCH-AND-BOUND

Como o método ascendente não necessariamente calcula o ponto máximo de $w(\Pi)$ e desde que, o $\max_{\Pi} w(\Pi)$ pode ser menor que C* (custo da rota ótima), é dado um outro método que trabalha com o método ascendente dentro de um procedimento branch-and-bound.

Como se sabe, um procedimento Branch-and - Bound particiona sucessivamente o conjunto das soluções viáveis em subconjuntos e calcula o limite inferior no custo de cada um deles.

No caso do P.C.V. esta partição é caracterizada por indução e exclusão no conjunto de todos os arcos.

Por definição

X,Y - conjuntos disjuntos de arcos

T(X,Y) - conjunto de todas as árvores-1 que incluem os arcos em X e excluem os arcos em Y.

$$W_{X,Y}(\Pi) = \min_{k \in T(X,Y)} [c_k + \Pi \cdot v_k]$$

Portanto $W_{X,Y}(\Pi)$ é um limite inferior no custo para o P.C.V.

- Uma entrada típica no procedimento é da forma:

$$(X,Y,\Pi,W_{X,Y}(\Pi))$$

- Inicia-se com a entrada: $(\phi, \phi, 0, w(0))$ onde 0 é o vetor n-dimensional 0.

- Num passo geral é selecionada a lista $(X,Y,\Pi,W_{X,Y}(\Pi))$ que tenha o menor limite inferior e é efetuada a iteração:

$$\Pi^{m+1} = \Pi^m + \bar{t} \cdot v_{k(\Pi^m)}$$

onde $k(\Pi^m)$ é o índice da árvore-1 de custo mínimo em T(X,Y) relativa ao custo $(c_{ij} + \Pi_i^m + \Pi_j^m)$.

- Para cada m > 0, verifica-se se $W_{X,Y}(\Pi^m) \geq \overline{C}$ onde $\overline{C}$ é um limite superior no custo de uma rota. Caso afirmativo este problema é descartado.

- Senão, compara-se $W_{X,Y}(\Pi^m)$ com a anterior $W_{X,Y}(\Pi^{m-1})$. Se $W_{X,Y}(\Pi^m) > W_{X,Y}(\Pi^{m-1})$ continuar as iterações até para algum h, ter que

$$\max W_{X,Y}(\Pi^{\ell}) = \max W_{X,Y}(\Pi^{\ell})$$

$$0 \leq \ell \leq ph - 1 \qquad 0 \leq \ell \leq p(h+1) - 1$$

isto é, não ocorre nenhum melhoramento para um bloco de p iterações, onde p é um parâmetro do programa.

- Um Π' é então escolhido tal que:

$$W_{X,Y}(\Pi') = \max W_{X,Y}(\Pi^{\ell})$$

$$0 \leq \ell \leq ph - 1$$

e então efetua-se uma ramificação, com novas entradas da forma $(X_i, Y_i, \Pi', W_{X_i Y_i}(\Pi'))$ onde $X_i \supset X$, $Y_i \supset Y$ e toda rota em T (X,Y) está em um dos conjuntos $T(X_i, Y_i)$.

Foi adotada a seguinte estratégia para a ramificação:

Os arcos que ainda não foram incluídos ou excluídos são ordenados de acordo com o seguinte:

Calcula-se a quantidade que o limite poderia crescer, se se excluisse cada arco, e tem-se a sequência resultante dos arcos $e_1, e_2, \ldots, e_r$ onde:

$$W_{X,Y \cup \{e_1\}}(\Pi') \geq W_{X,Y \cup \{e_2\}}(\Pi') \geq \cdots \geq W_{X,Y \cup \{e_r\}}(\Pi')$$

E

$$X_1 = X \qquad Y_1 = Y \cup \{e_1\}$$

$$X_2 = X \cup \{e_1\} \qquad Y_2 = Y \cup \{e_2\}$$

$$\vdots$$

$$X_q = X \cup \{e_1, e_2, \ldots, e_{q-1}\} \quad Y_q = Y \cup R_i$$

onde q é o menor índice para o qual existe um vértice i, tal que X não contém dois arcos incidentes nele, mas X_q tem.

R_i consiste de todos os arcos incidentes em i, e não em X_q.

R_i está em Y_q, desde que qualquer rota que exclue dois arcos incidentes em i, excluem todos os outros arcos incidentes em i.

As listas geradas no processo são de três tipos:

1a.) Aquelas que nunca são escolhidas como uma de menor limite.

2a.) Aquelas que são escolhidas mas são eli

minadas no processo ascendente porque seu comprimento excede $\overline{C}$.

3a.) Aquelas que são selecionadas e a partir das quais continua-se a ramificação.

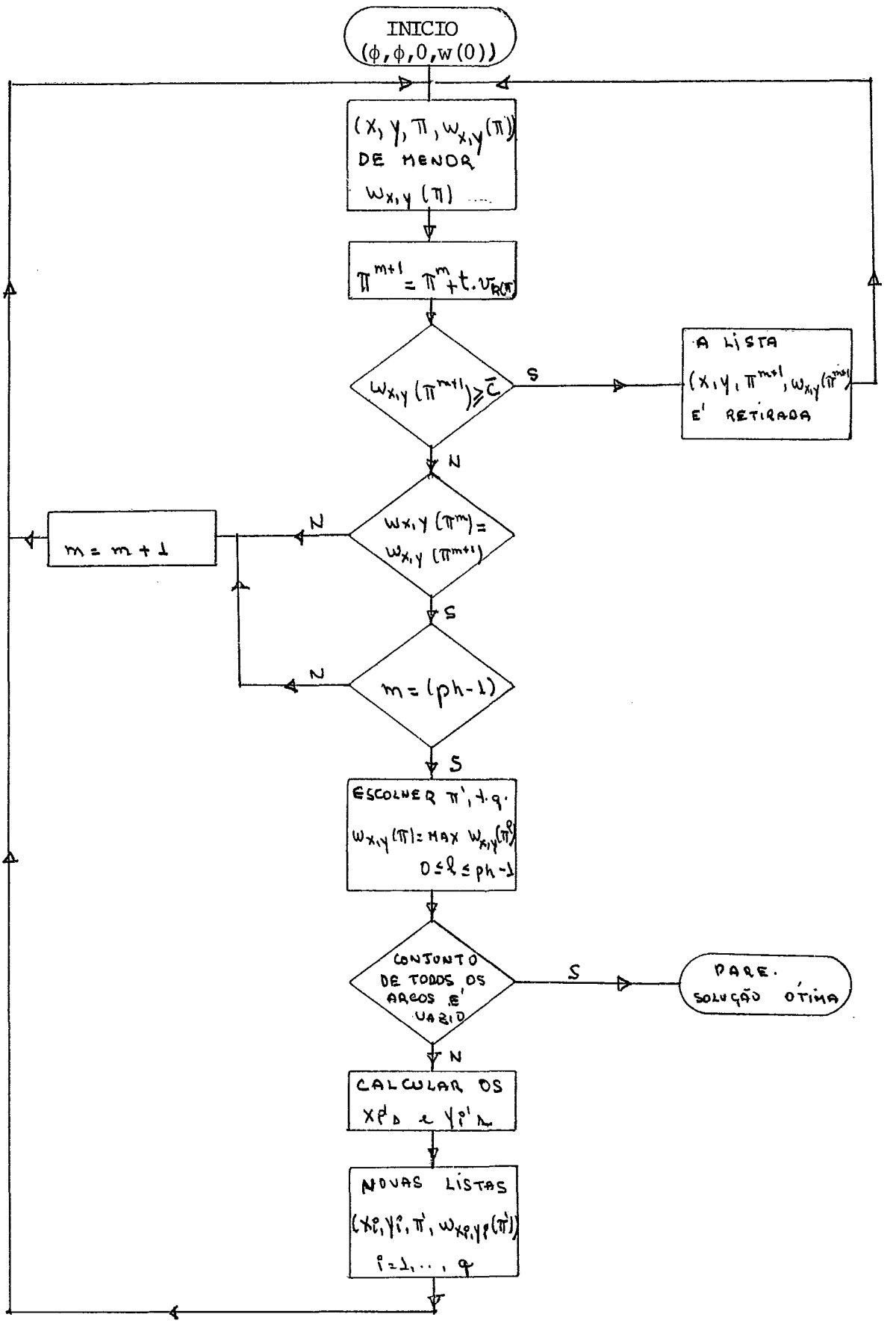
O processo global é uma árvore cujos nós são todas as listas geradas.

Bazaraa e Goode [1] extendem a formulação do P.C.V. simétrico de Held e Karp [10] e [11] para o P.C.V. não simétrico e o seu dual, no Capítulo II.

É generalizado a definição da árvore-1 onde o nó inicial era o nó 1, agora qualquer um dos nó do grafo pode ser o nó inicial.

Um método de otimização pelo subgradiente semelhante ao método de Held e Karp em [11] é estudado para resolver o problema dual do P.C.V., que corresponde a função $w(\Pi)$ em Held e Karp e um procedimento branch-and-bound para encontrar a rota ótima se não foi encontrado no processo de otimização da função dual.

PROCEDIMENTO BRANCH-AND-BOUND



C A P Í T U L O IV

IV.1. ALGUNS MÉTODOS APROXIMADOS

A característica básica de um método aproximado é que, em geral, a sua regra de parada acontece não somente quando um rota ótima é encontrada e a rota final alcançada depende do ponto inicial, de modo que é possível produzir rotas úsando os pontos de partida diferentes.

IV.2. MÉTODO DE SHEN-LIN E B.W. KERNIGHAN

O algorítmo de Shen-Lin e Kernighan [14] é baseado no melhoramento de rotas, isto é, consiste no aperfeiçoamento iterativo de um conjunto de soluções, pelo procedimento seguinte: Gera-se uma solução viável T, aleatoriamente, para o P.C.V. simétrico e a partir dela tentamos encontrar uma outra solução T', melhorando T', por alguma transformação. Se T' for melhor que T, troque T por T' e tentamos melhorá-lo novamente, assim por diante, até não ser possível mais encontrar solução melhor e então a última, vai ser a solução localmente ótima.

Um tipo de transformação bastante aplicado

para melhorar a rota T é a mudança de um número fixo k de arcos de T, por outros pertencentes a S-T, onde S é o conjunto de todos os arcos, tal que a solução resultante seja viável e melhor e isto é repetido tantas vezes quanto for o número de tais grupos encontrados. Quando não é possível melhorar T por tais trocas tem-se uma solução ótima local.

O problema se resume então em encontrar os elementos certos para serem trocados.

Croes [4] e Lin [13] resolveram o P.C.V. simétrico com sucesso para k = 2 e k = 3 respectivamente.

O método a ser descrito aqui é uma generalização de transformações por trocas.

Definições:

S - conjunto de todos os arcos (n(n-1)/2)arcos, caso do P.C.V. simétrico)

T - um subconjunto de S, que forma uma rota (segundo um critério de viabilidade C)

f - função objetivo

O que se quer é encontrar uma rota de comprimento mínimo.

Considerando uma rota arbitrária T de comprimento f(T) e qualquer rota T' de comprimento f(T') < f(T). Supondo que T e T' se diferem por k arcos.

Basicamente, o algoritmo a ser exposto ten

Basicamente, o algoritmo a ser exposto tenta transformar T em T' identificando sequencialmente os k pares de arcos a serem trocados entre T e S-T.

Ou seja, o que se quer é encontrar dois conjuntos de arcos $X = \{x_1,\ldots,x_k\}$ e $Y = \{y_1,\ldots,y_k\}$ tal que se os arcos em X são retirados e substituídos por arcos de Y, o resultado é uma rota com custo menor.

Exemplo:

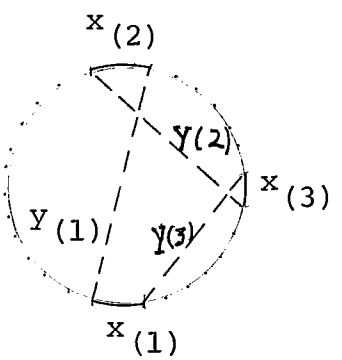


Figura I-a) Rota T

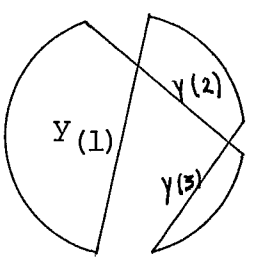


Figura I-b) Rota T'

Nas figuras acima estão ilustrados os casos onde k = 3 e onde a figura I-a) é a rota T, indicando X e Y e a figura I-b) a rota T' resultante.

Os arcos são enumerados naturalmente, $x_{(1)}$ e $y_{(i)}$ tem um nó em comum, assim como $y_{(i)}$ e $y_{(i+1)}$ ($x_{(k+1)} = x_{(i)}$.

Supondo que a enumeração foi feita e a sequência que se quer encontrar é $x_{(1)}, y_{(1)}; x_{(2)}, y_{(2)},\ldots,$.

Como não sabemos como T' deve ser, tenta-se

encontrar qualquer sequência que reduza T para T' com f(T') < f(T) e iterar o processo em T' até não ser possível mais reduções.

Definindo o ganho de trocar x(i) por y(i) como:

$$g_i = |x(i)| - |y(i)|$$

onde $x|(i)|$ e $y|(i)|$ são os comprimentos de x(i) e y(i) respectivamente.

Se f(T') < f(T), tem-se que

$$\sum_{i=1}^{k} g_i = f(T) - f(T') > 0 ,$$

mesmo que exista alguns g_i's negativos.

Isto baseia-se em parte, pelo seguinte fato: se uma sequência de números tem uma soma positiva, existe uma permutação ciclíca desses números, tal que toda soma parcial é positiva. (Demonstração, pags, 501-502 de [14]).

Como só tem sentido considerar sequências de g_i's que tem somas positivas, dentre elas só é necessário considerar aquelas cujas somas parciais sejam sempre positivas . Este critério no ganho reduz bastante o número de sequências a serem examinadas.

O Algoritmo

Passo 1: Gerar uma rota inicial aleatória T.

Passo 2: $G^* = 0$

- Escolher um nó qualquer t(1)
- Escolher um arco x(1) de T, adjacente a t(1)
- i = 1

Passo 3: - t(2) - outro nó de x(1)

- A partir de t(2), escolher o arco y(1) para t(3) com g(1) > 0
- Se este y(1) não existe, vá para o passo 6 (D).

Passo 4: i = i + 1

Escolher x(i) (que liga t(2i-1) e t(2i)) e y(i) pelo se - guinte:

A) x(i) é escolhido de modo que, se t(2i) é ligado a t(1) a configuração resultante é uma rota.

B) y(i) é algum arco no nó t(2i), t também comum a x(i), sujeito a C), D) e E). Se não existe y(i), vá para o Passo 5.

C) Para garantir que os x's e os y's sejam disjuntos , x(i) não pode ser um arco previamente ligado (isto é um y(j), j < i) e de modo semelhante y(i) não pode ser um arco previamente quebrado.

D) $G(i) = \sum_{j=1}^{i} g_j > 0$

E) Para assegurar que o critério de viabilidade de A) possa ser satisfeita em i + 1 o y(i) escolhido deve permitir a quebra de um x(i + 1).

F) Antes de construir y(i), checamos se fechando o cir

cuito ligando t(2i) e t(1) dará um valor de ganho me - lhor do que o melhor já visto previamente.

- Seja y*(i) um arco ligando t(2i) com t(i) e seja y*(i) = |y*(i)| - |x(i)|.

- Se G(i - 1) + g*(i) > G*, faça G* = G(i - 1) + g*(i) e faça k = i. (G* é o melhor melhoramento em T. G* $\geq$ 0 é monótona não decrescente. O índice k define o conjunto que vai ser trocado para achar G*).

Passo 5: - Termine a construção de x(i) e y(i) no Passo 2 até o Passo 4, quando não existirem mais arcos x(i) e y(i) que satisfaçam os Passos 4 (C-E) ou quando $G_i \leq G^*$. (Este é o critério de parada).

- Se G* > 0, tome uma rota T' com f(T') = f(T) - G* e repita o processo inteiro a partir do Passo 2, usando T' como a rota inicial.

Passo 6: - Se G* = 0, efetua-se uma volta para trás limitada, pelo seguinte:

A) Repetir os Passos 4 e 5, escolhendo y(2)'s tão longos quanto satisfaçam o critério de ganho, g(1) + + g(2) > 0.

B) Se todas as escolhas de y(2) no Passo 4 B) são verificadas sem melhora, volte ao Passo 4 A) e tente alternar a escolha por x(2).

C) Se com isto não houver melhoramento, uma outra volta é feita no Passo 3, onde os y(1)'s são examina -

dos em ordem crescente ao comprimento.

D) Se os y(1)'s são também verificados sem melhoramento, tentar alternar x(1) no Passo 2.

E) Se isto falhar, um novo t(1) é selecionado e repete - se o procedimento a partir do Passo 2.

Passo 7: O procedimento termina quando todos os n valores de t(1) forem examinados sem um melhoramento. O que se pode fazer é considerar outras rotas aleatórias no Passo 1.

Exemplo:

Iniciando com uma rota T e dois nós adjacentes t(1) e t(2), x(1) é o arco ligando-os

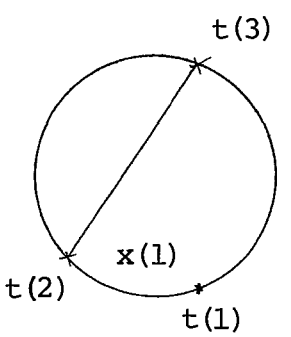
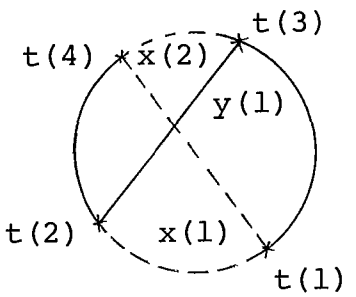


Figura 2 - Rota T e o nó t(3), o nó mais próximo de t(2) e y(1) o arco (t(2),t(3)).

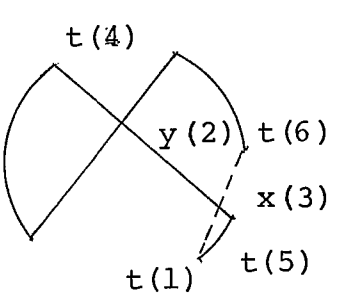
Como os x's e y's devem ser disjuntos, y(1) não pode ser um arco que já tenha sido ligado a t(2).

- g(1) = |x(1)| - |y(1)|;
- se g(1) < 0, volte |Passo 6 D| e faça t(2) ser outro vizinho de t(1)
- i = 2 e t(4) o vizinho de t(3) e x(2) e o arco (t(3),

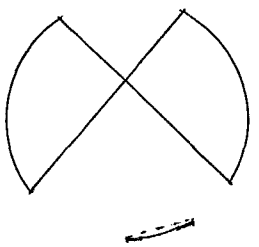
t(4)), como mostra a figura abaixo:



- Se y(2) foi escolhido para ligar t(4) a t(1), o resultado seria uma rota e se g(1) + g(2) > 0 podemos melhorar T trocando x(1) e y(2). Este melhoramento é G* no Passo 4 (F).
- Escolha o vizinho mais próximo ao nó t(4) e faça y(2) ser o arco (t(4), t(5)) , t(5) não pode ser um dos nós que já estiveram ligados a t(4).
- Existe apenas uma escolha para t(6) e x(3) se desejarmos fechar a rota imediatamente, como mostra a figura abaixo:



- Se fizermos x(3) ser outro arco ligado a t(5) a rota fica separada em duas peças, como na figura:



- Checamos novamente para ver ser ligando t(6) a t(1) dá um ganho melhor do que o avaliado ligando t(4) a t(1), atualize G* e faça k = 3.
- Se $g(1) + g(2) + g(3) \leq G^*$, assume-se a troca em k = 2 e também se a melhor escolha para y(3) foi (t(6), t(1)). Caso contrário, continua-se o procedimento, selecionando t(7) e assim por diante.

IV.3. MÉTODO HEURÍSTICO DE KARG E THOMPSON

Karg e Thompson [12], definem uma rota do P.C.V. como uma permutação aciclíca $(i_1, i_2, \ldots, i_n)$ de inteiros 1, 2, ... , n.

As idéias heurísticas deste método são baseadas na natureza geométrica e aritmética do problema, e apesar do método não garantir a rota ótima, dá soluções bastante próximas do ótimo.

O método aqui desenvolvido consiste em duas fases:

1a.) É a parte básica do método, quando se constrói um conjunto de soluções ótimas locais que se supõe estejam próximas ao ótimo ou que contenha a solução ótima.

2a.) Consiste de um procedimento de partição, do problema original com n grande em vários subproblemas, que são resolvidos pelo algoritmo da primeira fase.

PASSO BÁSICO - CÓDIGO 1

A única restrição feita para o método é que as permutações a serem construídas sejam aciclícas.

O método consiste em começar com duas cidades escolhidas aleatoriamente das n cidades e considerá-las como uma permutação de comprimento 2, e então inserir uma terceira cidade de modo que o comprimento da permutação de 3 cidades seja mínimo e então inserir uma quarta cidade tal que a permutação seja de comprimento mínimo e assim por diante até n.

Passo 1: Dadas n cidades, a matriz de custo A, ordene as n cidades arbitrariamente. Some as duas primeiras cidades da ordenação, para formar uma permutação aciclíca de comprimento 2, (I1, I2).

Passo 2: Supondo que uma permutação $(I_1, I_2, \ldots, I_k)$ de k cidades foi construída $(2 \leq k < n)$.

Das cidades restantes, tome a próxima cidade da ordenação e chame de H. Fazendo j variar de 1 até k, calcular

$$d(j) = A(I_j,H) + A(H, I_{j+1}) - A(I_j, I_{j+1})$$

onde $I_{k+1} = I_1$ quando $j = k$.

Passo 3: Determinar s , onde

$d(s) = \min \{d(j)\}$ $\quad j = 1,\ldots,k.$

Passo 4: Fazer $I_{j+1} = I_j$ para $j = s+1,\ldots,k$ e $I_{s+1} = H$

$(I_1, I_2,\ldots,I_s, H, I_{s+2},\ldots,I_{k+1})$

permutação com k + 1 cidades.

Passo 5: Se k + 1 = n então pare.

Senão faça k = k + 1 e volte ao Passo 2.

Exemplo: n = 5

	1	2	3	4	5
1	∞				
2	30	∞			
3	26	24	∞		
4	50	40	24	∞	
5	40	50	26	30	∞

Passo 1: Ordenando as n cidades arbitrariamente

1, 2 , 3, 4, 5.

$(I_1, I_2) = (1,2)$

Passo 2: H = 3

d(1) = A(1,3) + A(3,2) - A(1,2) = 26 + 24 - 30 = 20

d(2) = A(2,3) + A(3,1) - A(2,1) = 24 + 26 - 30 = 20

Passo 3: d(s) = min {d(1), d(2)}

s = 1 ou 2

Passo 4: (1,3,2) ou (1,2,3)

Passo 5: k + 1 = 3

Passo 2: H = 4

Considerando a permutação (1,3,2)

d(1) = A(1,4) + A(4,3) - A(1,3) = 50 + 24 - 26 = 52

d(2) = A(3,4) + A(4,2) - A(3,2) = 24 + 40 - 24 = 40

d(3) = A(2,4) + A(4,1) - A(2,1) = 40 + 50 - 30 = 60

Passo 3: d(s) = 40 e s = 2

Passo 4: (1,3,4,2)

Passo 5: k + 1 = 4

Passo 2: H = 5

d(1) = A(1,5) + A(5,3) - A(1,3) = 40 + 26 - 26 = 40

d(2) = A(3,5) + A(5,4) - A(3,4) = 32

d(3) = A(4,5) + A(5,2) - A(4,2) = 40

d(4) = A(2,5) + A(5,1) - A(2,1) = 60

Passo 3: d(s) = 32 e s = 2

Passo 4: (1,3,5,4,2)

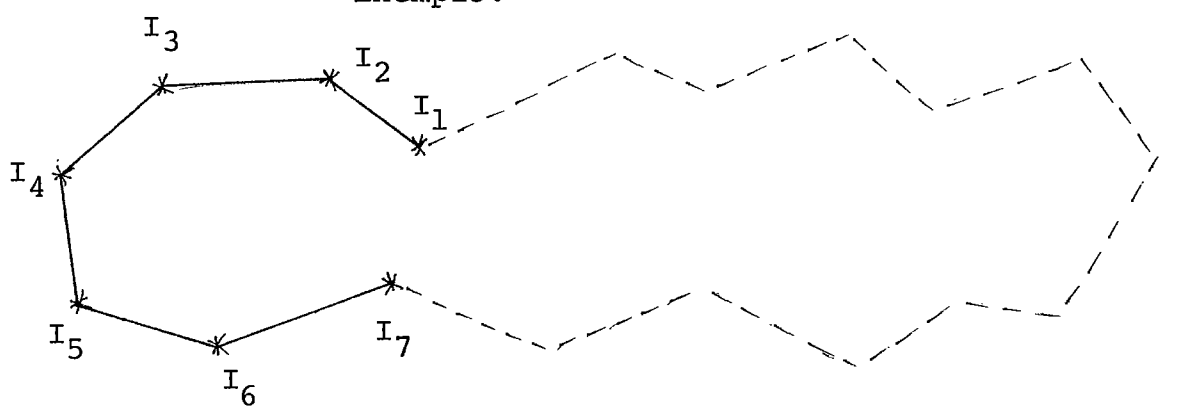
Passo 5: k + 1 = 5 PARE

Valor da solução = 152 e a solução ótima é (1,2,3,4,5) com valor de 148.

O CÓDIGO 1 trabalha bem com problemas de n pequeno, mas um fato interessante acontece quando o CÓDIGO 1 é aplicado várias vezes em problemas de n grande. Em geral, nas soluções melhores, o problema tende a ser convexo nas periferias da rota, mas não acontecendo o mesmo no centro do problema.

Foi acrescentado ao CÓDIGO 1, uma flexibilidade para especificar subproblemas e trabalhar com eles separadamente.

Exemplo:



Notamos que o problema do exemplo pode ser fatorado em dois outros: o primeiro com 7 cidades e outro com n-5 cidades. O problema com 7 cidades sendo aproximadamente convexo é fácil de ser resolvido pelo CÓDIGO 1 e o problema com n - 5 pode ser resolvido como um novo problema sujeito a novas fatorações. A única restrição nas soluções dos subproblemas é que o arco em comum deve aparecer apenas uma vez na solução de cada subproblema no caso simétrico e no caso não simétrico, o arco comum deve aparecer em uma direção em um subproblema e em outro, em direção oposta.

CÓDIGO 2 - ALGORITMO DE APRENDIZADO

O CÓDIGO 2 é o algoritmo CÓDIGO 1 acrescido de um procedimento de fatoração, isto é, um algoritmo que aprende aspectos nos resultados dos cálculos anteriores do programa, define os subproblemas escolhidos para trabalhar mais tarde com mais detalhe.

Passo 0: Ler os dados iniciais - as n cidades e a matriz de custo A.

Passo 1: Escolher duas cidades quaisquer X e Y e colocar as cidades restantes em uma ordem qualquer.

Passo 2: Com X e Y, as duas cidades iniciais, aplicar o CÓDIGO 1, para construir uma permutação de n cidades e calcular o comprimento da rota.

Passo 3: Voltar k - 1 vezes ao Passo 1 e construir uma nova permutação e cada vez comparar com a rota previamente encontrada, guardando a melhor observada em k experimentos, onde k é lido nos dados ou então é escolhido como um múltiplo de n. Escrever a melhor rota, com o valor de seu comprimento.

Passo 4: Escolher uma cidade A'aleatoriamente na melhor rota encontrada, depois encontrar B, que esteja mais distante de A'e então encontrar C, também que esteja mais distante de B. C é a cidade diâmetro.

Passo 5: A partir da cidade diâmetro, definir um subproblema convexo pelo seguinte: fazendo I(P) = C, onde P é o índi-

ce da cidade C na melhor rota encontrada. Calcular a distância d de I(P) a I(P-1). Depois calcule d de I(P-1) a I(P+1) e a de I(P+1) a I(P-2) e assim por diante, enquanto d cresce. Continuar o procedimento mesmo enquanto d decresce e parar tão logo quanto d começa a crescer novamente. Então essas cidades formam um subproblema convexo.

Passo 6: Uma vez definido o subproblema convexo verificar se ele inclui todas as cidades da lista corrente, então pare.

Senão, temos um novo subproblema.

Determinar as cidades X e Y e voltar ao Passo 1.

IV.4. PROGRAMA E RESULTADOS

Foi programado o método acima em FORTRAN IV. O programa principal consiste no CÓDIGO 2, onde o CÓDIGO 1 é a subrotina CODE 1A, na primeira vez e CODE 1B após ser definido um subproblema convexo, porque foi necessário considerar o arco fixo que deve aparecer uma vez no subproblema convexo e outra vez no problema com as cidades restantes.

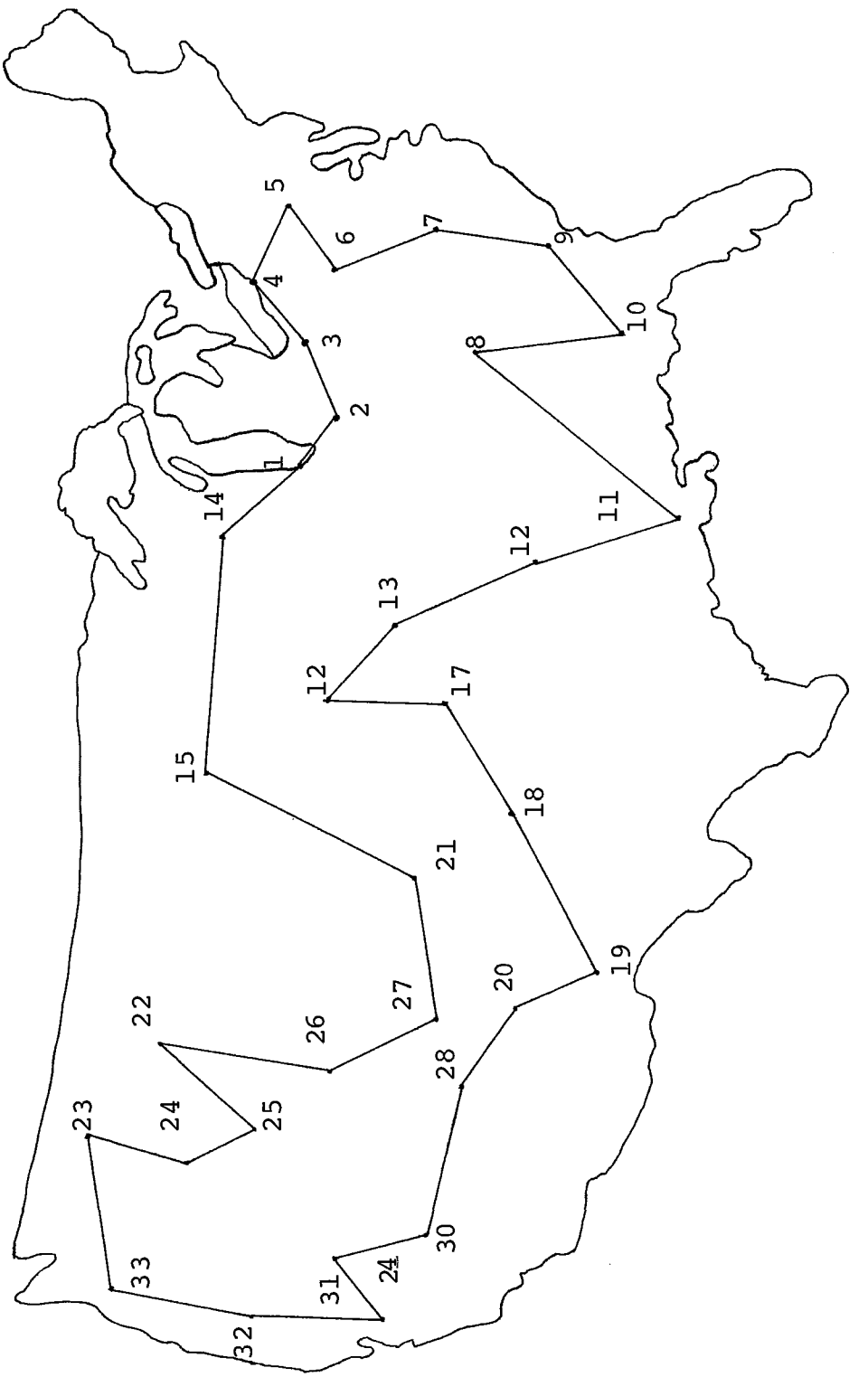
A subrotina a determinar o subproblema convexo é ICONV.

Após efetuar os cortes, a subrotina CONST , no final, junta todos os subproblemas convexos para formar a rota final.

O programa foi rodado com n = 33 e n = 42 com os dados de [12] e foram encontradas rotas próximas do ótimo.

Sendo um método heurístico, a rota final depende muito das soluções iniciais e não influiu se estas fossem soluções viáveis ou não.

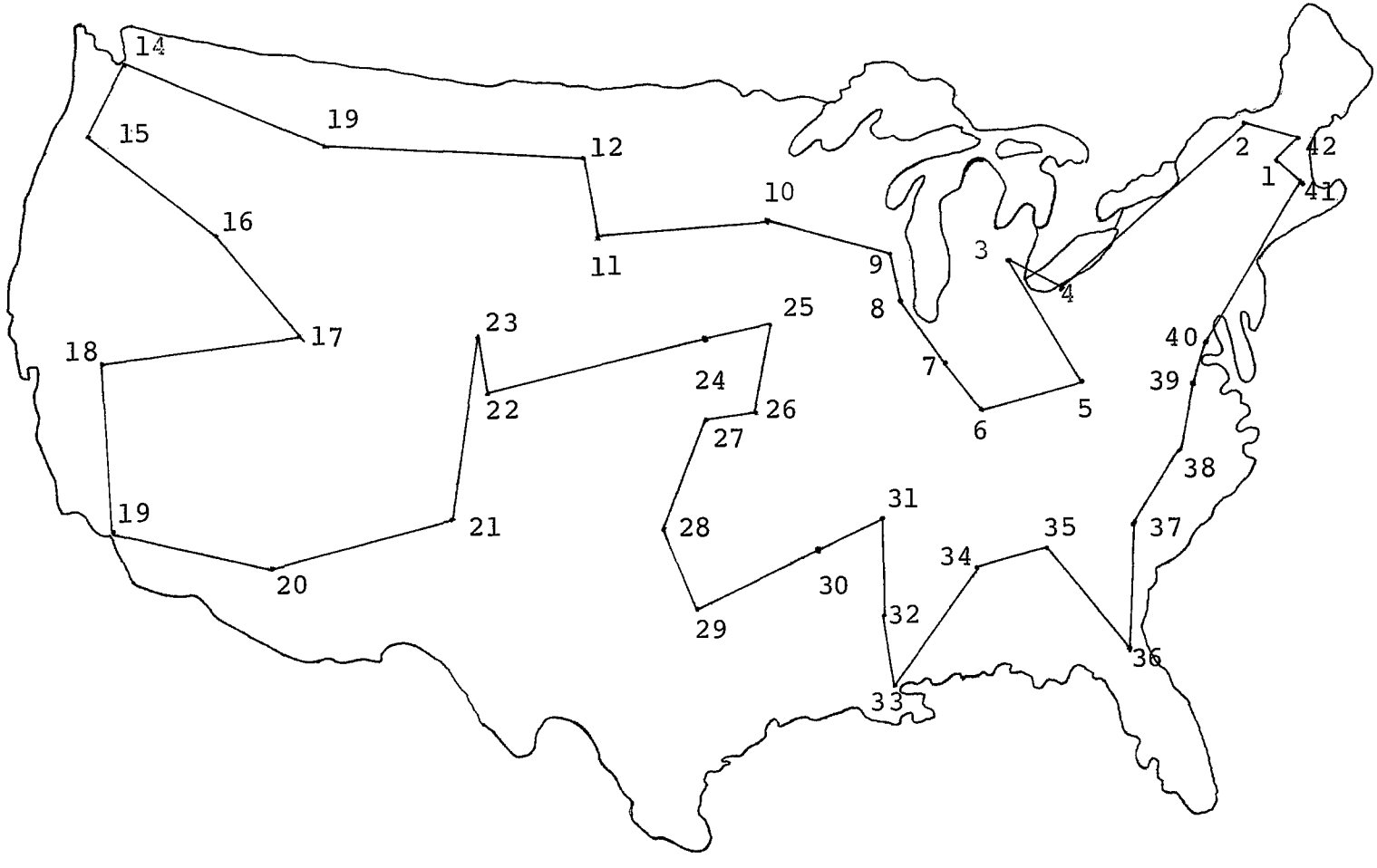
N = 33
MELHOR SOLUÇÃO ENCONTRADA
DISTÂNCIA TOTAL = 10.959



N = 42

MELHOR SOLUÇÃO ENCONTRADA

DISTÂNCIA TOTAL = 717



C A P Í T U L O V

V.I. GENERALIZAÇÃO DO P.C.V.

Um caso especial do P.C.V. é o problema de sequenciamento de n tarefas de Gilmore e Gomory [9].

Dada uma máquina com uma variável x real, que descreve seu estado e as tarefas $J_1, J_2, \ldots, J_n$ que ela deve sequenciar. Cada tarefa J_i, requer um estado inicial A_i e um estado final B_i, isto é, a tarefa J_i, só é iniciada quando $x = A_i$ e termina quando $x = B_i$.

Com cada tarefa está relacionada um custo (dinheiro, tempo, etc) de mudar o estado da máquina para começar uma outra tarefa.

O problema é encontrar a sequência das n tarefas de custo mínimo.

Se a tarefa J_j seguir a J_i, então o estado da máquina muda para A_j.

O custo desta mudança é c_{ij} e é dado por:

$$c_{ij} = \int_{B_i}^{A_j} f(x)\, dx \quad \text{se} \quad A_j \geq B_i$$

$$c_{ij} = \int_{A_j}^{B_i} g(x)\, dx \quad \text{se} \quad B_i > A_j$$

onde f(x) e g(x) são funções integráveis satisfazendo f(x) + + g(x) $\geq$ 0. f(x) pode ser interpretada como o custo densidade para fazer crescer a variável estado da máquina e g(x) o custo densidade para fazer decresce-la.

Um exemplo: a temperatura variável de um forno é x e uma tarefa J_i entrará no forno a uma temperatura A_i e sairá dela a B_i e para a próxima tarefa, a temperatura muda. f(x) é o custo de aumentar a temperatura de um grau e g(x) o custo de esfriar um grau quando a temperatura é x.

Uma outra generalização do P.C.V. é o P.C.V. depende do tempo (P.C.V.D.T.) em [8]. Neste problema o custo de transição não depende apenas de duas locações sucessivas , mas também do período quando essas transições ocorrem.

Uma aplicação deste problema é o problema geral de sequenciamento de tarefas em uma máquina, que consiste em minimizar o custo total do tempo de execução em uma máquina de sequenciar tarefas, onde o custo é uma função não decrescente do tempo de completar cada tarefa.

A P E N D I C E

```
FILE  8=CARTAO,UNIT=READER
C                    PROGRAMA PRINCIPAL
      INTEGER A(33,33),TI(33),CE(75),SI,TE(33),RE(33),CI(75),SL,
     *MT(20),TO(25,25),IRR(33),TA1(20),TA2(20),VI(33),DI(33),DO(3
     *3)
      COMMON/RAN/S
      S=2**19-1
      READ(8,/) N
      DO 2 I=1,N
      READ(8,60) (A(I,J),J=1,I)
   60 FORMAT(2014)
    2 CONTINUE
      DO 31 I1=1,N
      DO 31 J1=1,N
      A(I1,J1)=A(J1,I1)
   31 CONTINUE
      READ(8,/) (TI(I),I=1,N)
      DO 10 I=1,N
   10 TI(I)=TI(I)*10**4
      NZ=1
      NO=N
C                    CALCULO DA ROTA  DE MENOR DISTANCIA COM K=75
   44 CE(1)=0
      K=1
      CALL ICAL(TI,A,N,CE,K,DI,MO)
      DO 11 I=1,N
   11 RE(I)=TI(I)
      NEN=1
      DO 7 K=2,75
      CE(K)=0
      CALL ICAL(TI,A,N,CE,K,DI,MO)
      IF(CE(NEN).LE.CE(K)) GO TO 7
      NEN=K
      CE(NEN)=CE(K)
      DO 12 I=1,N
   12 RE(I)=TI(I)
    7 CONTINUE
      WRITE(5,/) CE(NEN),(RE(I)/10**4,I=1,N)
C                    DETERMINACAO DA CIDADE DIAMETRO
      JA=RE(N/2)
      CALL MAXIM (JA,A,RE,LA,N,VI,MO)
      JB=RE(LA)
      CALL MAXIM (JB,A,RE,LO,N,VI,MO)
      JC=RE(LO)
      WRITE(5,63) JC
   63 FORMAT(5X,'JC=',I6,//)
C       DETERMINACAO DAS CIDADES DO SUBPROBLEMA CONVEXO
      CALL ICONV(LO,RE,N,A,TE,MV,TA1,TA2,DO,MO)
      IF(MOD(TE(1)/100,100).EQ.TE(MV)/10**4) CALL ROT(TE,MV,MO)
      IF(MOD(TE(1)/100,100).EQ.0) GO TO 20
```

```
      IF(MOD(TE(1)/100,100).EQ.TE(2)/10**4) GO TO 20
      JY=TE(1)
      DO 28 L=2,(MV+1)
      JH=TE(L)
      TE(L)=JY
      JY=JH
   28 CONTINUE
      MV=MV+1
      TE(1)=MOD(TE(2)/100,100)*10**4+(TE(2)/10**4)*100
   20 IF(MOD(TE(MV)/100,100).EQ.0.OR.MOD(TE(MV)/100,100).EQ.TE(MV
     **-1)/10**4) GO TO 23
      TE(MV+1)=MOD(TE(MV)/100,100)*10**4+(TE(MV)/10**4)*100
      MV=MV+1
C     CALCULO DO SUBPROBLEMA COM AS CIDADES RESTANTES
   23 IF(MV.EQ.N) GO TO 57
      IW=TE(MV)
      IF(MOD(IW/100,100).NE.0) IW=(IW/10**4)*10**4
      IZ=TE(1)
      IF(MOD(IZ/100,100).NE.0) IZ=(IZ/10**4)*10**4
      TI(1)=IZ+IW/100
      TI(2)=IW+IZ/100
      MI=3
      DO 59 J=1,N
      DO 43 MQ=1,MV
      IF(RE(J).EQ.TE(MQ)) GO TO 59
   43 CONTINUE
      TI(MI)=RE(J)
      MI=MI+1
   59 CONTINUE
      IF(MOD(TE(1)/100,100).NE.0.AND.MOD(TE(MV)/100,100).NE.0) GO
     * TO 27
      IF(MOD(TE(1)/100,100).NE.0) GO TO 25
      IF(MOD(TE(MV)/100,100).NE.0) GO TO 26
      CALL ROT(TE,MV,MO)
      ID=TE(1)
      TE(1)=ID+TE(2)/10**2
      TE(2)=TE(2)*ID/10**2
      GO TO 61
   25 CALL ROT(TE,MV,MO)
      TE(3)=TE(3)+TE(1)/10**4
      TE(2)=TE(2)+TE(1)/10**4
      TE(1)=TE(1)+TE(2)/10**2
      IF(MV.LE.4) GO TO 54
      GO TO 61
   26 CALL ROT(TE,MV,MO)
      CALL ROT(TE,MV,MO)
      TE(1)=TE(1)+TE(3)/10**4
      TE(2)=TE(2)+TE(3)/10**4
      TE(3)=TE(3)+TE(2)/10**2
      IF(MV.LE.4) GO TO 54
```

```
      GO TO 61
   27 CALL ROT(TE,MV,MQ)
      CALL ROT(TE,MV,MQ)
      M1=TE(2)/10**4
      M2=TE(3)/10**4
      TE(2)=M1*10**4+M2*100
      GO TO 54
   61 WRITE(5,69) (MQ,TE(MQ),MQ=1,MV)
   69 FORMAT(5X,'TE(',I2,')=',I6)
      CI(NZ)=0
   57 IF(MV.GT.3) GO TO 53
      DO 74 I=1,MV
      IF(MOD(TE(I)/100,100).NE.0) GO TO 54
   74 CONTINUE
   53 CALL CODE10(TE,A,MV,CI,NZ,DI,MQ)
      GO TO 64
   54 CALL DIST(A,TE,MV,CI,NZ,MQ)
   64 WRITE (5,65) NZ,CI(NZ),(TE(J),J=1,MV)
   65 FORMAT(//,20X,'SUBPROBLEMA CONVEXO',//,5X,'CI(',I2,')=',I6,
     *5X,14(I6,','),//,18(I6,','),//,18(I6,','),///)
      DO 29 J=1,MV
   29 TO (NZ,J)=TE(J)
      MT(NZ)=MV
      IF(MV.EQ.N)  GO TO 40
      NZ=NZ+1
      N=N-MV+2
      IF(N.GT.3) GO TO 44
      DO 58 J=1,N
   58 TE(J)=TI(J)
      MV=N
      GO TO 54
   40 JS=0
      DO 76 IO=1,NZ
      JS=JS+CI(IO)
   76 CONTINUE
      CALL CONST(TD,NZ,MT,IRR,MQ)
      WRITE(5,78) JS,(IRR(M)/10**4,M=1,MQ)
   78 FORMAT(8(/),10X,'DISTANCIA TOTAL=',5X,I8,///,10X,
     *'ROTA OTIMA',10X,30(I2,','),//,10X,20(I2,','))
      STOP
      END
```

```
      SUBROUTINE ICAL (T,A,N,C,K,D,MO)
      INTEGER T(1),A(MO,MO),C(1),D(1)
      DO 30 I=1,N
      IF(MOD(T(I)/100,100).NE.0) GO TO 8
   30 CONTINUE
      CALL RAND(T,N,MO)
      CALL CODE1A(A,T,N,C,K,D,MO)
      GO TO 10
    8 CALL IRANT(T,N,MO)
      CALL CODE1B(T,A,N,C,K,D,MO)
   10 RETURN
      END
      SUBROUTINE RAND(T,N,MO)
      INTEGER T(1)
      COMMON/RAN/X
      IO=0
      IT=1
   21 NOT=T(IT)
   22 A=RANDOM(X)
      DO 10 I=(IT+1),(N-1)
      IF(A.GE.(I-1)/FLOAT(N).AND.A.LT.I/FLOAT(N)) GO TO 17
   10 CONTINUE
      IF(I.EQ.IO) GO TO 22
      IO=I
   17 T(IT)=T(I)
      T(I)=NOT
      IF(IT.EQ.2) GO TO 60
      IT=IT+1
      GO TO 21
   60 RETURN
      END
      SUBROUTINE IRANT(T,N,MO)
      INTEGER T(1)
      COMMON/RAN/X
      IF(MOD(T(1)/100,100).NE.0) GO TO 21
      IF(MOD(T(2)/100,100).NE.0.AND.MOD(T(2)/100,100).NE.T(1)/100
     *00) GO TO 22
   21 IZ=1
      GO TO 23
   22 IZ=2
   23 NOT=T(IZ)
      NAT=T(IZ+1)
      A=RANDOM(X)
      DO 10 I=3,(N-1)
      IF(A.GE.(I-1)/FLOAT(N).AND.A.LT.I/FLOAT(N)) GO TO 17
   10 CONTINUE
   19 T(IZ)=T(I-1)
      IF(MOD(T(I-1)/100,100).EQ.T(I-2)/10000) GO TO 27
      T(IZ+1)=T(I)
      T(I-1)=NOT
```

```
      T(I)=NAT
      GO TO 60
   17 IF(I.EQ.N) GO TO 19
      IF(MOD(T(I)/100,100).EQ.0) GO TO 18
      IF(MOD(T(I)/100,100).EQ.T(I-1)/10**4) GO TO 19
   26 T(IZ)=T(I)
      T(IZ+1)=T(I+1)
      T(I)=NOT
      T(I+1)=NAT
      GO TO 60
   18 IF(MOD(T(I+1)/100,100).EQ.0) GO TO 26
      T(IZ)=T(I+1)
      T(IZ+1)=T(I+2)
      T(I+1)=NOT
      T(I+2)=NAT
      GO TO 60
   27 T(IZ)=T(I-2)
      T(IZ+1)=T(I-1)
      T(I-2)=NOT
      T(I-1)=NAT
   60 RETURN
      END
      SUBROUTINE ORDENA(T,IMIN,M,MO)
      INTEGER T(1)
      IL=T(IMIN)
      DO 4 I=IMIN+1,M
      ILL=T(I)
      T(I)=IL
      IL=ILL
    4 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CODE1A(A,T,N,C,K,D,MD)
      INTEGER A(MD,MD),T(1),C(1),D(1),H
      DO 7 M =3,N
      M1=M-1
      H=T(M)
      L1=T(1)
      L2=T(2)
      IMIN=1
      D(IMIN)=A(L1/10**4,H/10**4)+A(H/10**4,L2/10**4)-A(L1/10**4,
     *L2/10**4)
      DO 1 I=2,M1
      N1=T(I)
      N2=T(I+1)
      IF((I+1).NE.M ) GO TO 17
      N2=T(1)
   17 D(I)=A(N1/10**4,H/10**4)+A(H/10**4,N2/10**4)-A(N1/10**4,N2/
     *10**4)
      IF(D(IMIN).LT.D(I)) GO TO 1
      IMIN=I
    1 CONTINUE
      IF((IMIN+1).EQ.M) GO TO 26
      IL=T(IMIN+1)
      MIN1= IMIN+2
      DO 3 I=MIN1,M
      ILL=T(I)
      T(I)=IL
      IL=ILL
    3 CONTINUE
   26 T(IMIN+1)=M
    7 CONTINUE
      DO 4 I=1,N
      L3=T(I)
      L4=T(1)
      IF((I+1).GT.N) GO TO 5
      L4=T(I+1)
    5 C(K)=C(K)+A(L3/10**4,L4/10**4)
    4 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MAXIM(JW,A,R,LI,N,V,MO)
      INTEGER A(MO,MO),R(1),V(1)
      MR=R(1)
      MAX=A(JW/10**4,MR/10**4)
      LI=1
      DO 22 M=2,N
      MR=R(M)
      V(M)=A(JW/10**4,MR/10**4)
      IF(V(M).LE.MAX) GO TO 22
      MAX=V(M)
      LI=M
   22 CONTINUE
      RETURN
      END
      SUBROUTINE COMV(LI,R,N,A,TW,MD,TA,TU,DU,MO)
      INTEGER R(1),DU(1),A(MO,MO),TW(1),TA(1),TU(1)
      IP=0
      IT=1
      IO=LI
      K=LI
      DO 1 I=1,N-1
      K=K+((-1)**I)*I
      IF(K.LE.0) K=K+N
      IF(K.GT.N) K=K-N
      I1=R(IO)
      I2=R(K)
      DU(I)=A(I1/10**4,I2/10**4)
      IF(I.LT.3) GO TO 26
      IF(DU(I-2).GT.DU(I-1).AND.DU(I-1).LT.DU(I)) GO TO 2
   26 IF(MOD(I,2).EQ.0) GO TO 22
      IP=IP+1
      TA(IP)=R(K)
      GO TO 21
   22 IT=IT+1
      TU(IT)=R(K)
   21 IO=K
    1 CONTINUE
    2 DO 24 J=1,IP
      M=IP-J+1
      TW(J)=TA(M)
   24 CONTINUE
      TU(1)=R(LI)
      DO 25 I=1,IT
   25 TW(IP+I)=TU(I)
      MD=IP+IT
      RETURN
      END
```

```
      SUBROUTINE ROT(T,MD,MO)
      INTEGER T(1)
      IME=T(MD)
      DO 32 J=1,MD-1
      I=MD-J+1
   32 T(I)=T(I-1)
      T(1)=IME
      RETURN
      END
      SUBROUTINE CODE18(T,A,N,C,K,D,NO)
      INTEGER T(1),A(MO,MO),C(1),D(1)
      M=3
   41 IF(M.GE.N) GO TO 40
      IF(MOD(T(1)/100,100).EQ.0) GO TO 10
      M=M+1
      IF(MOD(T(1),100).EQ.0) GO TO 11
      IF(M.GT.4) GO TO 12
      IF(MOD(T(4)/100,100).NE.0) GO TO 13
      M=M+2
      IF(M.GT.N) GO TO 40
      GO TO 12
   13 M=M+3
      IF(M.GT.N) GO TO 40
   12 IF(MOD(T(M)/100,100).NE.0) GO TO 15
      CALL FIXI (T,A,M,D,MO)
      GO TO 41
   15 M=M+1
      CALL IFIX(T,A,M,D,MO)
      GO TO 41
   11 IF(MOD(T(3)/100,100).EQ.0) GO TO 24
      IF(M.EQ.N) GO TO 40
      IF (M.EQ.4) M=M+1
   24 IF(MOD(T(M)/100,100).NE.0) GO TO 16
      CALL DIXC(T,A,M,D,MO)
      GO TO 41
   16 M=M+1
      CALL IBRID(T,A,M,D,MO)
      GO TO 41
   10 IF(MOD(T(2)/100,100).EQ.0) GO TO 18
      M=M+1
      GO TO 24
   18 IF(M.GT.3) GO TO 30
      IF(MOD(T(M)/100,100).NE.0) GO TO 31
      CALL DIXC(T,A,M,D,MO)
      GO TO 30
   31 M=M+1
      CALL IBRID(T,A,M,D,MO)
   30 M=M+1
      IF(M.GT.N) GO TO 40
      GO TO 24
```

```
      ENTRY DIST(A,T,N,C,K,MO)
40    DO 4 I=1,N
      L3=T(I)
      IF(MOD(T(I),100).NE.0.AND.MOD(T(I+1),100).NE.0) GO TO 4
      IF(MOD(T(I)/100,100).NE.0.AND.MOD(T(I)/100,100).EQ.T(I+1)/1
     *0**4) GO TO 4
      L4=T(1)
      IF((I+1).GT.N) GO TO 25
      L4=T(I+1)
25    C(K)=C(K)+ A(L3/10**4,L4/10**4)
 4    CONTINUE
      RETURN
      END
      SUBROUTINE FIXI(T,A,M,D,MO)
      INTEGER T(1),A(MO,MO),D(1),H
      L1=T(3)
      L2=T(4)
      H=T(M)
      IMIM=4
      D(IMIM)=A(L1/10**4,H/10**4)+A(H/10**4,L2/10**4)-A(L1/10**4,
     *L2/10**4)
      IF(M.GT.5) GO TO 31
      I=5
      GO TO 32
31    DO 2 I=5,M
      IF(MOD(T(I)/100,100).EQ.T(I-1)/10000) GO TO 2
32    N1=T(I-1)
      N2=T(I)
      IF(I.NE.M) GO TO 17
      N2=T(1)
17    D(I)=A(N1/10**4,H/10**4)+A(H/10**4,N2/10**4)-A(N1/10**4,N2/
     *10**4)
      IF(D(IMIM).LE.D(I)) GO TO 2
      IMIM= I
 2    CONTINUE
      IF(IMIM.EQ.M) GO TO 18
      CALL DROZNA (T,IMIM,M,MO)
18    T(IMIM)=H
      RETURN
      END
```

```
      SUBROUTINE IFIX(T,A,M,D,MO)
      INTEGER T(1),A(MO,MO),D(1)
      L1=T(3)
      L2=T(4)
      IB1=T(M-1)
      IB2=T(M)
      IMIM=4
      D(IMIM)=A(L1/10**4,IB1/10**4)+A(IB2/10**4,L2/10**4)-A(L1/10
     **4,L2/10**4)
      DO 2 I=5,M-1
      IF(MOD(T(I)/100,100).EQ.T(I-1)/10000) GO TO 2
      N1=T(I-1)
      N2=T(I)
      IF(I.NE.M-1) GO TO 3
      N2=T(I)
    3 D(I)=A(N1/10**4,IB1/10**4)+A(IB2/10**4,N2/10**4)-A(N1/10**4
     *,N2/10**4)
      IF(D(IMIM).LE.D(I)) GO TO 2
      IMIM=I
    2 CONTINUE
      IF(IMIM.EQ.M-1) GO TO 18
      CALL ORDENA (T,IMIM,M-1,MO)
   18 T(IMIM)=IB1
      IF(IMIM+1.EQ.M) GO TO 19
      CALL ORDENA (T,IMIM+1,M,MO)
   19 T(IMIM+1)=IB2
      RETURN
      END
      SUBROUTINE DIXC(T,A,M,D,MO)
      INTEGER T(1),A(MO,MO),D(1),H
      M1=M-1
      L1=T(1)
      L2=T(2)
      H=T(M)
      IF(MOD(T(1)/100,100).EQ.T(2)/10000) GO TO 10
      IMIM=1
      GO TO 12
   10 IMIM=2
   12 D(IMIM)=A(L1/10**4,H/10**4)+A(H/10**4,L2/10**4)-A(L1/10**4,
     *L2/10**4)
      DO 2 I=IMIM+1,M1
      IF(MOD(T(I)/100,100).EQ.T(I+1)/10000) GO TO 2
      N1=T(I)
      N2=T(I+1)
      IF((I+1).NE.M) GO TO 3
      N2=T(I)
    3 D(I)=A(N1/10**4,H/10**4)+A(H/10**4,N2/10**4)-A(N1/10**4,N2/
     *10**4)
      IF(D(IMIM).LE.D(I)) GO TO 2
      IMIM=I
```

```
 2 CONTINUE
   IF(IMIM+1.EQ.M) GO TO 18
   CALL ORDENA (T,IMIM+1,M,MO)
18 T(IMIM+1)=M
   RETURN
   END
   SUBROUTINE IBRID(T,A,M,D,MO)
   INTEGER T(1),A(MO,MO),D(1)
   M1=M-2
   L1=T(1)
   L2=T(2)
   IB1=T(M-1)
   IB2=T(M)
   IF(MOD(T(1)/100,100).EQ.T(2)/10000) GO TO 10
   IMIM=1
   GO TO 12
10 IMIM=2
12 D(IMIM)=A(L1/10**4,IB1/10**4)+A(IB2/10**4,L2/10**4)-A(L1/10
  **4,L2/10**4)
   DO 2 I=IMIM+1,M1
   IF(MOD(T(I)/100,100).EQ.T(I+1)/10000) GO TO 2
   N1=T(I)
   N2=T(I+1)
   IF((I+1).NE.(M-1)) GO TO 3
   N2=T(1)
 3 D(I)=A(N1/10**4,IB1/10**4)+A(IB2/10**4,N2/10**4)-A(N1/10**4
  *,N2/10**4)
   IF( D(IMIM).LE.D(I)) GO TO 2
   IMIM=I
 2 CONTINUE
   IF(IMIM+1.EQ.M-1) GO TO 18
   CALL ORDENA(T,IMIM+1,M-1,MO)
18 T(IMIM+1)=IB1
   IF(IMIM+2.EQ.M) GO TO 19
   CALL ORDENA(T,IMIM+2,M,MO)
19 T(IMIM+2)=IB2
   RETURN
   END
```

```
      SUBROUTINE CONST(T,NZ,MV,IR,MO)
      INTEGER T(25,1),IR(1),MV(1)
C     O VETOR IR E PREENCHIDO COM O PRIMEIRO SUBPROBLEMA CONVEXO
   12 IF(MOD(T(1,1)/100,100).NE.0.AND.MOD(T(1,1)/100,100).EQ.T(1
     *,2)/10**4) GO TO 11
      CALL ROTI(T,MV,1,MO)
      GO TO 12
   11 DO 1 J=1,MV(1)
    1 IR(J)=T(1,J)
      N=MV(1)
   10 DO 2 L=2,NZ
      DO 3 M=1,N-1
      DO 4 NO=1,MV(L)-1
      IF(IR(M)/10**4.EQ.T(L,NO)/10**4.AND.IR(M+1)/10**4.EQ.T(L,NO
     *+1)/10**4) GO TO 13
      IF(IR(M+1)/10**4.EQ.T(L,NO)/10**4.AND.IR(M)/10**4.EQ.T(L,NO
     *+1)/10**4) GO TO 14
    4 CONTINUE
    3 CONTINUE
    2 CONTINUE
   13 IZ=L
      IF(NO.EQ.1.AND.(NO+1).EQ.2) GO TO 16
      DO 31 J=1,NO-1
      CALL ROTI(T,MV,IZ,MO)
   31 CONTINUE
   16 DO 35 NO=3,MV(IZ)
      CALL ORDENA(IR,M+NO-2,N+1,MO)
      IR(M+NO-2)=T(IZ,MV(IZ)-NO+3)
      N=N+1
      WRITE(5,/)N,M,NO
      WRITE(5,/) (IR(IL),IL=1,N)
   35 CONTINUE
      GO TO 32
   14 IZ=L
      IF(NO.EQ.1.AND.(NO+1).EQ.2) GO TO 15
      DO 36 I=1,NO-1
      CALL ROTI(T,MV,IZ,MO)
   36 CONTINUE
   15 DO 7 NO=3,MV(IZ)
      CALL ORDENA(IR,M+NO-2,N+1,MO)
      IR(M+NO-2)=T(IZ,NO)
      N=N+1
    7 CONTINUE
   32 IF(N.EQ.MO) GO TO 30
      DO 8 L=IZ,NZ-1
      DO 9 NO=1,MV(L+1)
      T(L,NO)=T(L+1,NO)
    9 CONTINUE
      MV(L)=MV(L+1)
    8 CONTINUE
```

```
      NZ=NZ-1
      GO TO 10
30    RETURN
      END
      SUBROUTINE ROTI(T,MV,L,MO)
      INTEGER T(25,1), MV(1)
      JW=T(L,1)
      DO 22 J=1,MV(L)-1
      T(L,J)=T(L,J+1)
22    CONTINUE
      T(L,MV(L))=JW
      RETURN
      END
```

B I B L I O G R A F I A

[1] BAZARAA, Mokhtar S. e GOODE, Jamie J, "The Traveling Salesman Problem: A Duality Approach", Mathematical Programming 13, 221-237, (1977).

[2] BELLMORE, M. e NEMHAUSER, G.L, "The Traveling Salesman Problem: A Survey", Operations Research 16, 538-558 (1968).

[3] BELLMORE, M. e MALONE, John C, "Pathology of Traveling Subtour Elimination Algorithms", Operations Research , 278-306 (1971).

[4] CROES, G.A., "A Method for Solving Traveling Salesman Problems", Operations Research 6, 791-812 (1958).

[5] DANTZIG, G.B., FULKERSON, D.R. e JOHNSON, S.M., "Solution of a Large Scale Traveling Salesman Problem", Operations Research 2, 393-410 (1954).

[6] FLOOD, M.M., "The Traveling Salesman Problem", Operations Research 4, 61-75 (1956).

[7] FORD, L.R.Jr. e FULKERSON, D.R., "Flows in Network", Princeton University Press (1962).

[8] GEOFFRION, A.M., "Lagrangian Relaxation And Its Use In Integer Programming", Mathematical Programming Study 2, 82-114 (1974).

[9] GILMORE, P.C. e GOMORY, R.E., "Sequencing a One State Variable Machine: A Solvable Case of the Traveling Salesman Problem", Operations Research 12, 655-679 (1964).

[10] HELD, M. e KARP, R.M., "The Traveling Salesman Problem and Minimum Spanning Trees", Operations Research 18, 1138-1162 (1970).

[11] HELD, M. e KARP, R.M., "The Traveling Salesman Problem and Minimum Spanning Trees , Part II",Mathematical Programming 1, 6-25 (1971).

[12] KARG, L.L. e THOMPSON, G.L., "A Heuristic Approach To Solving Traveling Salesman Problems", Management Science 10, 225-248 (1964).

[13] LIN, S., "Computer Solution of the Traveling Salesman Problem", Bell System Technical Journal 44, 2245-2269 (1965).

[14] LIN, S. e KERNIGHAN, B.W., "An Effective Heuristic Algorithm for the Traveling Salesman Problem", Operations Research 21, 498-516 (1973).

[15] LITTLE, John D.C., MURTY, K.G., SWEENEY, D.W. e KAUL, C. "An Algorithm for the Traveling Salesman Problem", Operations Research 11, 979-989 (1963).

[16] MILIOTIS, P., "Integer Programming Approaches to the Traveling Salesman Problem", Mathematical Programming 10, 367-378 (1976).

[17] MILLER, C.E., TUCKER, A.W. e ZEMLIN, P.A., "Integer Programming Formulation of Traveling Salesman Problems", J.ACM 7, 326-329 (1960).

[18] PICARD, J.C. e QUEYRANNE , "The Time Dependent Traveling Salesman Problem And Application to the Tardiness Problem in One-Machine Scheduling", Rapport Technique £ P 76-R-14 (1976).

[19] SALKIN, Harvey M., "Integer Programming", Addison-Wesley Publishing Company, (1975).

[20] WAGNER, H.M., "Principles of Operations Research with Applications to Managerial Decisions", 2 Prentice Hall, 1975.