UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

MÉTODOS PARA CONSTRUÇÃO DE ROTAS EULERIANAS EM GRAFOS MISTOS COM APLICAÇÃO NA DISTRIBUIÇÃO DE BENS E SERVIÇOS

MIGUEL ANTONIO BUENO DA COSTA

CAMPINA GRANDE - PARAÍBA
AGOSTO DE 1982

C837m Costa, Miguel Antonio Bueno da.
Métodos para construção de rotas eulerianas em grafos mistos com aplicação na distribuição de bens e serviços / Miguel Antonio Bueno da Costa. - Campina Grande, 1982.
96 f.

Dissertação (Mestrado em Ciências) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1982.
"Orientação : Prof. Dr. Peter Joachim Siegfried Brucker".
Referências.

1. Algoritmos - Grafos. 2. Rotas Eulerianas - Construção. 3. Rotas Eulerianas. 4. Dissertação - Ciências. I. Brucker, Peter Joachim Siegfried. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 004.021(043)

MÉTODOS PARA CONSTRUÇÃO DE ROTAS EULERIANAS EM GRAFOS MISTOS COM APLICAÇÃO NA DISTRIBUIÇÃO DE BENS E SERVIÇOS

MIGUEL ANTONIO BUENO DA COSTA

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DO CURSO DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

Peter J. S. Brucker

PETER JOACHIM SIEGFRIED BRUCKER - Ph.D.
- Presidente -

ANTONIO ILDEFONSO DE ALBUQUERQUE MELO - Ph.D.

EVILSON DE ARAÚJO BARROS - M.Sc.

CAMPINA GRANDE
ESTADO DA PARAÍBA - BRASIL
AGOSTO - 1982

Aos meus pais,

MIGUEL e RUTH;

A minha esposa e filho,

HELÉDIA E RAFAEL

RESUMO

O objetivo principal desse estudo é o desenvolvimento de métodos que solucionem o Problema do Carteiro Chinês em grafos mistos, visando aplicações diretas na distribuição de bens e serviços públicos. Devido a essas aplicações, foram levantadas algumas restrições associadas às leis de trânsito e manobras de veículos, quando essas se fazem necessários. São apresentadas soluções que adaptam essas restrições aos algoritmos à rota final.

O grafo original é modificado através de procedimentos heurísticos, que o transformam num grafo Euleriano, onde será aplicada a Rota Final. Todos os algoritmos, relativos a esses procedimentos e à rota final, são apresentados de uma forma estruturada, facilitando a sua compreensão e implementação.

Ao final são apresentados alguns aspectos que podem ser abordados futuramente, visando uma melhor adaptação do trabalho ao problema real.

ABSTRACT

The main purpose of this study is to develop a method of solving the Chinese Postman Problem in mixed graphs, aiming direct aplications in the distribution of assets and public services.

Due, to these aplications, some restrictions associated to the traffic laws and car maneuvering were developed, when they were found necessary. Solutions which adapt these restrictions to the algoritms related to the final route, are shown here.

All the algoritms related to these procedures and to the final route, are presented here in a estructural way, simplifying its comprehension and its implementation.

At the end, some aspects that could be approached in the future are shown, aiming for a better adaptation of the work to the real problem.

AGRADECIMENTOS

Agradeço a todos, que de forma direta ou indireta, colaboraram para que este trabalho se realizasse, e em especial ao Professor Dr. Peter Joachim Siegfried Brucker pela preciosa orientação e amizade.

Deixo aqui um agradecimento especial ao Professor M.Sc., Evilson de Araújo Barros e ao Professor Dr. Ildefonso de Albuquerque Melo, pelas importantes contribuições feitas na redação final deste trabalho.

Eu não poderia deixar de agradecer a minha esposa Helédia, pela paciência, compreensão e amizade, com que me acompanhou durante todo o desenrolar do trabalho.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

Nas cidades de médio e grande porte, as distribuições de bens e serviços, como coleta de lixo, entrega de correspondência, limpeza de ruas, entrega normal de gás e outras, são feitas de forma ainda insatisfatória. Esses serviços são operados, na maioria das vezes, pelo próprio coletor ou entregador de forma aleatória, encarecendo os custos de operação, principalmente quando são utilizados veículos motorizados.

Nesse estudo, o principal objetivo é fornecer métodos que solucionem o problema de se encontrar uma rota, numa rede formada pelas ruas de uma cidade, cujo custo total seja o menor possível.

Existe uma total identificação entre esse problema específico e o Problema do Carteiro Chinês (PCC) em grafos mistos, cuja finalidade é encontrar uma rota que percorra todos

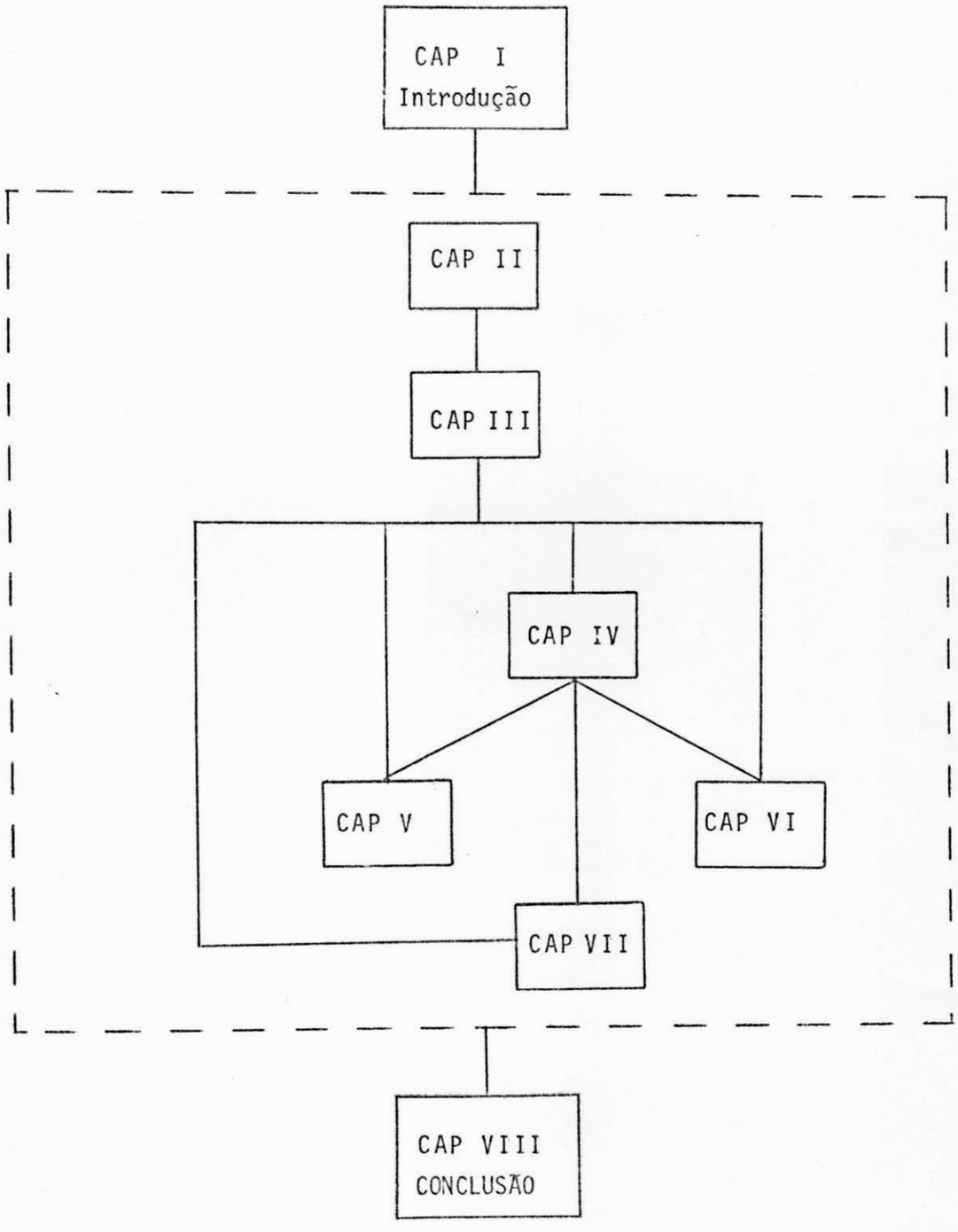
os arcos e ramos do grafo pelo menos uma vez. O PCC foi descrito pela primeira vez por Kuan Meiko [17], e algoritmos eficientes para sua solução foram apresentados por Edmonds [10], para grafos não direcionados e grafos totalmente direcionados. Papadimitriou [19], mostrou que, no caso de grafos mistos, o PCC pertence à classe dos problemas NP-Complete, para os quais ainda não foram encontradas soluções com ordem de complexidade polinomial. Foi proposta, por Edmonds e Johnson, uma solução heurística para o problema. Essa solução sofreu algumas alterações, feitas por Frederickson [14], e foi utilizada por Santos [20] na preparação do grafo para a rota. Essa preparação consiste na transformação do grafo original em um grafo de Euler, onde é possível traçar uma rota que atravesse todos os arcos e ramos exatamente uma vez.

Apesar do estudo de Santos ter se aprofundado no assunto fluxos em redes, tratou de uma maneira muito superficial o tema casamento em grafos gerais, além de apresentar os algoritmos de uma forma um pouco complicada e não operacional. Devido a esses motivos, uma preocupação na primeira parte desse trabalho foi, além de fazer uma abordagem mais profunda no assunto casamento, apresentar cada algoritmo de forma mais estruturada, facilitando sua compreensão.

Com relação à parte principal do trabalho, foram desenvolvidos algoritmos eficientes que encontram uma rota num grafo Euleriano. Como em muitos casos de distribuição de bens e serviços são envolvidos veículos, surgiram alguns problemas, de trânsito e de manobra, que precisaram ser compatibilizados com os algoritmos. Esses problemas estão associados a contorno proibido, retornos em U e "rotas" com a origem diferente do destino. Todos esses pontos foram estudados, e receberam soluções heurísticas particulares.

O trabalho está organizado da seguinte forma: uma base teórica para o desenvolvimento do assunto é fornecida nos capítulos 2 e 3; os capítulos 4, 5 e 6 apresentam os algoritmos de menores caminhos, fluxos em redes e casamento, através de uma discussão dos mesmos, descrições em Pidgin-Algol e análise de complexidade; no capítulo 7 são apresentadas as idéias e soluções para a rota e para todas as restrições; e por fim a conclusão, apresentação dos resultados e algumas sugestões de trabalhos futuros.

Relação entre os capítulos desse trabalho



CAPÍTULO II

NOÇÕES FUNDAMENTAIS SOBRE GRAFOS

Nesse capítulo serão apresentados alguns conceitos sobre grafos, que serão utilizados durante todo o transcorrer do trabalho.

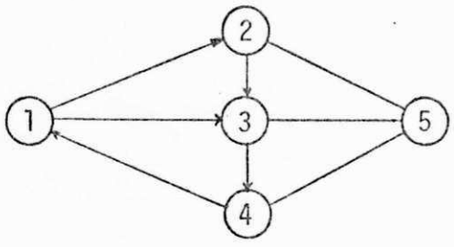
2.1 - Grafo Misto

Um grafo misto $G=(V,E,A)$ consiste de três conjuntos:

- Um conjunto finito de vértices $V=\{1,2,\ldots,n\}$;
- Um conjunto de ramos $E \subseteq \{\{i,j\} \mid i,j \in V\}$;
- Um conjunto de arcos $A \subseteq V \times V$.

Caso um dos dois conjuntos A ou E seja vazio, o grafo torna-se um caso particular do grafo misto G. O grafo $G=(V,A)$ é chamado grafo totalmente direcionado, e $G=(V,E)$ grafo não direcionado.

Exemplo 1: grafo misto



V={1,2,3,4,5}

E={{2,5},{3,5},{4,5}}

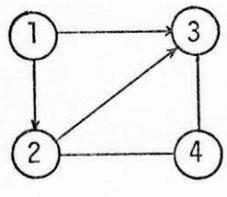
A={(1,2),(1,3),(2,3),(3,4),(4,1)}

2.2 - Incidência e Grau

Diz-se que um arco (ou ramo) é incidente com um vértice, quando esse vértice for um dos vértices terminais desse arco (ou ramo).

Grau de um vértice i, denotado por grau(i), é definido como sendo a soma dos arcos e ramos incidentes com ele. Define-se grau de entrada de um vértice i, representado por entr(i), como a soma de todos os arcos que possuem i como vértice final. O grau de saída, saída(i), corresponde à soma dos arcos que possuem i como vértice inicial.

Exemplo 2:



grau(1)=grau(4)=2

grau(2)=grau(3)=3

entr(1)=entr(4)=0 saída(3)=saída(4)=0

entr(2)=1 saída(1)=2

entr(3)=2 saída(2)=1

2.3 - Deficit

Chama-se deficit de um vértice, representado por def(i), a diferença entre entr(i) e saída(i).

Com relação ao exemplo anterior tem-se:

def(1)= -2

def(2)= 0

def(3)= 2

def(4)= 0

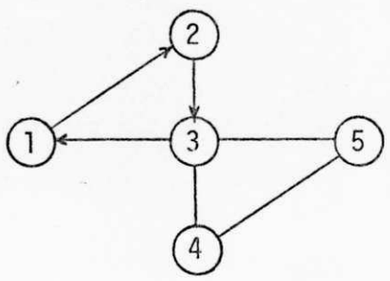
2.4 - Caminho, Circuito, Cadeia

Caminho com relação a um grafo G=(V,E,A), é uma sequência finita $ca=e_1, e_2,\ldots,e_k, e_{k+1},\ldots,e_n$, onde $e_k=(i_k,i_{k+1})$ ou $e_k=\{i_k,i_{k+1}\}$. Quando $i_1=i_{n+1}$ esse caminho é chamado circuito.

Dá-se o nome de cadeia a um caminho onde não são consideradas as orientações dos arcos.

Exemplo 3:

G=



caminho : (1,2),(2,3),{3,5},{5,4}

circuito : (1,2),(2,3),(3,1)

cadeia : (3,1),{3,5},{5,4}

2.5 - Rota

Define-se rota como sendo um circuito que contém todos os arcos e ramos do grafo. No exemplo anterior tem-se:

r=(1,2),(2,3),{3,4},{4,5},{5,3},(3,1)

2.6 - Rota de Euler (Euleriana)

É uma rota onde todos os eε(EUA) são atravessados exatamente uma vez.

2.7 - Grafo de Euler

É um grafo que admite uma rota de Euler. O grafo do exemplo 3 admite essa rota, portanto é um grafo de Euler.

2.8 - Grafo Conectado

Um grafo G é conectado, se existir pelo menos uma cadeia entre todos os pares de vértices em G.

Os grafos associados aos exemplos 1, 2 e 3 são todos conectados.

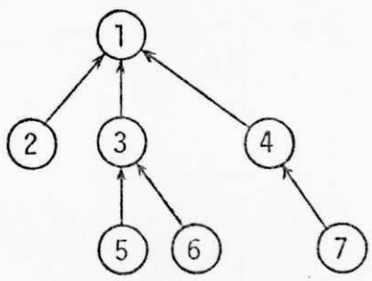
2.9 - Árvore

Uma árvore é um grafo conectado, com as seguintes características:

(i) o vértice r, chamado raiz, possui saída(r)=0;

(ii) todos os outros vértices i≠r, possuem saída(i)=1.

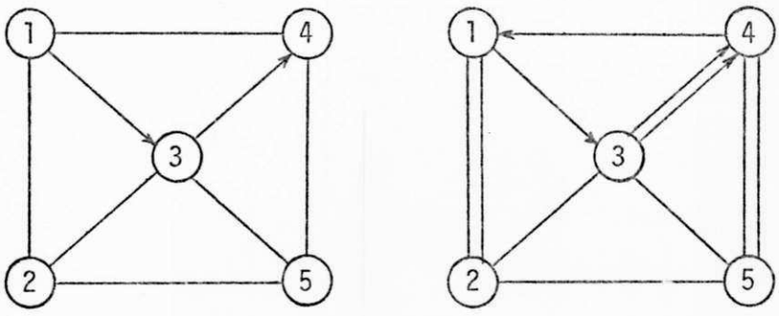
Exemplo 4:



2.10 - Aumentação

Um grafo G'=(V,E',A') é uma aumentação de G=(V,E,A), se G' for um grafo construído a partir de G, pelo direcionamento de alguns ramos e repetições de arcos e ramos.

Exemplo 5:



Uma aumentação G' é chamada aumentação de Euler, se G' for uma grafo Euleriano.

Associando a cada $e \varepsilon (E \cup A)$ um custo c(e), o custo total de uma aumentação será $c(E' \cup A') = \sum_{e \varepsilon (E' \cup A')} c(e)$

O Problema do Carteiro Chinês corresponde a uma aumentação G' de Euler do grafo G, sendo o custo da rota igual ao custo de G'.

CAPÍTULO III

PROCEDIMENTOS HEURÍSTICOS PARA O PROBLEMA

Para se atingir o objetivo final, que é a geração da rota, é preciso utilizar vários métodos e procedimentos intermediários que resolvam os problemas mais imediatos. Para esses problemas, que serão denominados subproblemas, serão geradas subrotinas que farão as devidas modificações no grafo original, preparando-o para que seja possível a implementação da rota final.

A preparação do grafo é feita através de três subrotinas chamadas MIXED A, MIXED 1A e MIXED 2A. Essas subrotinas necessitarão de métodos que resolvam problemas de Menores Caminhos, Fluxos em Redes e Casamento em Grafos Gerais. Uma preocupação nesse capítulo será detalhar cada um desses subproblemas, fornecendo conceitos e teoremas que serão necessários para a compreensão dos algoritmos descritos nos capítulos 4, 5

e 6. Por fim serão apresentados os algoritmos MIXED A, MIXED 1A e MIXED 2A, responsáveis, como já foi citado, pela preparação final do grafo no qual será aplicado o algoritmo da ROTA FINAL.

3.1 - Menores Caminhos

Dado um grafo G=(V,A), com a matriz C=[c(i,j)] correspondente aos custos dos arcos (i,j), o problema dos menores caminhos se preocupa em encontrar um caminho entre dois vértices pré-definidos, de tal forma que a soma dos custos associados aos arcos pertencentes ao caminho seja mínima. É importante frisar que esses custos c(i,j) podem ser positivos, negativos ou zero, com a restrição de que não ocorram circuitos negativos no grafo. Essa restrição é fácil de ser compreendida, pois o objetivo sendo o de encontrar o menor caminho entre dois vértices, não poderá deparar com um circuito negativo, pois nesse caso o método tenderia a não mais sair desse circuito e o valor do caminho iria para menos infinito.

Num caso particular pode ser de interesse encontrar não apenas o menor caminho entre dois vértices pré-definidos, e sim todos os menores caminhos entre quaisquer dois vértices pertencentes ao grafo. Isso pode ser feito de uma maneira relativamente simples, aplicando um método que resolva o problema de encontrar o caminho de um vértice para todos os outros, n vezes. Uma outra opção é a utilização de um método que resolva o problema diretamente, encontrando os menores caminhos entre todos os pares de vértices do grafo. Algoritmos para esse caso e os anteriores serão apresentados e discutidos no capítulo seguinte.

É bom observar também que os custos c(i,j), associados aos arcos (i,j), não são necessariamente menores do que

c(i,k)+c(k,j), para todos i,j,k, pois nesse caso o menor caminho entre i e j seria o arco (i,j), e a finalidade de se encontrar os menores caminhos perderia todo o sentido.

Outra importante observação é que, na busca da solução para esse subproblema, serão utilizados grafos direcionados. Isso pode ser considerado, pois qualquer ramo {i,j} que surgir, poderá ser transformado em dois arcos (i,j) e (j,i), com c(i,j)=c(j,i). A única restrição para essa transformação é a de que o custo c(i,j)$\geq$ 0, pois caso contrário poderão ocorrer circuitos negativos. Nesse estudo isso não será problema, pois utilizando distâncias, os custos serão sempre não negativos.

3.2 - Fluxos em Redes

Um subproblema bastante delicado e muito importante para esse trabalho diz respeito ao problema de fluxos em redes. Dado um grafo G=(V,A), com capacidades q(i,j), associadas aos arcos (i,j), que controlam a máxima quantidade de fluxo que pode atravessar cada arco (i,j), um dos problemas mais comuns relacionados com fluxos, é o de se determinar o valor do máximo fluxo que pode ser transportado de um vértice inicial, denominado fonte, para um vértice terminal chamado sumidouro. Um outro tipo de problema, cuja técnica de resolução será efetivamente utilizada nesse trabalho, consiste em, sendo fornecido um fluxo previamente estabelecido, determinar um fluxo de mínimo custo que percorra o grafo da fonte ao sumidouro.

Para esses tipos de problemas algumas técnicas de resolução foram desenvolvidas por Ford e Fulkerson [13] e Busacker e Gowen [2], e serão descritas no capítulo 5.

3.2.1 - Fluxos Máximos

Tem-se um grafo $G=(V,A)$, com capacidade $q(i,j)$ atribuídas aos arcos (i,j), um vértice fonte $s \in V$ e um vértice sumidouro $t \in V$. Pretende-se encontrar o fluxo máximo de s para t, sendo $x(i,j)$ a quantidade de fluxo que atravessa o arco (i,j). Torna-se claro que $0 \leq x(i,j) \leq c(i,j)$.

Considerando que, para todos os nós $j \neq s,t$, a quantidade de fluxo que chega no vértice é igual a que sai, torna-se necessário que $\sum_i x(i,j)$ seja igual a $\sum_i x(j,i)$, fazendo com que

$$\sum_i x(i,j) - \sum_i x(j,i) = \begin{cases} -f, & \text{se } j=s \\ 0, & \text{se } j \neq s,t \\ f, & \text{se } j=t \end{cases}$$

Portanto, sendo f o fluxo que satisfaz as restrições acima, pode-se enunciar o problema do fluxo máximo através da programação linear, da seguinte forma:

Maximizar f

Sujeito a:

$$\sum_i x(i,j) - \sum_i x(j,i) = \begin{cases} -f, & \text{se } j=s \\ 0, & \text{se } j \neq s,t \\ f, & \text{se } j=t \end{cases}$$

$$x(i,j) \leq q(i,j)$$

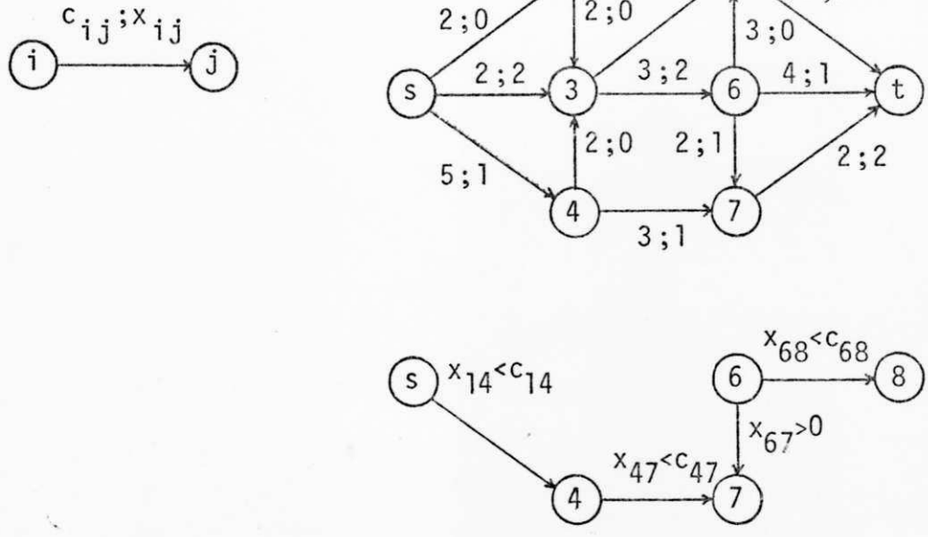
$$x(i,j) \geq 0$$

Serão introduzidas agora definições que são utilizadas em alguns teoremas citados a seguir.

Definição III.1 - Caminho com Fluxo Aumentado

Considere um caminho P não direcionado ligando s com t. Diz-se que P é um caminho com fluxo aumentado, com respeito a um fluxo pré-determinado f=x(i,j), se $x(i,j)<q(i,j)$ para todos os arcos (i,j) direcionados no sentido de s para t, e $x(i,j)>0$ para os arcos direcionados em sentido inverso.

Exemplo:



Caminho com Fluxo Aumentado

Definição III.2 - Conjunto de Corte - (S,T)

Se o conjunto dos vértices V de um grafo for dividido em dois subconjuntos S e T, de tal forma que $s \in S$ e $t \in T$, define-se como sendo um Conjunto de Corte-(S,T) o conjunto formado por todos os arcos (i,j) direcionados no sentido de S para T. E define-se como capacidade do conjunto de corte-(S,T), como sendo $c(S,T)= \sum_{i \in S} \sum_{j \in T} c(i,j)$.

De posse das definições acima serão introduzidos três importantes teoremas, utilizados no desenvolvimento de métodos que tratam do problema de fluxos em redes. Esses teoremas

serão apenas citados pois suas provas são relativamente simples e podem ser encontradas na referência Lawler [16].

Teorema I

Um fluxo é máximo se e somente se o grafo não permitir a construção de um caminho de fluxo aumentado, de s para t.

Teorema II

Se todas as capacidades, associadas aos arcos , são inteiras, então existe um fluxo máximo inteiro.

Teorema III

O valor do fluxo máximo, de s para t, é igual a capacidade mínima de um conjunto de corte-(S,T).

3.2.2 - Fluxo de Mínimo Custo

Considerando um grafo G=(V,A), com custos c(i,j) e capacidades q(i,j) associadas aos arcos. O problema do fluxo de custo mínimo, segundo a programação linear, seguirá o seguinte modelo:

$$\text{Minimizar} \quad z = \sum_{(i,j)\in A} c(i,j) \cdot x(i,j)$$

$$\text{Sujeito a:} \quad \sum_{j} x(i,j) - \sum_{j} x(j,i) = 0 \quad \text{para todo } i \neq s,t$$

$$\sum_{j} x(s,j) = f$$

$$x(i,j) \leq q(i,j)$$

$$x(i,j) \geq$$

Na realidade o que se pretende é obter o fluxo de mínimo custo, de um vértice fonte para um sumidouro, respeitando um fluxo padrão f pré-estabelecido.

Definição III.3 - Custo de um Caminho com Fluxo Aumentado

Considerando somente os arcos (i,j) pertencentes ao caminho com fluxo aumentado, define-se o custo desse caminho como sendo a soma dos custos dos arcos direcionados no sentido de s para t, menos os custos dos arcos direcionados em sentido contrário.

Definição III.4 - Circuito de Fluxo Aumentado

É um caminho com fluxo aumentado fechado. Seu custo é calculado de maneira análoga ao caminho aumentado.

Após essas definições é possível apresentar dois teoremas, utilizados na elaboração do algoritmo de Busacker e Gowen, cujas provas se encontram nas referências Lawler [16] e Christofides [3].

Teorema IV

Um fluxo de valor f é de mínimo custo sse ele não admitir um circuito de fluxo aumentado, com custo negativo.

Teorema V

Um acréscimo & em um fluxo f de custo mínimo, ao longo de um caminho com fluxo aumentado de custo mínimo, produz um fluxo de custo mínimo de valor f+&.

3.3 - Casamento em Grafos Gerais

Considere-se um grafo G=(V,E), cujo conjunto dos vértices V pode ser dividido em dois subconjuntos S e T, de tal forma que todo ramo {i,j} possua um de seus vértices terminais em S e o outro em T. Ao grafo G=(S,T,E) dá-se o nome de Grafo Bipartido. Uma aplicação prática para esse tipo de grafo é a atribuição de elementos pertencentes a S a elementos de T, por exemplo, atribuir funções à máquinas ou máquinas a homens. Assim através de uma combinação entre os elementos, procura-se encontrar a que produz melhor solução.

É chamado de casamento um subconjunto $M \subseteq E$ tal que, se forem tomados dois ramos quaisquer pertencentes a M, não existe um vértice j comum a ambos. Os vértices incidentes com algum ramo pertencente ao casamento são denominados casados, caso contrário são chamados expostos.

Nesse estudo o interesse é trabalhar com um grafo mais geral, onde pode ser possível ou não a divisão do conjunto de vértices em dois subconjuntos, respeitando a condição para ser bipartido. Com relação a casamentos em grafos gerais, o interesse desse trabalho reside na solução do casamento perfeito de mínimo custo.

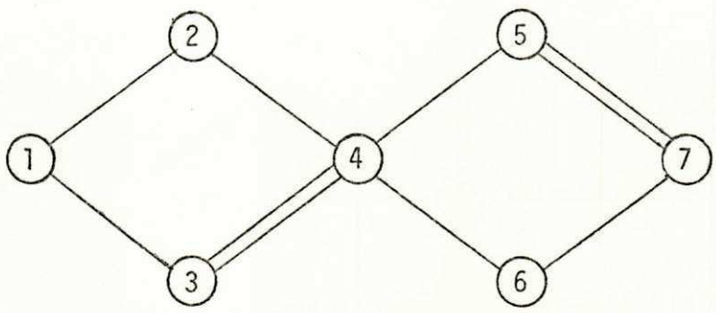
Serão apresentadas agora algumas definições e teoremas, que serão utilizados quando o trabalho tratar especificamente do casamento.

Definição III.6 - Caminho Alternado

Seja $M \subseteq E$ um casamento no grafo G=(V,E).

Caminho alternado é um caminho elementar cujos ramos estão, alternadamente, em M e não em M.

Exemplo:



1 — 3 ═ 4 — 5 ═ 7

Definição III.7 - Caminho Aumentado

É um caminho alternado, onde os vértices inicial e final estão expostos.

Exemplo: Em relação a figura anterior, tem-se um caminho aumentado dado por

1 — 3 — 4 — 5 ═ 7 — 6

Definição III.8 - Árvore Alternada

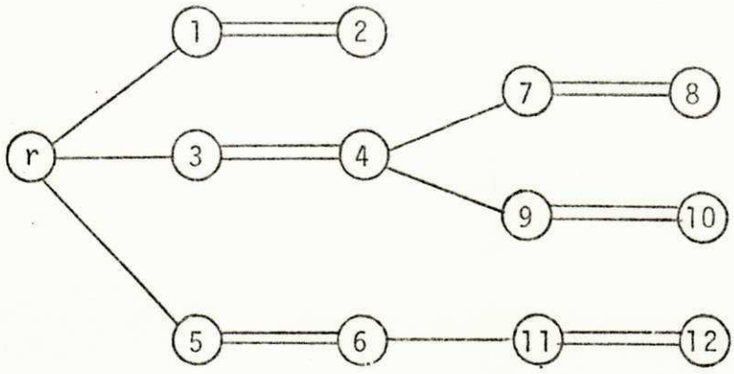
É uma árvore T, que segue as seguintes restrições:

(i) Um dos vértices de T, chamado raiz, é um vértice exposto;

(ii) Todos os caminhos inicializados pela raiz são caminhos alternados;

(iii) Todos os caminhos máximos, iniciados a partir da raiz, são de cardinalidade par.

Exemplo:



Árvore Alternada

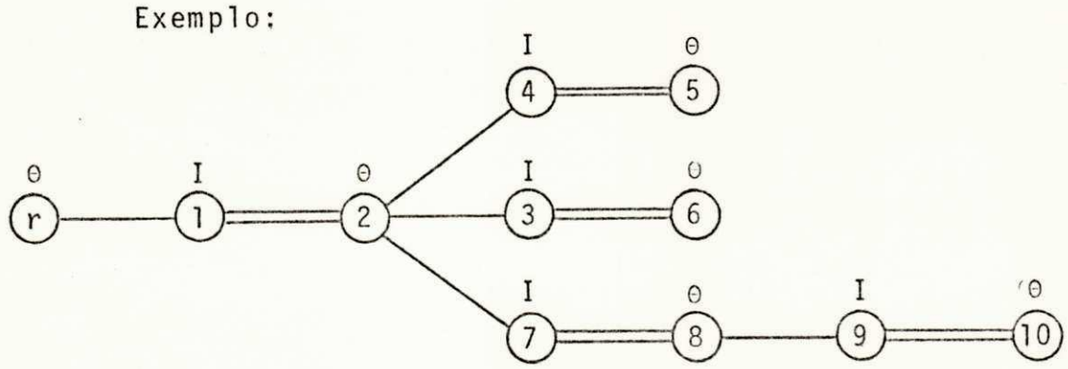
Definição III.9 - Rotulação de Árvore

Nos algoritmos que serão mostrados no capítulo 6, será muito utilizada a rotulação de árvores, usada na identificação dos caminhos aumentados. Essa rotulação é feita da seguinte forma:

(i) A raiz da árvore é rotulada como "saída", e será usado como símbolo desse rótulo a letra θ;

(ii) Para qualquer caminho, iniciado pela raiz, os vértices são rotulados, alternadamente, "saída" e "entrada". O rótulo "entrada" será simbolizado por I.

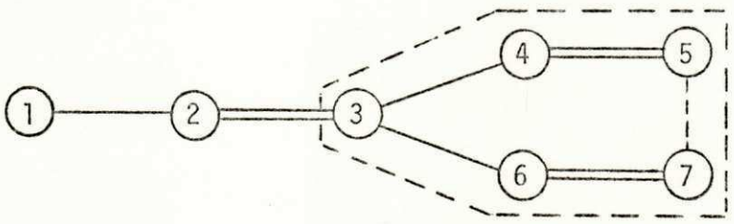
Exemplo:



Definição III.10 - Flores

Uma flor é um caminho aumentado, onde o vértice exposto inicial coincide com o vértice exposto final, formando um circuito de cardinalidade ímpar.

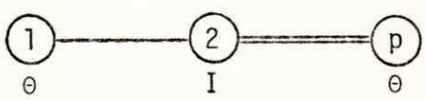
Exemplo:



flor

Os algoritmos de casamento, segundo uma idéia de Edmonds e Johnson [11], utilizam as flores para simplificar o grafo com o qual estão trabalhando. Essa simplificação é feita através do encolhimento das mesmas, onde a flor é substituída por um pseudo-vértice, diminuindo portanto o número de vértices e ramos da árvore. Após o encolhimento, desenvolvendo-se a árvore, poderão surgir novas flores que englobem as primeiras, que são chamadas de flores mais externas.

Para a árvore do exemplo anterior, o encolhimento da flor a transformaria em uma outra, que seria a seguinte:



onde p é o pseudo-vértice.

Feita a apresentação desses conceitos, serão citados agora dois teoremas que praticamente formam a base dos algoritmos associados ao casamento perfeito de mínimo custo. Como nos teoremas citados anteriormente, serão evitadas as provas desses últimos, pois fogem ao escopo desse trabalho. Elas podem

ser encontradas na referência Christofide [3] .

Teorema VI

Um casamento M é de máxima cardinalidade se não existir nenhum caminho aumentado em G, relativo a M.

Teorema VII

Sendo B uma flor, portanto um conjunto ímpar V(B) de vértices, e se v for qualquer vértice pertencente a V(B), então existe um casamento de máxima cardinalidade que deixa v exposto.

3.4 - Condições para a Existência de uma Rota em Grafos Mistos

O problema do Carteiro Chinês em grafos mistos (PCCGM), ao contrário do mesmo em grafos direcionados e não direcionados, pertence a uma classe de problemas onde é praticamente impossível a existência de um algoritmo polinomial, denominada classe dos problemas NP-Complete. Isso foi mostrado por Papadimitriu [19]

O problema de se encontrar uma rota de Euler em um grafo não direcionado é resolvido através de um algoritmo eficiente, apresentado por Edmonds [9], que utiliza como condições necessárias e suficientes para a resolução, o fato de todos os vértices do grafo possuir um número par de ramos incidentes sobre si.

Edmonds e Johnson [10] mostraram também que a condição necessária e suficiente para a existência de uma rota de Euler em um grafo direcionado, é que, para todo vértice pertencente ao grafo, seu grau de entrada seja igual ao de saída.

Considerando um grafo misto, Ford e Fulkerson [13] mostraram que se ambas as condições citadas anteriormente ocorrerem, então é possível encontrar uma rota de Euler nesse grafo. Levando em conta esse fato, Edmonds e Johnson [11] propuseram um método para a resolução do PCCGM, que consiste em se fazer uma modificação no grafo G=(V,E,A) original, através do direcionamento de alguns ramos e da repetição de certos ramos e arcos.

É claro que a modificação do grafo original deve ser feita de tal forma, que o aumento do custo associado a essa mudança seja o mínimo possível. Para isso, e visando assegurar a existência dessa modificação, o seguinte teorema será muito útil, e sua prova se encontra em Peter Brucker [1].

Teorema VIII

Seja G=(V,E,A) um grafo misto conectado. Se existir uma modificação de Euler G' de G, também existirá uma modificação G'' que satisfaz:

(i) Grau(i) é par para todo i∈V;

(ii) Entr(i)=Saída(i) para todo i∈V.

Se existir uma modificação satisfazendo (i) e (ii), então existe uma modificação de Euler. A modificação de Euler é de mínimo custo sse a correspondente modificação que satisfaz (i) e (ii) for de mínimo custo.

A modificação que satisfaz (i) é chamada de aumentação grau-par, e a que satisfaz (ii) aumentação entrada-igual-saída.

3.4.1 - Modificação Grau-Par

1. - Encontre em G o conjunto $V' \subseteq V$ de vértices de grau ímpar. Sabe-se que $|V'|$ é par, e isso pode ser verificado por teorema apresentado e provado em Deo [5].

2. - Para todos os $i, j \in V'$, encontre uma cadeia de custo mínimo $c(i,j)$ entre i e j.

3. - Encontre um casamento de mínimo custo M no grafo não direcionado, com conjunto de vértices V' e distâncias $c(i,j)$, que cobrem todos os vértices.

4. - Para todos $\{i,j\} \in M$, duplique todos os ramos e arcos pertencentes a cadeia de mínimo custo entre i e j.

Como as cadeias estão ligando vértices ímpares i e j, com a duplicação dos arcos e ramos dessa cadeia, os vértices i e j serão incrementados de uma unidade e se tornarão de grau par, enquanto que os vértices intermediários serão incrementados de duas unidades. Portanto, ao fim do procedimento citado, todos os vértices do grafo terão grau par.

3.4.2 - Modificação Entrada-Igual-Saída

Antes de mostrar como deve ser feita essa modificação, serão necessários os seguintes procedimentos preliminares:

(i) Para todos os arcos $e \in A$, associe a capacidade $u(e)=\infty$ e o custo $c(e)$ igual ao custo original.

(ii) Troque cada ramo $\{i,j\}\varepsilon E$ por três arcos $\vec{e}$, $\overleftarrow{e}$, $\overleftarrow{e}'$, onde $\vec{e}$ deve ser orientado de uma forma arbitrária, e os outros dois serão, então, orientados no sentido contrário. Os custos e as capacidades serão as seguintes:

$$c(\vec{e})=c(\overleftarrow{e})=c(e)$$
$$c(\overleftarrow{e}')=0$$
$$u(\vec{e})=u(\overleftarrow{e})=\infty$$
$$u(\overleftarrow{e}')=2$$

Um problema de fluxos em redes é associado ao grafo $G^*=(V,D,c,u)$, onde V é o conjunto dos vértices, D o conjunto dos arcos, c a função custo e u a função capacidade, ambas definidas de $D\rightarrow V$.

O problema é o seguinte:

$$\text{Minimize} \quad \sum_{e\varepsilon D} c(e) \cdot x(e)$$

$$\text{Sujeito a:} \quad \sum_{e\varepsilon Entr(i)} x(e) - \sum_{e\varepsilon Saída(i)} x(e)=def(i)$$

para todo $i\varepsilon V$

$x(e)\geq 0$ e inteiro

(iii) Troque cada ramo $e\varepsilon A$ por $x(e)+1$ arcos paralelos com ele.

(iv) Se $e\varepsilon E$ somente três considerações poderão ser feitas:

- Se $x(\overleftarrow{e}')=0$, então trocamos e por $x(e)+1$ arco paralelo com $\vec{e}$;

- Se $x(\overleftarrow{e}')=2$, então trocamos e por $x(e)+1$ arco paralelo com $\overleftarrow{e}$;
- Se $x(\overleftarrow{e}')=1$, não alteramos e.

Modificação Entrada-Igual-Saída

1. - Resolva o problema de fluxo definido por (i) e (ii);
2. - Utilize a solução do problema de fluxo para construir uma modificação Entrada-Igual-Saída, de acordo com os procedimentos (iii) e (iv).

A aplicação dessa modificação resultará um grafo G' direcionado, com entr(i)=saída(i) para todo $i \in V$.

3.5 - Algoritmos Aproximativos

O método proposto por Edmonds e Johnson [11] foi analisado por Frederickson [14], que concluiu que a razão entre a rota gerada pelo algoritmo e a rota ótima era menor ou igual a 2. Com algumas modificações Frederickson [14] melhorou o método, diminuindo a razão para 5/3. Serão mostrados agora os algoritmos aproximativos utilizados.

MIXED 1A

1. - Construir uma aumentação Grau-par G' do grafo G;
2. - Construir uma aumentação Entrada-Igual-Saída G'' do grafo G'.

Esse algoritmo resolve o problema, pois a aumentação Entrada-Igual-Saída mantém o grau par para todos os vérti

ces. Pode-se também inverter a posição das modificações dentro do algoritmo Mixed 1A, e é exatamente essa a idéia do algoritmo Mixed 2A.

MIXED 2A

1. - Construir uma aumentação Entrada-Igual-Saída G' do grafo G;
2. - Construir uma aumentação Grau-par G'' do grafo G', no subgrafo de G' contendo somente os ramos.

A filosofia do algoritmo principal é chamar as duas subrotinas Mixed 1A e Mixed 2A, que na maioria das vezes fornecerão resultados diferentes, fazer uma análise dos resultados e escolher o melhor.

MIXED A

1.- Call MIXED 1A
2.- Call MIXED 2A
3.- Compare as soluções e escolha a melhor.

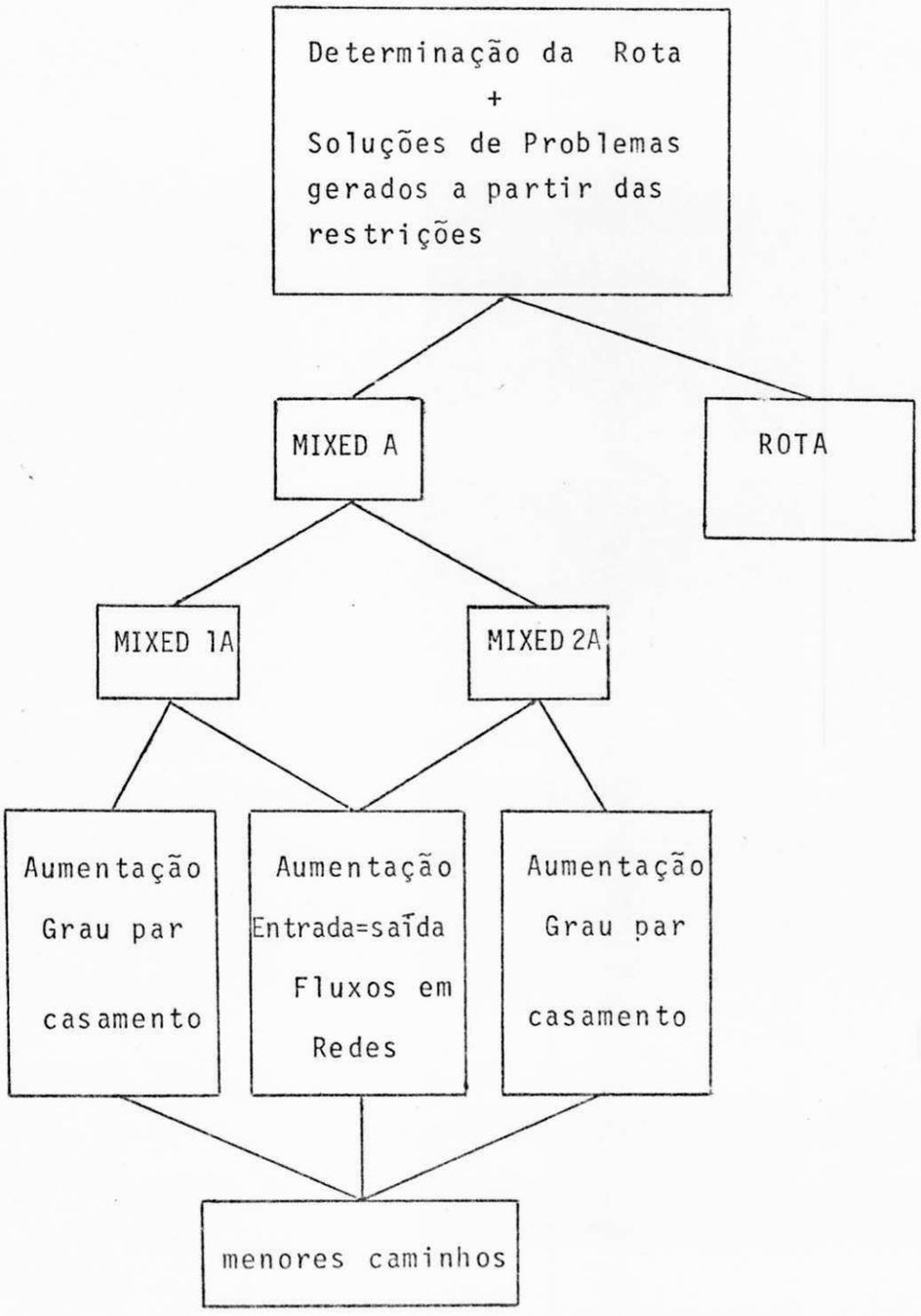
Basicamente é essa a idéia do trabalho inicial , que fornecerá como resultado um grafo G'', modificado em relação ao original G, com grau(i) par e entr(i)=saída(i) para todos os $i \in V$. Portanto trata-se de um grafo Euleriano, onde é possível traçar uma rota que resolva o Problema do Carteiro Chinês em Grafos Mistos.

Nesse estudo, por se tratar de uma continuidade de um trabalho anterior, não será feita uma abordagem mais profunda no assunto que trata da preparação do grafo de Euler.

Maiores detalhes podem ser encontrados no estudo desenvolvido por Santos [20].

3.6 - Fluxograma Geral do Trabalho

Os procedimentos preliminares, os heurísticos e a rota, juntamente com suas restrições, estão interligadas de acordo com o seguinte quadro:



CAPÍTULO IV

ALGORITMOS PARA A DETERMINAÇÃO DOS MENORES CAMINHOS

Trabalhando com uma rede G=(V,A,c), conectada e valorada, serão mostrados os principais algoritmos existentes, que solucionam o problema de se encontrar o menor caminho entre:

(i) Dois vértices específicos s e t pertencentes a V;

(ii) Um vértice $s \varepsilon V$ e todos os outros $i \varepsilon V$;

(iii) Quaisquer dois pares de vértices i,j pertencentes a V.

A apresentação de cada algoritmo seguirá a seguinte ordem: a filosofia do algoritmo, sua descrição em Pidgin Algol e uma discussão sobre a complexidade do mesmo.

Neste capítulo serão abordados dois algoritmos em especial, o de Dijkstra, que pode ser aplicado para qualquer

um dos problemas (i), (ii) ou(iii), com a restrição dos custos serem não negativos, e o de Floyd-Warshall que resolve o (iii) sem restrição para os custos. Será mostrada também uma nova versão do algoritmo de Dijkstra, que trabalha com um caso específico de especial interesse para esse trabalho.

4.1 - Algoritmo de Dijkstra [9]

A idéia desse algoritmo é, inicialmente, atribuir a todos os vértices, com exceção da origem, o rótulo temporário infinito. O vértice origem s recebe o rótulo permanente zero. Um conjunto S é criado para conter todos os vértices temporários. O passo principal do algoritmo é formado por um comando "While", onde o objetivo final é transformar todos os vértices temporários em permanentes. Quando isso ocorre, tem-se todos os menores caminhos entre o vértice inicial e todos os outros. No final do algoritmo é feita uma observação para o caso de se desejar apenas o menor caminho entre dois vértices pré-determinados.

Algoritmo

1. $S \leftarrow \{1,2,\ldots,n\}$;
2. $L(1) \leftarrow 0$;
3. FOR i ← 2 UNTIL n DO $L(i) \leftarrow \infty$;
4. WHILE $S \neq \emptyset$ DO
   Begin
5. Encontre $i \in S$ com $L(i)=\min\{L(j) \mid j \in S\}$;
6. IF $L(i)=\infty$ THEN Stop;
7. $S \leftarrow S \setminus \{i\}$;
8. FOR (todos os sucessores de i) DO $L(j) \leftarrow \min\{L(j), L(i)+$

$+ c_{ij}\}$

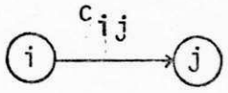
End

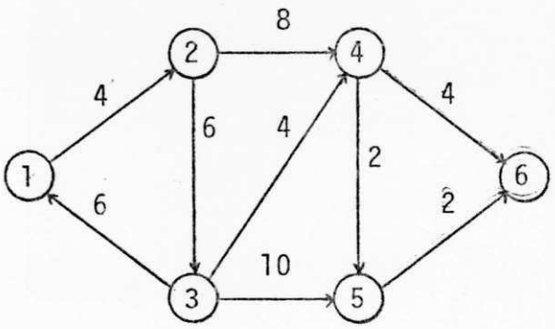
Obs.: (1) - Se for de interesse somente o menor caminho de 1 para t, basta trocar o comando 6. por:
IF (L(i)=∞ ou i=t) THEN Stop;

(2) - Se for desejado os menores caminhos entre todos os pares de vértices, basta aplicar o algoritmo n vezes, trocando sempre o vértice inicial.

4.1.1 - Exemplo

Dado o grafo G=(V,A,c) abaixo, deseja-se encontrar os menores caminhos entre 1 e todos os outros vértices.





- O algoritmo inicializa a solução montando o conjunto
  S = {1,2,3,4,5,6}
  e atribuindo os rótulos L(1)=0; L(j)=∞ para j≠1
- Escolhe-se um i∈S, tal que L(i)=min{L(j)|j∈S}
  No caso é L(1)=0
- 1 é eliminado do conjunto S ⟶ S={2,3,4,5,6}
- Para todos os sucessores de 1, no caso somente o vértice 2, calcula-se L(2)=min{∞,0+4}=4
- É escolhido um novo i∈S, com L(i)=min{L(j)|j∈S}; o processo se repete: i=2

- Obtêm-se então um novo S={3,4,5,6}
  L(3)=10; L(4)=12; e o processo continua até o conjunto se tornar vazio.

i=3	i=4	i=5
S={4,5,6}	S={5,6}	S={6}
L(5)=20	L(5)=14;L(6)=16	L(6)=16

- Nesse ponto tem-se as seguintes distâncias:

j	d_{1j}
2	4
3	10
4	12
5	14
6	16

4.1.2 - Complexidade

Os cálculos computacionais críticos em termos de ordem de complexidade desse algoritmo, se prendem a um comando "While" que é acionado n vezes. Cada vez que o comando é ativado, um vértice é escolhido e examinado, sendo rotulados todos os seus sucessores que ainda são temporários. Esse fato exige, num caso extremo, n adições e n comparações. Todo esse procedimento faz com que a ordem de complexidade do algoritmo seja $O(n^2)$.

Se for desejado encontrar os menores caminhos entre quaisquer dois vértices pertencentes ao grafo, pode-se aplicar o algoritmo n vezes, o que faz com que a complexidade se

torne de ordem $O(n^3)$.

4.2 - Algoritmo de Floyd-Warshall [12]

Será apresentado agora um algoritmo específico para calcular os menores caminhos entre todos os pares de vértices de um grafo G=(V,A,c), conectado e valorado.

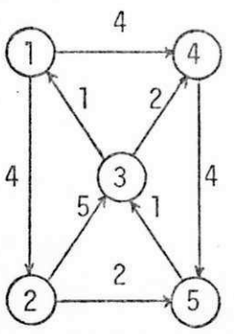
O algoritmo de Floyd-Warshall é um método iterativo, que modifica matrizes formadas a partir da matriz distância relativa ao grafo original. Cada matriz gerada possui valores menores ou no máximo iguais aos seus correspondentes anteriores. Com isso o método pesquisa novos caminhos, comparando-os com os já analisados, e no final fornece uma matriz com os menores caminhos entre quaisquer dois vértices do grafo.

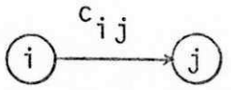
O Algoritmo

1. FOR i ← 1 UNTIL n DO
2. FOR j ← 1 UNTIL n DO
3. IF $((i,j)\varepsilon A)$ THEN $d_{ij} \leftarrow c_{ij}$; ELSE $d_{ij} \leftarrow \infty$
4. FOR i ← 1 UNTIL n DO $d_{ii} \leftarrow 0$;
5. FOR k ← 1 UNTIL n DO
6. FOR i ← 1 UNTIL n DO
7. FOR j ← 1 UNTIL n DO
8. $d_{ij} \leftarrow \min\{d_{ij}, d_{ik}, d_{kj}\}$

4.2.1 - Exemplo

- Deseja-se encontrar os comprimentos dos menores caminhos entre quaisquer pares de vértices do seguinte grafo valorado:





D=		1	2	3	4	5
	1	0	4	∞	4	∞
	2	∞	0	5	∞	2
	3	1	∞	0	2	∞
	4	∞	∞	∞	0	4
	5	∞	∞	1	∞	0

- Pode-se pensar nesse algoritmo como um único passo geral, onde todos os elementos d_{ij}, escolhidos num passo k, são comparados com a soma dos valores d_{ik} e d_{kj}. Se $d_{ij} \leq d_{ik}+d_{kj}$, d_{ij} permanece inalterado, caso contrário $d_{ij} \leftarrow d_{ik}+d_{kj}$.

Para k=1

- Nesta situação são analisados todos os elementos d_{ij}, que são comparados com a soma dos respectivos d_{i1} e d_{1j}.

$d_{11}=\min\{0,0+0\}=0$

$d_{12}=\min\{4,0+4\}=4$

$d_{13}=\min\{\infty,0+\infty\}=\infty$

$d_{14}=\min\{4,0+4\}=4$

$d_{15}=\min\{\infty,0+\infty\}=\infty$

$d_{21}=\min\{\infty,\infty+0\}=\infty$

$d_{22}=\min\{0,\infty+4\}=0$

- E assim por diante, para todos os outros d_{ij}, que nesse exemplo, para k=1, permanecem inalterados, com excessão do elemento $d_{32}=\min\{\infty,1+4\}=5$.

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

- Repetindo o mesmo procedimento acima para k=2, teremos a seguinte matriz resultante:

	1	2	3	4	5
1	0	4	9	4	6
2	∞	0	5	∞	2
3	1	5	0	2	7
4	∞	∞	∞	0	4
5	∞	∞	1	∞	0

- Para k=3

	1	2	3	4	5
1	0	4	9	4	6
2	6	0	5	7	2
3	1	5	0	2	7
4	∞	∞	∞	0	4
5	2	6	1	3	0

- Para k=4

	1	2	3	4	5
1	0	4	9	4	6
2	6	0	5	7	2
3	1	5	0	2	6
4	∞	∞	∞	0	4
5	2	6	1	3	0

- Para k=5

	1	2	3	4	5
1	0	4	7	4	6
2	4	0	3	5	2
3	1	5	0	2	6
4	6	10	5	0	4
5	2	6	1	3	0

- Essa matriz resultante contém todos os valores que correspondem aos menores caminhos entre dois vértices quaisquer.

- Obs.: Circuitos Negativos

O algoritmo de Floyd-Warshall não restringe a valoração dos arcos quanto a não negatividade. Por esse motivo podem ocorrer circuitos negativos no decorrer dos cálculos. Se isso acontecer, o fato será registrado na diagonal da matriz correspondente, através do surgimento de valores negativos. Pode-se, nesse ponto, interromper o algoritmo e enviar um aviso de que surgiu um circuito negativo.

4.2.2 - Complexidade

Esse algoritmo é composto de dois ninhos formados de comandos "FOR". Um deles possui cálculos computacionais de complexidade $O(n^2)$ e o outro, formado por três comandos "FOR", possui cálculos de ordem $O(n^3)$. Portanto a ordem de complexidade do algoritmo de Floyd-Warshall é $O(n^3)$.

4.3 - Algoritmo SP1 - Uma Versão do Algoritmo de Dijkstra [7]

Essa nova versão do algoritmo de Dijkstra é aplicada a um caso bem especial do problema de menores caminhos. Sua utilização ocorre, quando o que se procura são os menores caminhos entre todos os vértices de grau ímpar de um grafo não

direcionado. Esse algoritmo é muito importante devido a sua alta eficiência. Essa eficiência se deve ao fato de, sendo $i_1,\ldots,i_n$ todos os vértices ímpares, ele só precisa calcular as menores distâncias entre i_v e i_{v+1}, $i_{v+2},\ldots,i_n$ para $v=1,2,\ldots,v-1$, devido a simetria do grafo. Outra vantagem desse algoritmo, é que ele utiliza um vetor D(i,j) para armazenar as menores distâncias já computadas em passos anteriores, que podem portanto serem utilizadas em passos futuros.

- No algoritmo são utilizados os seguintes conjuntos e variáveis:

  S : conjunto dos vértices numerados de 1 a n;

  L(i) : rótulo do vértice i;

  IZ : contador para os vértices ímpares;

  D(k,i) : vetor que armazena as distâncias entre k e os outros vértices ímpares;

  NVI : variável que indica o número de vértices ímpares do grafo;

  JEND : número de vértices de grau ímpar cujos menores caminhos ainda precisam ser encontrados.

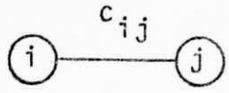
A idéia do algoritmo é, através sòmente dos vértices ímpares, encontrar os menores caminhos entre eles, armazenando-os para evitar repetições de cálculos. Toda vez que um vértice ímpar i_v é encontrado, calcula-se as menores distâncias entre ele e todos os outros ímpares pertencentes ao conjunto i_{v+1}, $i_{v+2},\ldots,i_n$. Para esse algoritmo isso já é suficiente, pois em passos anteriores já foram calculados os menores caminhos entre os vértices ímpares, compreendidos entre $i_0,\ldots,i_{v-1}$ e i_v . A função do contador IZ é indicar quando todos os vértices ímpares já foram considerados.

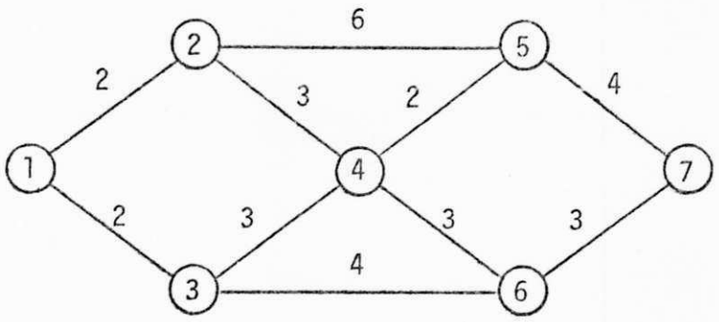
Algoritmo SP1

```
1.   FOR k←1 UNTIL n DO
2.     IF (grau(k) é ímpar) THEN DO
         Begin
3.         S←{1,...,n}; IZ←0;
4.         FOR i←1 UNTIL n DO   L(i)←∞;
5.         FOR i←1 UNTIL k-1 DO
6.           IF (grau(i) é ímpar) THEN DO
               Begin
7.               L(i)←D(k,i);
8.               S←S\{i};  IZ←IZ+1;
               End
9.         L(k)←0;
10.        JEND  NVI-IZ;
11.        WHILE (JEND>0) DO
             Begin
12.            Encontre KMIN com MIN=L(KMIN)=min L(i)
                                                 i∈S
13.            IF MIN=∞ THEN DO  Stop-Grafo não Conectado;
14.            IF (grau(KMIN) é ímpar) THEN DO
                 Begin
15.                D(k,KMIN)←MIN;
16.                JEND←JEND-1
                 End
17.            S←S {KMIN};
18.            FOR (todos os vértices j adjacentes a KMIN) DO
19.              L(j)=min{L(j), L(KMIN)+Custo(KMIN,J)}
           End
       End
```

4.3.1 - Exemplo

Dado o grafo G=(V,E,c) abaixo, deseja-se encontrar os menores caminhos entre todos os vértices ímpares.





- O algoritmo inicializa com k=1. Como o vértice 1 é par, ele é descartado e k←2.

  O conjunto S←{1,2,3,...,7};

  O contador IZ←0;

  Todos os vértices i recebem rótulos L(i)=∞;

  O vértice 2 recebe L(2)=0

  JEND←NVI-IZ = 4

  Nesse ponto é aplicado o algoritmo de Dijkstra normal, para encontrar os menores caminhos entre k e os outros vértices ímpares.

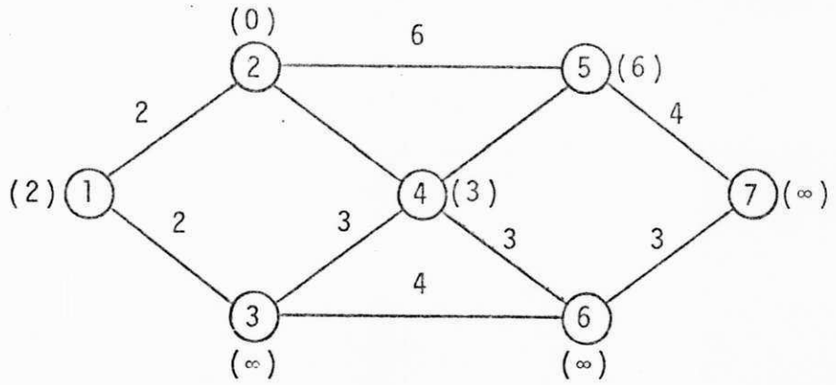
  - KMIN=2 é o vértice pertencente a S, que possui o rótulo de menor valor;

    D(2,2)=0; JEND=3;

    Novo S=S\{2}= {1,3,4,5,6,7}

    Os vértices adjacentes recebem novos rótulos:

    L(1)=2; L(4)=3; L(5)=6



- Enquanto JEND>0 o processo anterior é repetido.

MIN=2 KMIN=1 ; S={3,4,5,6,7} ; L(3)=4

MIN=3 KMIN=4 ; S={3,5,6,7} ; L(5)=5 ; L(6)=6

MIN=4 KMIN=3 ; D(2,3)=4 ; JEND=2 ; S={5,6,7}

MIN=5 KMIN=5 ; D(2,5)=5 ; JEND=1; S={6,7};L(7)=9

MIN=6 KMIN=6 ; D(2,6)=6 ; JEND=0; S={7} ; L(7)=9

Nesse ponto temos : D(2,2)=0

D(2,3)=4

D(2,5)=5

D(2,6)=6

- Voltamos ao passo 2, encontramos um novo vértice ímpar e continuamos os cálculos:

k←3

S={1,2,...,7} ; IZ=0;

L(1)=L(2)=...=L(7)=∞;

L(2)=4 ;

S={1,3,4,5,6,7} ; IZ=1 ;

L(3)=0 ; JEND=3;

MIN=0 KMIN=3 ; D(3,3)=0 ; JEND=2 ; S={1,4,5,6,7}

L(1)=2 ; L(4)=3 ; L(6)=4 ;

MIN=2 KMIN=1 ; S={4,5,6,7}

MIN=3 KMIN=4 ; S={4,6,7} ; L(5)=5

MIN=4 KMIN=6 ; D(3,6)=4 ; JEND=1 ; S={5,7}; L(7)=7

MIN=5 KMIN=5 ; D(3,5)=5 ; JEND=0 ; S={7}

Nesse ponto temos mais as distâncias: D(3,3)=0

D(3,5)=5

D(3,6)=4

- Repetindo o mesmo procedimento para k=5 e k=6, que são os últimos vértices ímpares, tem-se ao final a seguinte matriz distância.

	2	3	5	6
2	0	4	5	6
3	4	0	5	4
4	5	5	0	5
5	6	4	5	0

- Matriz distância para os vértices ímpares.

4.3.2 = Complexidade

Esse método trabalha somente com os m vértices ímpares pertencentes ao grafo de entrada. Para cada vértice ímpar, é encontrada a distância dele para todos os outros vértices ímpares, através do método de Dijkstra, de complexidade $O(n^2)$. Portanto a complexidade final do algoritmo é $O(mn^2)$.

CAPÍTULO V

ALGORITMOS PARA A DETERMINAÇÃO DE FLUXOS EM REDES

Dado um grafo $G=(V,A)$, e as capacidades $q_{ij} \in N$ dos arcos (i,j), serão apresentados dois algoritmos que resolvem os seguintes problemas:

(i) Determinação do fluxo máximo de uma rede;

(ii) Determinação do fluxo de mínimo custo.

Será mostrada também uma modificação no grafo original, utilizada nesse trabalho, que visa adaptá-lo da melhor maneira possível ao algoritmo do caso (ii).

O sistema de apresentação dos algoritmos será o mesmo utilizado no capítulo anterior, ou seja, uma descrição em Pidgin-Algol, a filosofia do algoritmo e uma discussão da complexidade do mesmo.

5.1 - Determinação do Fluxo Máximo em Redes

Uma grande parte dos algoritmos existentes, que tratam desse tipo de problemas, utilizam como técnica de resolução, um método desenvolvido por Ford e Fulkerson [13]. Esse é o principal motivo da escolha de seu algoritmo para apresentar nesse capítulo.

O algoritmo utiliza um sistema de rotulação, que consiste basicamente do seguinte:

. Um vértice i pode ser rotulado, em relação a um outro vértice j, de duas maneiras: (j^+,ε) ou (j^-,ε). No primeiro caso, o rótulo j^+ indica que o fluxo através do arco (j,i) pode ser incrementado. O rótulo j^- indica que o fluxo sobre o arco (i,j) pode ser decrementado. Em ambos os casos o ε representa o valor da quantidade máxima de fluxo que pode ser enviada de um vértice s, escolhido como fonte, para i.

O método tem por objetivo determinar a máxima taxa de fluxo de uma rede, e para isso, após serem definidos um vértice inicial e um final, o algoritmo determina, a cada iteração, um caminho de fluxo aumentado entre eles, com a sua respectiva capacidade. Feito isso, o rótulo do vértice final indicará o incremento máximo de fluxo para esse caminho. Quando não for mais possível encontrar um caminho de fluxo aumentado, é porque o máximo fluxo foi atingido.

No desenrolar dos cálculos, os vértices poderão estar em um dos três estados:

- Rotulado e Examinado. Ele, nesse caso, possui um rótulo e todos os seus vértices adjacentes já foram processados.
- Rotulado e não examinado. Não examinado porque nem todos os seus vértices adjacentes foram processados.
- Não rotulado.

Os vértices inicial s e final t, serão denominados fonte e sumidouro respectivamente.

Algoritmo de Ford-Fulkerson

1. $f \leftarrow 0$;
2. FOR (todos os arcos (i,j)) DO fluxo $x_{ij} \leftarrow 0$;
3. Rotule s como $(0^+, \varepsilon_s)$; $\varepsilon_s \leftarrow \infty$;
4. WHILE (existir um vértice rotulado e não examinado, e t não for rotulado) DO

   Begin
5. Encontre um vértice i rotulado e não examinado;
6. FOR (todos os vértices j não rotulados e adjacentes a i) DO

   Begin
7. IF($(i,j) \in A$ e $x_{ij} < q_{ij}$) THEN DO

   Begin
8. $\varepsilon_j \leftarrow \min\{q_{ij} - x_{ij}, \varepsilon_i\}$;
9. Rotule j com (i^+, ε_j)

   End
10. IF ($(j,i) \in A$ e $x_{ji} > 0$) THEN DO

   Begin
11. $\varepsilon_j \leftarrow \min\{x_{ji}, \varepsilon_i\}$;
12. Rotule j com (i^-, ε_j);

End

End

13. $\underline{i}$ se torna examinado;

End

14. IF ($\underline{t}$ não é possível de ser rotulado) THEN DO Stop- $f_{max}=f$

ELSE

Begin

15. $k \leftarrow t$;

16. WHILE (k=s) DO

Begin

17. IF(k possui rótulo (i^+,ε_k)) THEN DO $x_{ik} \leftarrow x_{ik}+\varepsilon_t$;

18. IF(k possui rótulo (i^-,ε_k)) THEN DO $x_{ki} \leftarrow x_{ki}-\varepsilon_t$;

19. $k \leftarrow i$

End

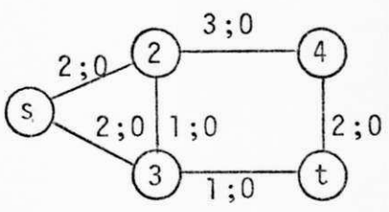
20. $f \leftarrow f+\varepsilon_t$;

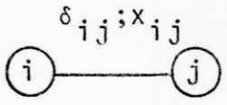
21. Cancele todos os rótulos e GO TO 2

End

5.1.1 - Exemplo

Dado o grafo G=(V,A) abaixo, com as capacidades q_{ij} dos arcos (i,j), deseja-se encontrar o fluxo máximo de $\underline{s}$ para $\underline{t}$.





- $\underline{s}$ recebe o rótulo $(0^+,\infty)$;
- $\underline{s} \leftarrow$ examinando

Os vértices, não rotulados, adjacentes a $\underline{s}$ recebem rótulos:

$\underline{2} \leftarrow (s^+,2)$

$\underline{3} \leftarrow (s^+,2)$

Tem-se portanto : $\underline{s}$ - rotulado e examinado

$\underline{2}$ - rotulado e não examinado

$\underline{3}$ - rotulado e não examinado

Todos os outros vértices - não rotulados

- $\underline{2}$ examinando

Os vértices, não rotulados, adjacentes a $\underline{2}$ recebem rótulos:

$\underline{4} \leftarrow (\underline{2}^+, 2)$

Tem-se agora: $\underline{s}$ e $\underline{2}$ - rotulados e examinados

$\underline{3}$ e $\underline{4}$ - rotulados e não examinados

Os outros são não rotulados

- $\underline{3}$ ← examinando

Os vértices, não rotulados, adjacentes a $\underline{3}$ recebem rótulos:

$\underline{t} \leftarrow (3^+, 1)$; $\varepsilon_t = 1$

- Nesse ponto $\underline{t}$ é rotulado e passa-se para o passo 15:

$k \leftarrow t$

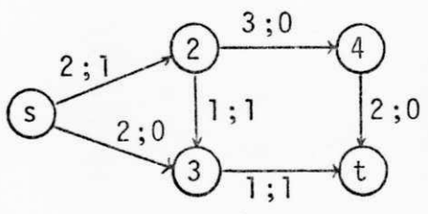
$x_{3t} \leftarrow x_{3t} + \varepsilon_t = 0+1=1$

$x_{23} \leftarrow x_{23} + \varepsilon_t = 0+1=1$

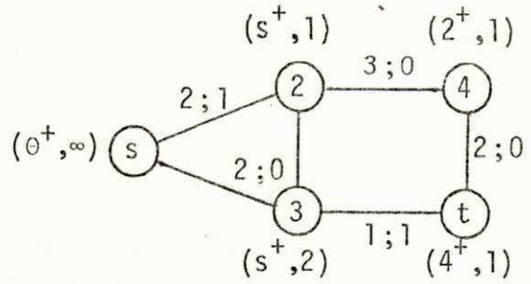
$x_{s2} \leftarrow x_{s2} + \varepsilon_t = 0+1=1$

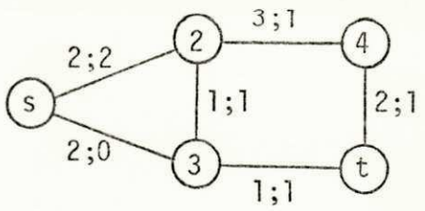
Tem-se como resultado da $1^{\underline{a}}$ iteração o seguinte grafo:

$f \leftarrow f + \varepsilon_t = 0+1=1$



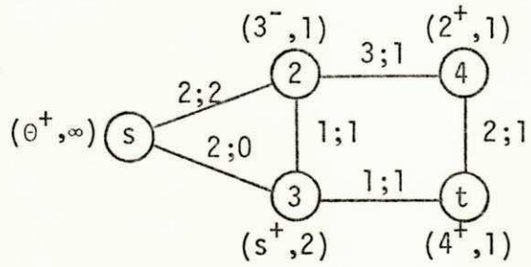
Para esse novo grafo, repetindo-se o procedimento anterior, tem-se novos rótulos e um novo fluxo:

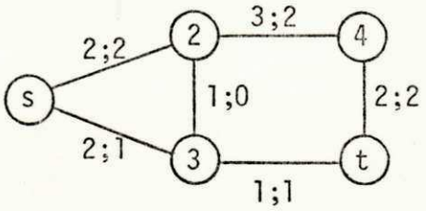




f←f+1 = 1+1 = 2

Para uma terceira iteração, obtém-se os seguintes rótulos e fluxos:





f←f+1 = 3

- Não é mais possível rotular t, portanto: f = fluxo máximo = 3

5.1.2 - Complexidade

Considerando um grafo com m arcos, e f como sendo o valor do fluxo máximo de s para t. Considerando também que as capacidades são inteiras. A complexidade computacional pode ser estimada da seguinte forma:

- Cada vez que um caminho é construído, no máximo 2m inspeções nos arcos são necessárias;
- E como as capacidades são inteiras, então no máximo f caminhos serão necessários.

Portanto a complexidade desse algoritmo é de ordem O(mf).

5.2 - Determinação do Fluxo de Mínimo Custo

Considere-se agora o problema de se conduzir um fluxo de valor f, de um vértice fonte s para um sumidouro t, de tal forma que o custo relacionado a esse fluxo seja mínimo. Para esse problema será utilizado o grafo G=(V,A,q,c), onde q é associado às capacidades dos arcos, e c aos custos dos mesmos.

O algoritmo mais conhecido, que trata desse problema, é o denominado "out-of-kilter", desenvolvido por Ford e Fulkerson [13], mas o método que será descrito nessa seção será o de Busacker-Gowen [2], que conceitualmente é mais simples, e em termos computacionais é comparável ao "out-of-kilter".

A idéia do algoritmo baseia-se na construção de uma sequência de grafos correspondentes aos fluxos atuais, com um conjunto de arcos A* formado pela união de outros dois conjuntos A' e A'', que são: $A' = \{a_{ij} | x_{ij} < q_{ij}\}$

$$A'' = \{a_{ji} | x_{ij} > 0\}$$

e pelos custo y_{ij} :

$$y_{ij} = \begin{cases} c_{ij} & , \text{ se } a_{ij} \in A' \\ -c_{ij} & , \text{ se } a_{ij} \in A'' \end{cases}$$

Nesses novos grafos são determinados os menores caminhos, e superpondo o fluxo desejado sobre o caminho correspondente, obtém-se o fluxo possível. Os cálculos terminarão quando o valor do fluxo desejado for atingido.

Algoritmo Busacker-Gowen

1. $\bar{f} \leftarrow$ fluxo pré-determinado;
2. $f \leftarrow 0$;
3. WHILE ($f \neq \bar{f}$) DO

   Begin
4. FOR (todos x_{ij}) DO

   Begin
5. IF ($x_{ij} < q_{ij}$) THEN DO $A' \leftarrow a_{ij}$;
6. IF ($x_{ij} > 0$) THEN DO $A'' \leftarrow a_{ji}$;

   End
7. FOR (todos a_{ij}) DO

   Begin
8. IF ($a_{ij} \varepsilon A'$) THEN DO $y_{ij} \leftarrow c_{ij}$;
9. IF ($a_{ij} \varepsilon A''$) THEN DO $y_{ij} \leftarrow -c_{ij}$;

   End

c Nesse ponto é construído o grafo $I(G)=(V,A^*,Y)$, onde

c $A^*=A' \cup A''$ e $Y=(y_{ij})$

10. Determine um caminho mais curto $\underline{p}$, de $\underline{s}$ para $\underline{t}$, em $I(G)$ com respeito aos custos y_{ij} ;
11. IF (não existir caminho) THEN DO Stop;
12. $\delta' \leftarrow \infty$;
13. $\delta'' \leftarrow \infty$;
14. FOR (todos $a_{ij} \varepsilon p$) DO

   Begin
15. IF($a_{ij} \varepsilon A'$) THEN DO $\delta' \leftarrow \min\{q_{ij}-x_{ij}, \delta'\}$;
16. IF($a_{ij} \varepsilon A''$) THEN DO $\delta'' \leftarrow \min\{x_{ji}, \delta''\}$

   End
17. $\varepsilon \leftarrow \min\{\delta', \delta'', \bar{f}-f\}$;

18. FOR (todos $a_{ij} \varepsilon p$) DO

Begin

19. IF($a_{ij} \varepsilon A'$) THEN DO $x_{ij} \leftarrow x_{ij} + \varepsilon$;

20. IFIa$_{ij} \varepsilon A''$) THEN DO $x_{ji} \leftarrow x_{ji} - \varepsilon$;

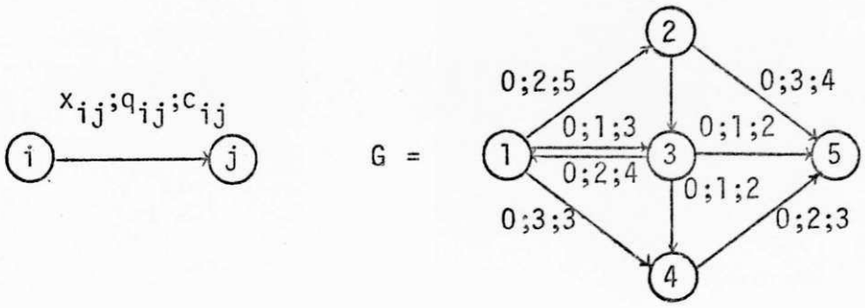
End

21. $f \leftarrow f + \varepsilon$

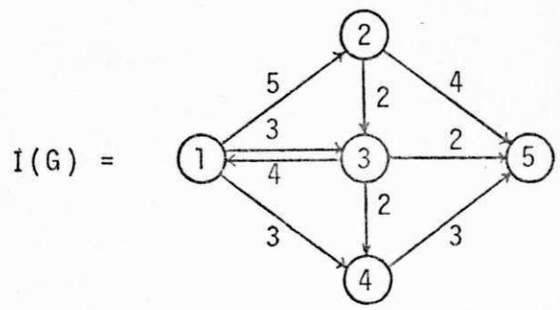
End

5.2.1 - Exemplo

Dado o grafo G=(V,A), com os respectivos fluxos, capacidades e custos relativos aos arcos, deseja-se encontrar o fluxo de mínimo custo para passar 5 unidades do vértice 1 para o 5. Portanto $\bar{f}=5$.



- Através dos passos de 1. a 9. monta-se o seguinte grafo I(G):



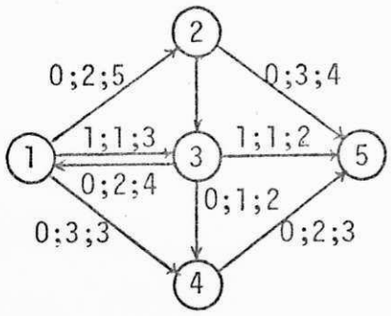
- É encontrado o menor caminho entre 1 e 5

p={1→3→5}

$\delta'=1$; $\delta''=\infty$; $\bar{f}-f=5$; $\varepsilon=1$;

$f=1$

- Devido ao fluxo que é atravessado, o grafo original é modificado e como $f \neq \bar{f}$, repete-se o procedimento anterior.

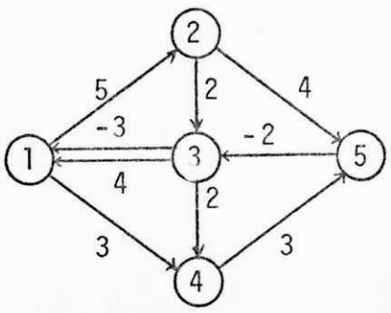


G'

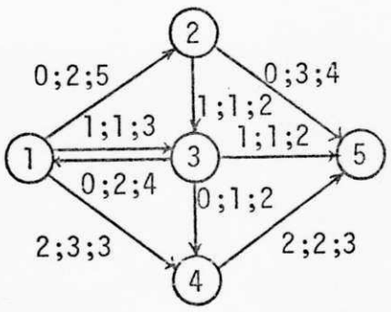
- $p=\{\underline{1}\to\underline{4}\to\underline{5}\}$

$\delta' = 2$; $\delta'' = \infty$; $\bar{f}-f=4$; $\varepsilon = 2$;

$f=3$



I(G')

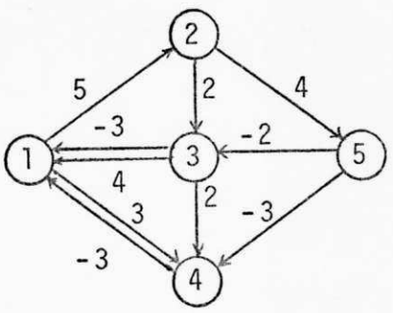


G"

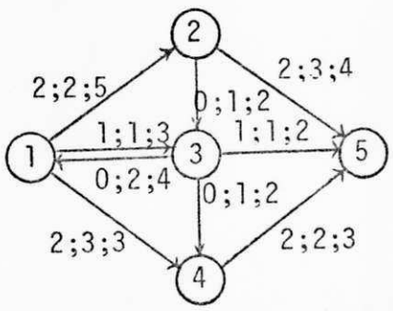
- $p=\{\underline{1}\to\underline{2}\to\underline{5}\}$

$\delta' = 2$; $\delta'' = \infty$; $\bar{f}-f=2$; $\varepsilon = 2$

$\bar{f}=f=5$



I(G")



- Stop. $\bar{f}=f=5$

$$\sum_{a_{ij}\in A} c_{ij}x_{ij} = 35 \text{ u.m.}$$

5.2.2 - Complexidade

Considerando as capacidades inteiras e a não presença de circuitos negativos, a complexidade desse algoritmo depende do método que será utilizado para se encontrar os menores caminhos. Utilizando-se um algoritmo de complexidade $O(n^3)$, como o de Floyd-Warshall por exemplo, e sendo f o fluxo desejado, o número máximo de caminhos necessários serão f, e portanto a ordem de complexidade do algoritmo de BUSACKER-GOWEN será $O(fn^3)$. Essa ordem de complexidade é a mesma do método "Out-of-kilter".

5.3 - Modificação do Grafo Original Visando Aplicação do Algoritmo de Busacker-Gowen

Para se realizar a aumentação Entrada-Igual-Saída, foi utilizado o algoritmo de Busacker-Gowen. Para isso foi necessária uma alteração no grafo de entrada, pois nesse caso particular tem-se, na maioria das vezes, vários vértices fontes e vários sumidouros. Isso ocorre porque, para se fazer entr(i)=saída(i) para todo $i \in V$, deve-se passar um fluxo $\bar{f} = \sum_{i \in V} \max\{def(i),0\}$ dos vértices i com $def(i)>0$, para os vértices j com $def(j)<0$.

A modificação no grafo de entrada é feita para evitar que se trabalhe com várias fontes e sumidouros. Isso é feito adicionando-se ao grafo original dois vértices artificiais s e t, ligados ao grafo da seguinte forma:

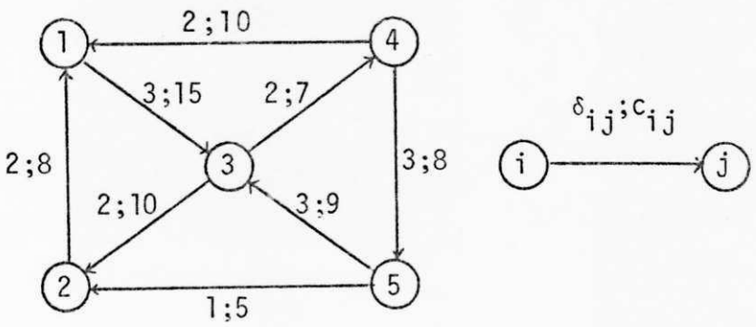
- O vértice s, que será a nova e única fonte, será ligado aos vértices i com $def(i)>0$, através de arcos (s,i), com capacidades $q_{si}=def(i)$ e custos $c_{si}=0$.

- O vértice sumidouro t estará ligado aos vértices j, que possuem def(j)<0, através de arcos (j,t) com capacidades q_{jt}=-def(j) e custos c_{jt}=0.

Dessa forma pode-se utilizar o algoritmo de Busacker-Gowen, sem modificação, para determinar o fluxo f de custo mínimo da origem s para o sumidouro t. Ao final, depois de cancelados os vértices artificiais, e providenciadas todas as devidas mudanças nas orientações e duplicações dos arcos e ramos, a entr(i)=saída(i) para todo iεV.

5.3.1 - Exemplo

- Modificação de um grafo de entrada, visando a aplicação do algoritmo de Busacker-Gowen.



- Nesse grafo tem-se dois vértices com deficit positivo:

1 - def(1)=1

2 - def(2)=1

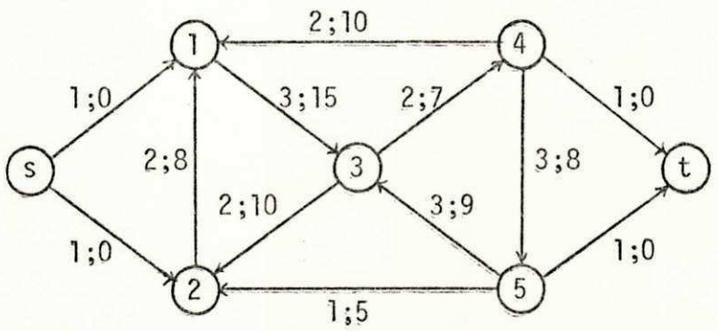
e dois com deficit negativo:

4 - def(4)=-1

5 - def(5)=-1

- Portanto é preciso passar através desse grafo um fluxo f= $\sum_{i\varepsilon V}$ max{def(i),0}=2, dos vértices 1 e 2 para os 4 e 5.

- Introduzindo os vértices artificiais s e t, tem-se o seguinte grafo modificado:



- Nesse grafo, com uma única fonte e um sumidouro, basta aplicar, sem nenhuma alteração, o algoritmo de Busacker-Gowen, para obter ao final, depois das modificações efetuadas, entr(i)=saída(i) para todo i∈V.

Obs.: Essa alteração não interfere na complexidade do algoritmo.

CAPÍTULO VI

CASAMENTO PERFEITO DE MÍNIMO CUSTO (CPMC)

O casamento trabalha com um grafo não direcionado G=(V,E), cujos ramos eεE são valorados não negativamente.

Os primeiros trabalhos sobre casamentos perfeitos de mínimo custo foram realizados por Edmonds [10], sendo a sua idéia central baseada no conceito de flores. Já Derigs [8] fez uma abordagem um pouco diferente de Edmonds, baseada no que ele chamou de Transformações Admissíveis.

6.1 - Conceitos sobre Casamentos

Definição 6.1.1

Um casamento é perfeito quando todos os vértices iεV são casados.

Definição 6.1.2

Seja $\mathcal{M}_0$ o conjunto formado por todos os casamentos do grafo G.

M é um Casamento de Máxima Cardinalidade sse $|M| \geq |M'|$ para todo $M' \in \mathcal{M}_0$.

Sendo $c(e) \geq 0$ os custos associados a cada ramo $e \in E$, define-se custo de um casamento M como sendo:

$$c(M) = \sum_{e \in M} c(e)$$

Definição 6.1.3

Seja $\mathcal{M}$ o conjunto de todos os casamentos perfeitos de G.

É chamado casamento perfeito de mínimo custo o casamento M cujo custo

$$c(M) = \min_{M' \in \mathcal{M}} c(M')$$

Uma condição necessária para a existência desse casamento é que o grafo possua um número par de vértices.

Definição 6.1.4

Seja M um casamento em G, e P um caminho alternado.

Define-se custo do caminho P como sendo

$$c(P) = \sum_{(i,j) \in P \setminus M} c_{ij} - \sum_{(i,j) \in P \cap M}$$

Sendo P(M) o conjunto de todos os caminhos aumentados de G, relativo a M, P_0 é um caminho aumentado de mínimo custo se

$$c(P_0) \leq C(P) \quad \text{para todo } P \in P(M)$$

Seja P' um caminho aumentado. O casamento melhorado $M \oplus P'$ é dado por

$$M \oplus P' = (M \setminus P') \cup (P' \setminus M)$$

Esboço do Algoritmo que Resolve o Problema do CPMC

Seja G=(V,E) um grafo completo, com |V| par e $c_{ij} \geq 0$ para todo $(i,j) \in E$.

A idéia central do método que encontra o CPMC é procurar os menores caminhos aumentados, e através deles melhor o casamento.

```
1.  M←Ø;
2.  WHILE (M não for completo) DO
      Begin
3.       Encontre o menor caminho aumentado P0 ε P(M);
4.       M ← M ⊕ P0
      End
```

6.2 - Fundamentos para o Algoritmo que encontra o Menor Caminho Aumentado

Se $s \in V$ é um vértice exposto, com respeito ao casamento M, e $P_{s(M)}$ o conjunto de todos os caminhos aumentados

com vértice inicial s, tem-se para $P \in P_{s(M)}$

$$c(M \oplus P) = c(M) + c(P)$$

Definição 6.2.1

Um circuito k é denominado <u>circuito alternado negativo</u> se $c(k)<0$.

Os teoremas citados a seguir constituem os primeiros critérios de otimalidade para o CPMC.

Teorema 6.I

$M \in \mathcal{M}$ é ótimo sse M não admitir circuitos alternados negativos.

Teorema 6.II

Seja M um casamento que não admite circuitos alternados negativos, e P o menor caminho aumentado com respeito a $P_{s(M)}$. Então $M \oplus P$ não permite circuitos alternados negativos.

6.3 - Transformações Admissíveis

Seja M um casamento e $P_{s(M)}$ o conjunto de todos os caminhos aumentados com vértice inicial s.

Definição 6.3.1

Uma <u>transformação</u> $T : c_{ij} \rightarrow c'_{ij}$ é <u>admissível</u> sse

$$c'(P) \geq 0 \qquad \forall\ P \in P_{s(M)}$$

Define-se $d(P) : c(P) - c'(P)$

Com respeito às Transformações Admissíveis, Derigs [7] formulou o seguinte critério de otimalidade.

Teorema 6.III

Seja T uma transformação admissível e $P_0 \in P_{s(M)}$, com

(i) $c'(P_0) = 0$

(ii) $d(P_0) \leq d(P) \qquad \forall\ P \in P_{s(M)}$

Então P_0 é um menor caminho aumentado.

A prova desse teorema é muito simples:

$$c(P_0) = c'(P_0) + d(P_0)$$

$$= d(P_0) \leq d(P) \leq c'(P) + d(P) = c(P) \quad \forall\ P \in P_{s(M)}$$

As transformações utilizadas nos algoritmos do CPMC são construídas da seguinte forma:

(i) Seja $\mathcal{R}$ um conjunto formado de subconjuntos $R_k \subset V$, cuja cardinalidade $|R_k| \geq 3$ é ímpar

(ii) Atribui-se pesos aos vértices i e conjuntos R_k

Y_i para todo $i \in V$

$Y_k \geq 0$ para todo $R_k \in$

(iii) Tomando $V_1 \subseteq V$, e definindo $\Psi(e)$ como sendo os vértices finais do arco e, define-se co-limite de V_1 por

$$\delta(V_1) = \{e \in E \mid |\Psi(e) \cap V_1| = 1\}$$

A transformação T : $c_{ij} \rightarrow c'_{ij}$ é dada por

$$c'_{ij} = c_{ij} - Y_i - Y_j - \sum_{\{i,j\} \in \delta(R_k)} Y_k$$

O grafo $G=(V,E')$, onde $E'=\{\{i,j\} \mid c'_{ij} = 0\}$, é chamado Grafo Admissível.

Através dessas Transformações Derigs [7] pretende assegurar sua hipótese de que, durante o desenrolar do algoritmo, os custos dos ramos $e \in M$ e dos ramos pertencentes às flores encolhidas devam ter custos zero. Para isso, toda vez que um menor caminho aumentado é encontrado, ou uma flor é encolhida, seus ramos são incorporados a um grafo admissível.

6.4 - Algoritmo para o Casamento Perfeito de Mínimo Custo

O método utiliza as seguintes subrotinas:

- aumentação;
- rotulação;
- FLOR;
- expansão dos vértices

A entrada para o algoritmo é um grafo completo $G=(V,E)$, com $|V|$ par e custos $c_{ij} \geq 0$ para todo $\{i,j\} \in E$.

- Os valores Y(i) associados aos vértices i são inicializados com 0;

- MATCH(i)=j indicará para cada vértice i casado o respectivo vértice adjacente j, associado a $\{i,j\}\in M$. Para os vértices expostos MATCH(i)=0;
- d_i^+ (d_i^-) indica o custo do menor caminho alternado de cardinalidade par (ímpar), do vértice i para um vértice exposto;
- KA(i) indica o vértice adjacente a i, no caminho associado a d_i^- .

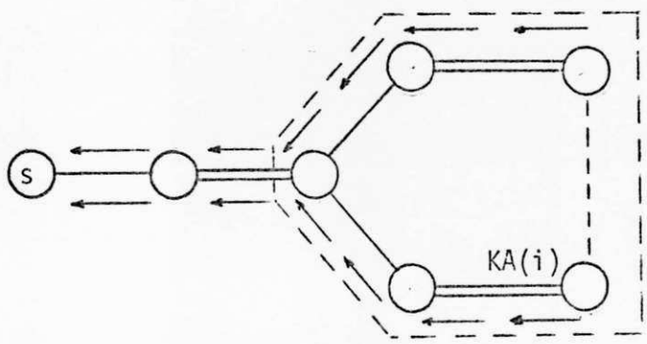
Algoritmo CPMC

1. FOR (todos os vértices i) DO
   Begin
2. MATCH(i)←0; Y(i)←MATCH(i);
3. Faça i ser não rotulado;
   End
4. FOR (todos os vértices i) DO
5. IF (i é exposto) THEN DO
   Begin
6. i recebe o rótulo saída "0";
7. $d_i^+ \leftarrow 0$;
   End
   ELSE
8. $d_i^+ \leftarrow \infty$;
9. FOR (todos os vértices i) DO
   Begin
10. $d_i^- \leftarrow c_{i\ell} = \min\{c_{ij} | j \text{ é rotulado "0"}\}$;
11. KA(i) ← ℓ
   End
12. WHILE (existirem vértices expostos) DO
   Begin
13. $\delta_1 \leftarrow d_i^- = \min\{d_j^- | j \text{ não é rotulado}\}$;
14. $\delta_2 \leftarrow (d_i^- + d_i^+)/2 = \min\{(d_j^- + d_j^+)/2 | j \text{ é rotulado "0"}\}$;
15. $\delta_3 \leftarrow Y(i) + d_i^- = \min\{Y(j) + d_j^- | j \text{ é rotulado entrada "I"}\}$;
16. $\delta \leftarrow \min\{\delta_1, \delta_2, \delta_3\}$;
17. IF $\delta = \delta_1$ THEN CALL ROTULAÇÃO(i);
18. IF $\delta = \delta_3$ THEN CALL EXPANSÃODOSVÉRTICES (i);
19. IF $\delta = \delta_2$ THEN DO

```
            Begin
20.             Volte do vértice i, marcado com Θ , e do KA(i)
                para os vértices expostos;
21.             IF (os vértices expostos forem idênticos) THEN
                CALL FLOR(i);
                ELSE CALL AUMENTAÇÃO(i);
            End
        End
23.  Expanda todas as flores, corrigindo o casamento.
```

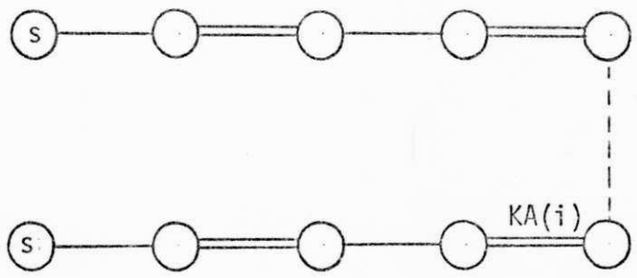
Obs.: Uma explanação sobre os passos 20., 21. e 22. :

- Quando é feita a volta de i e KA(i) para os vértices expostos, uma possibilidade é a coincidência desses vértices, como por exemplo:



Nesse caso há o aparecimento de uma flor e é acionada a subrotina FLOR.

- A outra possibilidade é a ocorrência de um caminho aumentado, quando os vértices expostos são distintos:



Quando essa possibilidade ocorre, a subrotina AUMENTAÇÃO é acionada e o casamento melhorado.

A subrotina ROTULAÇÃO é chamada quando $\delta=\delta_1$. Ela cuida do desenvolvimento de um caminho alternado, rotulando-o.

```
Subrotina ROTULAÇÃO(i)
1.  Rotule i com "I:KA(i)";
2.  j ← MATCH(i);
3.  Rotule j com "Θ:i";
4.  d_j^+ ← δ;
5.  FOR (todos os vizinhos ℓ de j) DO
6.      IF (ℓ≠i) THEN
7.          IF (d_j^+ + c(j,ℓ) < d_ℓ^-) THEN
                Begin
8.                  d_ℓ^- ← d_j^+ + c(j,ℓ);
9.                  KA(ℓ) ← j
                End
10. RETURN
```

A subrotina AUMENTAÇÃO é acionada quando $\delta=\delta_2$ e os caminhos alternados que atingem i e KA(i) são originados de vértices expostos distintos. Os dois caminhos alternados formam um caminho aumentado, e o casamento é melhorado (passos 1 e 2).

Todos os vértices rotulados com "Θ" recebem os seguintes novos pesos (passo 5).

(1) $\quad Y'_k \leftarrow Y_k + (\delta - d_k^+)$

São trocados os pesos dos vértices rotulados com "I" (passo 9).

(2) $\quad Y'_k \leftarrow Y_k - (\delta - d_k^-)$

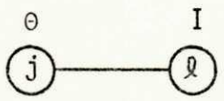
Os custos dos ramos dos caminhos alternados são reduzidos a zero. Isso é feito através dos passos 6.-7. e 10.-11..

Exemplo:

(a) $\quad (j,\ell) \notin M$

j é rotulado "Θ"

ℓ é rotulado "I"



(3) $\quad c''_{j\ell} \leftarrow c'_{j\ell} - \sum_{\substack{V_k \in \mathcal{R} \\ (j,\ell)\in\delta(V_k)}} Y'_k$

$$= c'_{j\ell} - (\delta - d_j^+) + (\delta - d_\ell^-)$$

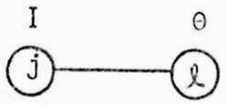
$$= c'_{j\ell} + d_j^+ - d_\ell^-$$

$$= 0 \qquad\qquad d_\ell^- = d_j^+ + c'_{j\ell}$$

(b) Se (j,ℓ) M

j é rotulado "I"

ℓ é rotulado "Θ"



(4) $c''_{j\ell} \leftarrow c'_{j\ell} + (\delta - d_j^-) - (\delta - d_\ell^+)$

$= c'_{j\ell} + d_\ell^+ - d_j^-$

$= 0$

$d_\ell^+ = d_j^-$

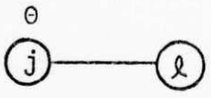
e

$c'_{j\ell} = 0$

(c) $(j,\ell) \notin M$

j é rotulado "Θ"

ℓ não é rotulado



$c''_{j\ell} \leftarrow c'_{j\ell} - (\delta - d_j^+)$

$= c'_{j\ell} + d_j^+ - \delta$

$\geq d_\ell^- - \delta$ $\qquad d_\ell^- \leq d_j^+ + c'_{j\ell}$

≥ 0 $\qquad \delta \leq \delta_1 \leq d_\ell^-$

Nos passos 12 → 15 é feita uma nova rotulação para os vértices j .

Nos passos 16 → 18 é feita uma nova avaliação dos d^- .

Subrotina Aumentação (i)

```
1.  Encontre o caminho aumentado P;
2.  M ← M + P;
3.  FOR (todos os vértices j) DO
        Begin
4.          IF (j é rotulado "Θ") THEN
              Begin
5.                Y(j) ← Y(j) + (δ-d_j^+);
6.                FOR (todos os vizinhos ℓ de j) DO
7.                   c_jℓ ← c_jℓ - (δ-d_j^+)
              End
8.          IF (j é rotulado "I") THEN
              Begin
9.                Y(j) ← Y(j) - (δ-d_j^-);
10.               FOR (todos os vizinhos ℓ de j) DO
11.                  c_jℓ ← c_jℓ + (δ-d_j^-);
              End
12.         Descarte os rótulos de j;
13.         d_j^+ ← d_j^- ← ∞;
14.         IF (j é exposto) THEN
              Begin
15.               Rotule j com "Θ:Ø";
16.               d_j^+ ← 0;
              End
        End
17. FOR (todos os vértices j) DO
        Begin
18.         d_j^- ← c_jℓ ← min{c_jk | K é "Θ"};
19.         KA(j) ← ℓ
        End
```

A subrotina FLOR é chamada quando $\delta=\delta_2$ e os caminhos alternados que atingem i e KA(i) tem origem no mesmo vértice exposto.

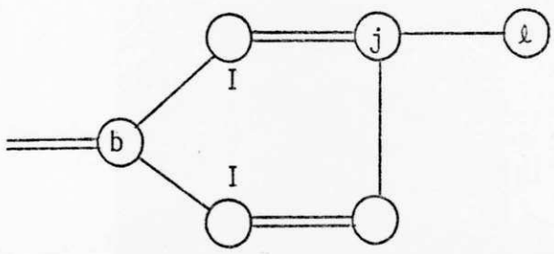
Os passos de 3 a 10, fazem transformações nos vértices pertencentes à flor, semelhantes às (1) e (2) feitas na subrotina AUMENTAÇÃO.

Os custos dos ramos pertencentes à flor são reduzidos a zero, da mesma forma que nos procedimentos (3) e (4).

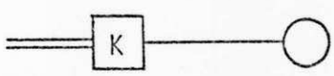
Exemplo:

(a) $j \in V_k$
$\ell \notin V_k$

j e rotulado "Θ"



O comprimento dos caminhos alternados que atingem ℓ é $d_j^+ + c_{j\ell}$



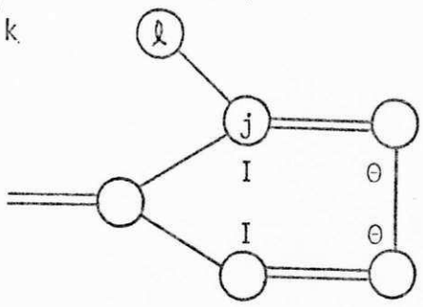
O comprimento do caminho aumentado é:

$$\begin{aligned} d_k^+ + c''_{k\ell} &= \delta + c''_{k\ell} \\ &= \delta + c'_{j\ell} - (d - d_j^+) \\ &= c'_{j\ell} + d_j^+ \end{aligned}$$

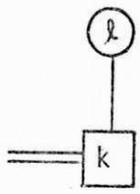
(b) $j \in V_k$

$\ell \notin V_k$

j possui rótulo "I"



Antes do Encolhimento.



Depois do Encolhimento

O comprimento dos caminhos alternados que atingem ℓ é dado por:

$$\begin{aligned} d_k^+ + c''_{k\ell} &= \delta + c'_{k\ell} \\ &= \delta + c'_{j\ell} + (\delta - d_j^-) \\ &= c'_{j\ell} + 2\delta - d_j^- \\ &= c'_{j\ell} - d_j^- + d_i^- + d_i^+ \quad \delta = (d_i^- + d_i^+)\ 2 \end{aligned}$$

Os passos de 11 a 15, tratam do encolhimento da flor, modificando o grafo, corrigindo o casamento, rotulando e atribuindo peso ao pseudo-vértice K, além de especificar o comprimento do caminho alternado de ℓ para o vértice exposto.

Os passos de 16 a 19 corrigem, se necessário, os valores d_ℓ^- para todos os vértices ℓ vizinhos ao pseudo-vértice K .

Quando $\delta=\delta_3=Y_i+\bar{d}_i$ e o vértice i é rotulado "I", o casamento aciona a subrotina EXPANSÃO DOS VÉRTICES.

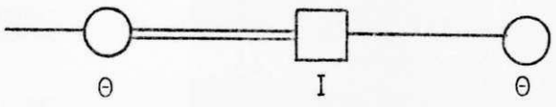
Nos passos de 1 a 3, da mesma forma que no procedimento (2), são feitas as transformações

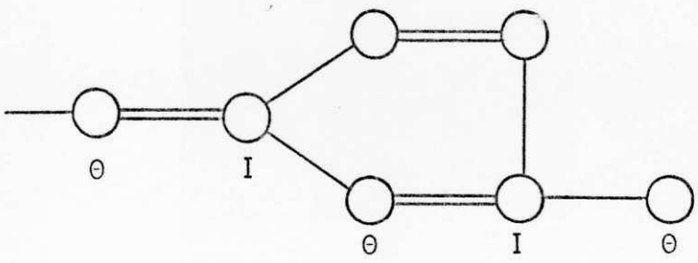
$$Y''_i \leftarrow Y'_i - (\delta - \bar{d}_i) = 0$$

$$\text{onde } \delta = Y'_i + \bar{d}_i$$

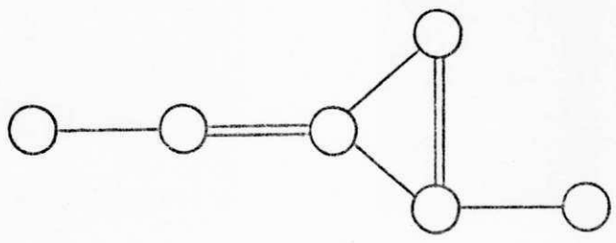
A expansão da flor é feita de uma forma contrária ao encolhimento da mesma.

Exemplo:

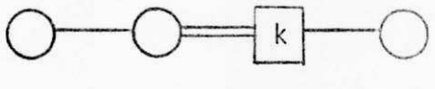




Quando do encolhimento da flor, pode ocorrer o aparecimento de um caminho aumentado, e por esse motivo o casamento interno à flor deve ser corrigido



Flor



Encolhimento da Flor

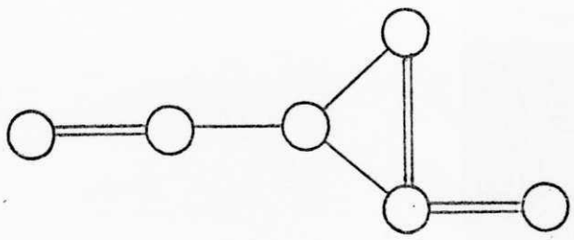
Subrotina FLOR(i)

1. Encontre a flor $B_k = G[V_k]$;
2. FOR (todos os vértices $j \in V_k$) DO
   Begin
3. IF (j é "Θ") THEN
   Begin
4. $Y(j) \leftarrow Y(j) + (\delta - d_j^-)$;
5. FOR (todos os vizinhos $\underline{\ell}$ de j) DO
6. $c_{j\ell} \leftarrow c_{j\ell} - (\delta - d_j^+)$
   End
7. IF (j é rotulado "I") THEN
   Begin
8. $Y(j) \leftarrow Y(j) - (\delta - d_j^-)$;
9. FOR (todos os vizinhos $\underline{\ell}$ de j) DO
10. $c_{j\ell} \leftarrow c_{j\ell} + (\delta - d_j^-)$
   End
   End
11. $G \leftarrow G \times B_k$; "encolhimento da flor"
12. corrigir o casamento;
13. rotule o pseudo-vértice K com "Θ";
14. $d_k^+ \leftarrow \delta$;
15. $Y(k) \leftarrow 0$;
16. FOR (todos os vizinhos $\underline{\ell}$ de K) DO
17. IF $((d_k^+ + c_{k\ell})\ d_\ell^-)$ THEN
   Begin
18. $d_\ell^- \leftarrow d_k^+ + c_{k\ell}$;
19. $KA(\ell) \leftarrow k$
   End

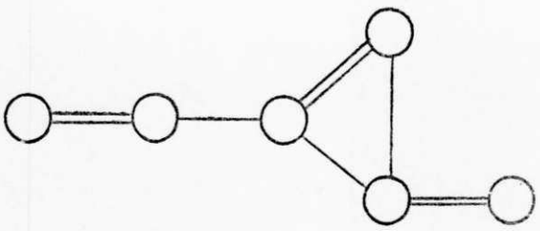
Surge um caminho aumentado, e o casamento é melhorado

Aumentação

Na expansão da FLOR (passo 4) existirá um vértice casado comum a dois ramos pertencentes ao casamento. Esse casamento deve ser corrigido (passo 5).



Expansão



Casamento Corrigido

Os passos de 6 a 17, modificam, quando necessário, os valores dos d_j^+ e d_j^-, para todos os $j \in V_i$. Modificam também os valores dos d_ℓ^-, para todos os vértices ℓ vizinhos a um j rotulado como "Θ".

Subrotina Expansão dos Vértices (i)

1. FOR (todos os vizinhos j de i) DO
2. $c_{ij} \leftarrow c_{ij} + (\delta - d_i^-)$;
3. $Y_i \leftarrow 0$;
4. Expanda a flor B_i em G;
5. Corrigir os rótulos e o casamento interno da flor;
6. FOR (todos os vértices $j \in V_i$) DO

   Begin

7. IF (j é rotulado "⊖") THEN

   Begin

8. $d_j^+ \leftarrow \delta$;
9. FOR (todos os vizinhos ℓ de j) DO
10. IF $((d_j^+ + c_{j\ell}) < d_\ell^-)$ THEN

   Begin

11. $d_\ell^- \leftarrow d_j^+ + c_{j\ell}$;
12. $KA(\ell) \leftarrow j$

   End

   End

13. IF (j é rotulado "I") THEN
14. $d_j^- \leftarrow \delta$;
15. IF (j não e rotulado) THEN

   Begin

16. $d_j^- \leftarrow d_\ell^+ + c_{\ell j} \leftarrow \min\{d_k^+ + c_{k\ell} \mid K$ é rotulado "⊖"$\}$
17. $KA(j) \leftarrow \ell$

   End

   End

6.5 - Implementação e Complexidade

Os algoritmos de Derigs [7] foram implementados por Kazakidis, em linguagem FORTRAN IV em sua tese de mestrado. Maiores detalhes sobre a implementação desses algoritmos são encontrados no trabalho feito por Costa [4].

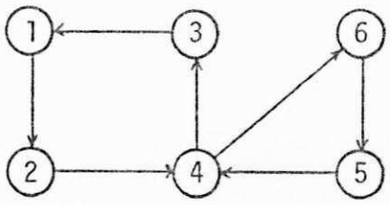
A complexidade desses algoritmos é a mesma do algoritmo de Edmonds (versão Lawler [16], ou seja $O(n^3)$, embora a implementação de Kazakidis na prática tenha se mostrado mais eficiente.

CAPÍTULO VII

ALGORITMOS PARA A CONSTRUÇÃO DA ROTA FINAL

Como já foi citado no capítulo 3, pode-se traçar uma rota de Euler em um grafo totalmente direcionado e conectado, bastando que entr(i)=saída(i) para todo $i \in V$. Com relação a grafos conectados não direcionados, a única restrição, que garante a rota de Euler, é a de que todo vértice $i \in V$ possua grau par. Em se tratando de grafos conectados mistos, ambas as condições são necessárias.

Embora essas condições sejam necessárias e suficientes para garantir a existência da rota de Euler, elas não são suficientes para permitir que tracemos a rota livremente, sem correr o risco de desconectar o grafo. Esse fato pode ser constatado no exemplo a seguir:

G = 

- Esse grafo é totalmente direcionado, conectado, entr(i) é igual a saída(i) para todo vértice i, e portanto está garantida a existência de uma rota de Euler. Mas dependendo do vértice inicial e do traçado da rota, pode ocorrer a desconexão do grafo. Iniciando por 1, pode-se traçar a seguinte rota:

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 1$$

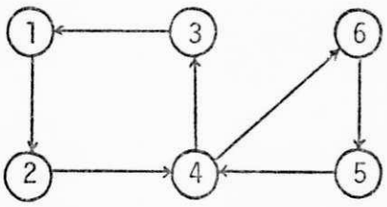
que resolve o problema. Mas essa rota foi feita de uma maneira conveniente, pois caso contrário poderia ter havido problemas, como o seguinte:

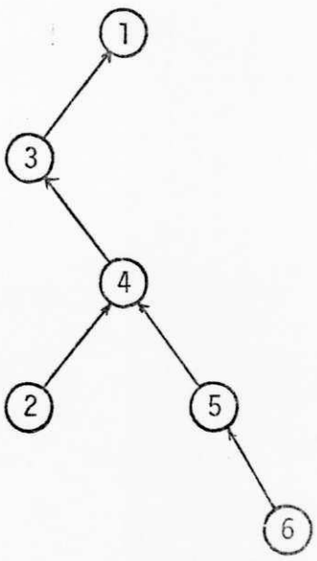
$$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$$

que é um circuito sem continuidade, e como todos os arcos devem ser atravessados uma única vez, não se trata portanto da rota desejada. Se for considerado o grafo não direcionado associado ao grafo acima, constata-se que o problema surgido anteriormente pode ocorrer também nesse tipo de grafo. E da mesma forma pode-se estender para grafos mistos.

Visando solucionar esse problema, Edmonds e Johnson apresentaram uma idéia baseada na construção de uma árvore "SPANNING".

Como foi visto no capítulo das definições, uma árvore é um grafo conectado, onde saída(raiz)=0 e saída(i)=1 para todo i≠raiz. Uma árvore é chamada "SPANNING" quando todos os vértices de G aparecem em T exatamente uma vez. O exemplo abaixo mostra, em relação ao grafo G, uma árvore "SPANNING".

G = 

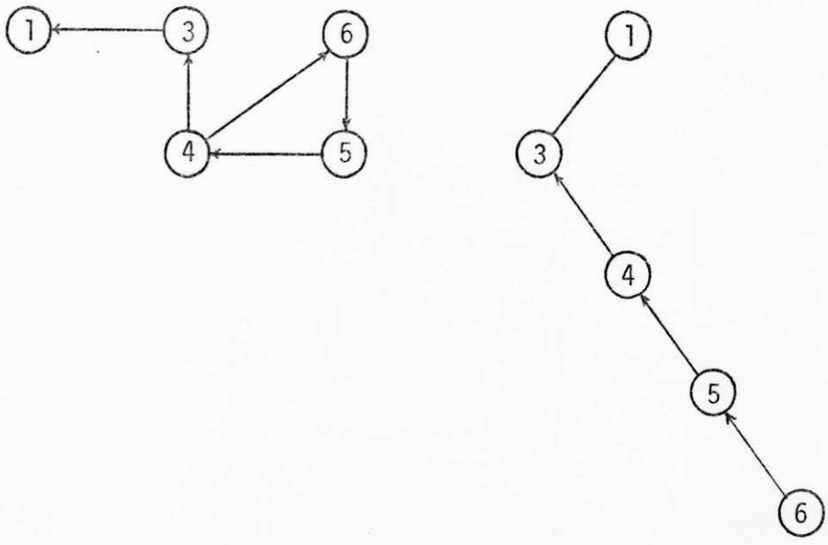
T = 

A idéia de Edmonds e Johnson [11] para a construção da rota, generalizada para grafos mistos, consiste em, partindo do vértice r escolhido como raiz, toda vez que um vértice n é atingido, deve-se sair desse vértice através de qualquer ramo ainda não utilizado; caso não exista mais nenhum, escolhe-se qualquer arco que não pertença à árvore, e por fim se só existir o arco da árvore, este deve ser utilizado.

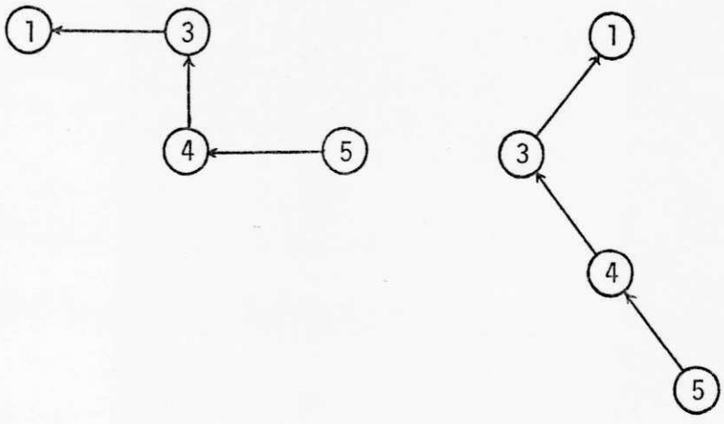
O motivo principal para a utilização da árvore "SPANNING" é que, se for seguida a idéia de Edmonds e Johnson [11], ela garante a conexidade do grafo durante todo o traçado da rota.

Exemplo: Utilizando o grafo G e a árvore T dados no último exemplo.

1 → 2 → 4



1 → 2 → 4 → 6 → 5



E por fim a rota de Euler

1 → 2 → 4 → 6 → 5 → 4 → 3 → 1

Pode-se notar que o grafo se mantêm conectado durante toda a construção da rota.

Essa idéia será utilizada, com algumas modificações, na produção do algoritmo da ROTAFINAL. As modificações se devem principalmente à aplicação prática do algoritmo. Como o

sistema final será utilizado para a distribuição de bens e serviços, e mais especificamente na coleta de lixo, surgem algumas restrições de trânsito e manobra, que devem ser levadas em conta. Essas restrições e suas soluções serão apresentadas com maiores detalhes mais adiante.

Serão apresentados agora os algoritmos relativos à montagem da árvore "SPANNING" e da ROTAFINAL.

7.1 - Árvore "SPANNING"

Por trabalhar com grafos mistos, e devido a proposta de Edmonds e Johnson de, se possível, utilizar sempre os ramos antes dos arcos, o algoritmo que monta a árvore "SPANNING" possui a seguinte filosofia:

- A árvore é desenvolvida através dos arcos o máximo possível, até um ponto onde é encontrado um vértice que não possua mais nenhum arco não utilizado. Isso é feito no algoritmo pela subrotina ESTENDE. Nesse ponto continua-se o desenvolvimento da árvore através dos ramos, até que um novo vértice contendo um arco não utilizado seja encontrado, e então volta-se a estender novamente através dos arcos. Esse processo é repetido até que a árvore contenha todos os vértices.

Os algoritmos que montam a árvores "SPANNING" utilizam os seguinte conjuntos:

T : conjunto que conterá os arcos e ramos da árvore;

R : conjunto contendo o vértice inicial i_0, e os vértices pertencentes à árvore, nos quais incidem ramos.

E(i): conjunto dos ramos incidentes com o vértice i;

A(i): conjunto dos arcos incidentes no vértice i.

```
      Subrotina Estende(k,T,R)
1.    FILA ← {k} ;
2.    WHILE(FILA≠Ø) DO
        Begin
3.        i ← Primeiro(FILA);
4.        Elimine i da FILA;
5.        WHILE(A(i)≠Ø) DO
            Begin
6.            Escolha um arco (h,i)εA(i);
7.            Elimine (h,i) de A(i);
8.            IF (h não é incidente com nenhum arco de T) THEN DO
                Begin
9.                T ← T U {(h,i)} ;
10.               Adicione h na FILA;
11.               IF (E(h)≠Ø) THEN DO R ← R U {h}
                End
            End
        End
```

Árvore "Spanning"

```
1.   FOR i ← 1 UNTIL n DO
       Begin
2.       E(i) ← conjunto dos ramos incidentes com o vértice i;
3.       A(i) ← conjunto dos arcos incidentes no vértice i;
       End
4.   T ← ∅;
5.   R ← {i_o};
6.   ESTENDE(i_o,T,R);
7.   WHILE (existir um r∈R, com E(r)≠∅) DO
       Begin
8.       R ← R {r};
9.       j ← r ;
10.      WHILE (existir k≠r, com  j,k∈ E(j)) DO
           Begin
11.          Elimine {j,k} de E(j) e E(k);
12.          IF(k não é incidente com algum arco em T) THEN DO
               Begin
13.              T ← T ∪ {k,j} ;
14.              IF(E(k)≠∅) THEN DO R ← R ∪ {k};
15.              ESTENDE(k,T,R)
               End
           End
       End
```

7.2 - Algoritmo da Rota

No algoritmo da rota, os vetores que contêm a árvore, os ramos e os arcos, são utilizados somente na montagem das listas associadas a cada vértice. A notação utilizada será a seguinte:

- ARBOR(i) : arco ou ramo da árvore associado ao vértice i;
- RAMOS(i) : ramos incidentes com o vértice i, excetuando o ramo pertencente à árvore, se for o caso;
- ARCOS(i) : arcos saindo do vértice i, com exceção daquele pertencente à árvore, se existir.

Utilizando essas notações, a filosofia do algoritmo ROTAFINAL pode ser escrita como sendo: partindo de um vértice inicial que foi escolhido como raiz da árvore "SPANNING", toda vez que um vértice i é atingido, a seguinte prioridade deve ser respeitada para se sair de i.

- (i) sair por algum ramo RAMOS(i), que ainda não foi utilizado;
- (ii) sair por algum arco ARCOS(i), ainda não utilizado;
- (iii) sair pelo ARBOR(i).

Sendo utilizada uma boa estrutura, como listas multiencadeadas, é muito simples controlar a sequência desejada para os ramos e arcos, com relação à rota. Basta preparar, para cada vértice i, uma LISTA(i) que faça um encadeamento compatível com as prioridades citadas para se montar a rota. Essas lis

tas terão portanto os ramos, se existirem, como primeiros elementos, depois os arcos e por fim o arco, ou ramo, da árvore.

Será utilizado um vetor, chamado ROTA, para armazenar os arcos e ramos pertencentes à rota.

Subrotina Montalista (i)

Para cada vértice i, monta-se uma LISTA(i), seguindo as prioridades já citadas. A estrutura lista encadeada é esolhida por ser muito conveniente ao objetivo em questão.

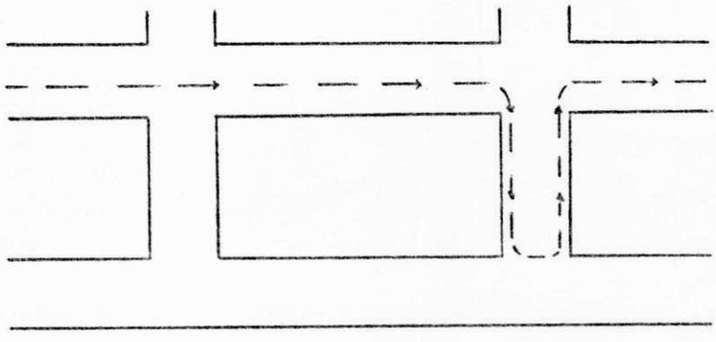
```
1.  WHILE (existirem j∈RAMOS(i)) DO
      Begin
2.      LISTA(i) ← j;
3.      RAMOS(i) ← RAMOS(i)\{j} ;
      End
4.  WHILE (existirem k∈ARCOS(i)) DO
      Begin
5.      LISTA(i) ← k ;
6.      ARCOS(i) ← ARCOS(i)\{k} ;
      End
7.  IF(i≠ vértice inicial) THEN DO LISTA(i) ← ARBOR(i)
```

Algoritmo ROTAFINAL

```
1.   FOR(todos os vértices i) DO
2.      MONTALISTA(i);
3.   i ← vértice inicial;
4.   WHILE (LISTA(i)≠Ø) DO
       Begin
5.       ROTA ← Elemento (i,j) que encabeça a LISTA(i);
6.       Elimina (i,j) da LISTA(i) ;
7.       IF((i,j) for ramo) THEN DO Elimina (j,i) da LISTA(j);
8.       i ← j
       End
9.    Imprime a ROTA
```

7.3 - O Problema do Retorno em U

Particularizando o Problema do Carteiro Chinês para o caso onde são tuilizados veículos, como na coleta de lixo, entrega de gás e outros, será encontrada uma restrição relacionada com o problema de manobra do veículo. Esse problema ocorre quando, num determinado ponto, o trecho que o veículo deve percorrer é igual ao trecho anterior percorrido, somente que em sentido contrário. Por exemplo:



A estrutura escolhida para traçar a rota, irá contribuir na solução desse tipo de problema, que será chamado de retorno em U. Apesar de não haver esgotado o assunto, não foi encontrada uma solução eficiente para o problema, optando-se então por uma solução heurística, que pode não otimizar, mas com certeza diminuirá o aparecimento de U na rota.

A idéia é toda baseada na estrutura das listas . Como foi visto, cada vértice possui uma lista onde são armazenados, primeiro os ramos, depois os arcos e por fim os arcos e ramos da árvore. Tem-se então para os vértices i e j o seguinte:

LISTA(i)

RAMOS(i)
ARCOS(i)
ARBOR(i)

LISTA(j)

RAMOS(j)
ARCOS(j)
ARBOR(j)

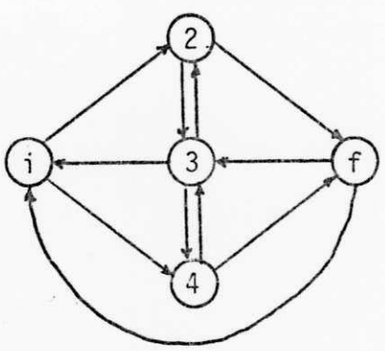
Tem-se, para cada U que surgir, duas oportunidades de evitá-lo. A idéia em síntese é a seguinte:

- Ao se atingir o vértice i, é feita uma análise do elemento que encabeça a LISTA(i). Esse elemento fornece qual é a lista para onde deve-se caminhar em seguida , podendo-se verificar portanto se vai ocorrer um U ou não, através da análise do primeiro elemento da LISTA (j). Caso esse elemento seja (j,i), basta fazer uma pesquisa nessa lista e tentar trocar dois elementos de posição, o primeiro com algum outro conveniente. Isso não é difícil de ser feito, visto que a estrutura de lista, implementada com apontadores, facilita a localização de

A proposta que é feita para a resolução desse problema, é a adição de um arco (f,i) entre o vértice inicial e final, com custo c(f,i) = M (valor muito alto).

A idéia de introduzir o arco (f,i) com um custo bem elevado, se deve ao fato de que, no traçado da rota, ele será atravessado uma única vez, permitindo sua eliminação e consequente formação de um caminho, entre o vértice inicial e final, que percorra todos os arcos e ramos do grafo uma única vez.

- Exemplo : Considere o grafo abaixo como sendo a saída do MIXED A.



Considere que a rota fornecida ao final seja:

i → 4

4 → 3

3 → 2

2 → 3

3 → 4

4 → f

f → 3

3 → i

i → 2

2 → f

f → i

seus elementos. Caso não seja possível trocar o elemento (j,i) por outro, uma outra tentativa será feita, agora em relação a LISTA(i). É feita uma pesquisa nessa lista, na tentativa de encontrar um componente (i,k), também conveniente, cuja LISTA(k) possua um primeiro elemento diferente de (k,i), ou possibilite a sua troca.

Verifica-se portanto, que as possibilidades para se evitar a ocorrência de U numa rota são grandes, apenas é necessário fazer uma observação com relação à conveniência da escolha dos elementos para se fazer as trocas.

- É muito importante que a filosofia básica do algoritmo ROTAFINAL não seja esquecida, e porisso na escolha de um elemento para se fazer a troca com o primeiro da lista, é preciso levar em consideração que esses elementos pertençam a um mesmo conjunto de entrada, RAMOS ou ARCOS, e por hipótese alguma deve-se utilizar, para troca, o elemento pertencente à ARBOR.

Sendo de interesse evitar o retorno em U, basta adicionar um novo passo, no algoritmo ROTAFINAL, com as recomendações feitas acima, entre os passos 4 e 5.

7.4 - "ROTA" com Vértice Inicial Diferente do Final

Muitas vezes, ao invés de uma rota com vértice inicial igual ao final, deseja-se encontrar uma "rota" onde os vértices inicial e final são distintos. Isso ocorre principalmente quando, utilizando veículos, a garagem ou depósito fica longe do ponto de origem.

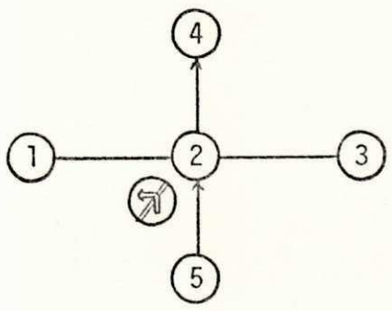
- O arco (f,i) pode ocorrer em qualquer posição da rota , que não interferirá na solução do problema, desde que seja percorrido uma única vez. Isso sempre ocorrerá , pois graças a seu elevado custo, os algoritmos intermediários, por trabalharem minimizando custos, não duplicarão esse arco.

- No exemplo apresentado, basta portanto eliminar o arco f → i, depois de gerada a rota, e tem-se um caminho entre i e f que passa por todos os arcos e ramos uma só vez.

7.5 - A Restrição de Proibido Contornar

Uma observação importante relacionada ao algoritmo ROTAFINAL, é a restrição de trânsito de contorno proibido , que surge com pequena frequência na prática, mas deve e foi levada em conta nesse trabalho. Através de um exemplo será mais fácil explicar tal restrição e apresentar uma solução.

- Exemplo:



- Esse cruzamento com cinco vértices mostra que apesar de, na prática, ser possível para que vem do vértice 5 ir para o vértice 1, visto que o trecho 5 → 2 permite isso e o trecho 2 — 1 é de mão dupla, na realidade, devido ao sinal de proibido contornar à esquerda, isso

não é permitido. Esse problema se torna muito sério em relação ao grafo dado, pois o algoritmo quando atingir o vértice 2 deve tomar conhecimento dessa proibição e respeitá-la.

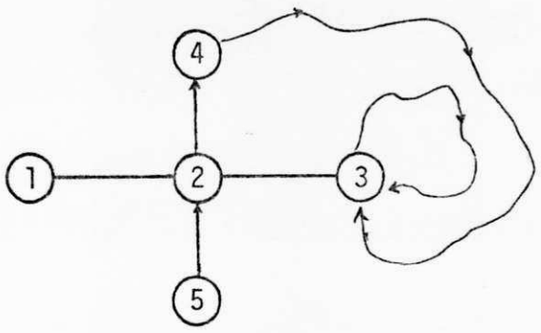
A idéia para solucionar esse problema segue a mesma linha de resolução do retorno em U. E pelos mesmos motivos apresentados para o retorno em U, aqui também será fornecida uma solução heurística, associada às estruturas utilizadas na rota.

Para uma melhor compreensão, o esquema de resolução será apresentado em paralelo ao exemplo acima. Tudo que for escrito entre parênteses estará associado ao exemplo. O esquema é o seguinte:

- Em primeiro lugar devem ser armazenados todos os vértices críticos (2) e respectivos arcos ou ramos que podem causar problemas. Esses arcos e ramos serão denominados entradas ((5,2)) e saídas ({2,1}) críticas.
  Toda vez que um vértice crítico (2) é atingido pela rota, examina-se o arco ou ramo que o atingiu, verificando tratar-se de uma entrada crítica ((5,2)) ou não. Caso não seja, a rota continua normalmente. Em caso positivo, procede-se da seguinte forma: a estrutura lista, associada ao vértice crítico (LISTA(2)), é analisada e se o seu primeiro elemento for uma saída crítica ({2,1}), uma tentativa de troca das posições dos elementos é feita, da mesma forma que no retorno em U, sempre respeitando as prioridades exigidas pela rota. Sendo possível a troca, o problema está resolvido. Caso contrário é acionada uma subrotina que analisa as possí

veis outras saídas ({2,3},(3,4)) do vértice crítico.
O ponto principal desse raciocínio é forçar a saída por uma das escolhidas como possíveis ({2,3}), não importando qual, pois todas serão examinadas, e aplicar um método que encontre o menor caminho entre o vértice atingido (3) e o crítico (2), com a seguinte restrição:

- Se a saída escolhida for um arco, não haverá problemas. Caso seja um ramo, deve-se forçar o método que encontra o menor caminho, a não utilizar esse ramo ({2,3}) na primeira iteração, visando evitar o retorno em U. Será fornecida uma proposta para modificar o algoritmo de Dijkstra, adaptando-o a essa situação.



Feito isso, calcula-se o menor caminho desejado (3→2), no grafo G' = G\{arcos e ramos críticos}.
Esse procedimento é repetido para todas as possíveis saídas não críticas ((2,4)), encontrando-se os novos caminhos (4 → 2).
Os caminhos, mais suas respectivas saídas forçadas, são analisados e escolhido o de menor custo, que será acrescido na rota.

Obs.: Modificação no Algoritmo de Dijkstra

Para se adaptar o algoritmo, descrito no capítulo 4, para a solução acima, basta elmiminar o vértice crítico k do conjunto S no passo 1., fazer uma iteração completa antes do comando WHILE, inserir novamente k em S e prosseguir com o método.

1. $S \leftarrow \{1,\ldots,n\}$;
2. $S \leftarrow S \{k\}$;
3. $L(1) \leftarrow 0$;
4. FOR i ← 2 UNTIL n DO $L(i) \leftarrow \infty$;
5. Encontre $i \in S$ com $L(i) = \min\{L(j) \mid j \in S\}$;
6. IF L(i) = THEN Stop;
7. $S \leftarrow S \{i\}$;
8. FOR (todos os sucessores j de i) DO $L(j) \leftarrow \min\{L(j), L(i)+c_{ij}\}$
9. $S \leftarrow S \{k\}$;
10. WHILE
    Begin
    End

CAPÍTULO VIII

CONCLUSÃO

Nesse trabalho nenhum ponto foi deixado em aberto com relação ao objetivo inicial. Todas as questões levantadas por Santos [20], ou sejam, a geração da rota e o problema de contorno proibido foram solucionadas. Outros pontos, como o retorno em U e a "rota" com vértice inicial e final diferentes, foram analisados visando um maior auxílio ao usuário na distribuição de bens e serviços. Esses problemas também receberam boas soluções, e pode-se apresentar o seguinte quadro com relação a todos os pontos levantados:

(i) Para a geração da rota, algoritmos eficientes foram desenvolvidos, o que possibilitou a implementação de um sistema, desenvolvido por Helédia Costa [4], que ao final, testado com dados das cidades de Ara

caju (Se) e Fortaleza (Ce), apresentou bons resultados;

(ii) O problema da "rota" com vértice inicial diferente do vértice final recebeu solução através da modificação do grafo inicial;

(iii) Para o retorno em U, devido ao fato de nem sempre ser possível minimizar a sua ocorrência, foi apresentada uma solução heurística, que pode não otimizar, mas diminui o aparecimento desse tipo de retorno.

(iv) O problema de contorno proibido também recebeu uma solução heurística, semelhante a do retorno em U.

Devido a essas três soluções heurísticas apresentadas, e ao fato desse trabalho ser aplicado diretamente a setores pré-fixados, serão feitas algumas sugestões para futuros trabalhos que poderão complementá-lo.

- Com relação às soluções heurísticas, ainda não foi feita uma análise do pior caso, associado aos custos adicionais necessários. Por isso fica aqui uma proposta de trabalho que analise outras possibilidades de solução, visando o desenvolvimento de uma teoria que, se possível, resolva o problema de forma eficiente.

- Para que se obtenha um bom resultado, na aplicação desse trabalho em um setor específico de uma cidade, é ne

cessário que, em termos globais, tenha sido feito um bom trabalho em relação a divisão dos setores da cidade. Portanto seria muito interessante o desenvolvimento de um estudo, separado do problema da rota, que tratasse especificamente desse assunto, possibilitando uma posterior aplicação da rota.

A idéia de se fazer o trabalho em separado se deve apenas ao tamanho e complexidade do mesmo.

BIBLIOGRAFIA

01. - BRUCKER, P., "The Chinese Postman Problem for Mixed Graphs". Graphteoretic Concepts in Computer Schience (Ed. H. Noltemeier) Berlim (1981).

02. - BUSACKER, R.G.; GOWEN, P.J., "A Procedure For Determining a Family of Minimal Cost Network Flow Patterns", Operations Research Office, Technical Paper 15.

03. - CHRISTOFIDES, N., "Graph Theory: An Algorithmic Approach", Academic Press, London (1975).

04. - COSTA, H.C.B:, "Rotas para Distribuição de Bens e Serviços: Proposta de um Sistema de Informação", tese de mestrado (1982).

05. - DEO, N., "Graph Theory with Applications to Engineering and Computer Science", Prentice Hall, London (1974).

06. - DERIGS, U., "Duality and Admissible Transformations in Combinatorial Optimization". Report 78-21 Math. Inst. der Universitat zu Koln (1978).

07. - DERIGS, U., "A Shortest Augmenting Path Method for Solving Minimal Perfect Matching Problems". Report 79-06 Math. Inst. der Universitat zu Koln (1979).

08. - DERIGS, U., "On Two Methods for Solving Minimal Perfect Matching Problems". Report 79-19 math. Inst. der Universitat zu Koln (1979).

09. - DIJKSTRA, E.W., "A Note on Two Problems in Connection with Graphs". Num. Math. 1, 269-271 (1959).

10. - EDMONDS, J., "The Chinese Postman Problem", Operations Research 13, Suppl. 1, 377 (1965).

11. - EDMONDS, J.; JOHNSON, E.L., "Matching, Euler Tours and The Chinese Postman". Math. Prog. 5, 88-124 (1973).

12. - FLOYD, R., "Algorithm 97: Shortest Path". Comm ACM 5, 345 (1962).

13. - FORD, L.R.; FULKERSON, D.R., "Flows in Networks", Princeton University Press. Princeton (1962).

14. - FREDERICKSON, G.N., "Approximation Algorithms For Some Postman Problems". J. ACM 26, 3, 538-554 (1979).

15. - HARARY, F., "Graph Theory", Addison-Wesley Publishing Company, Inc. Philippines (1972).

16. - LAWLER, E.L., "Combinatorial Optimization Networks and Matroids", Holt, Rinehart and Winston, USA (1976).

17. - MEI-KO, K., "Graphic Programming Using Odd or Even Points". Chinese Math. 1, 273-277 (1962).

18. - MINIEKA, E., "The Chinese Postman Problem For Mixed Networks", Management Science, 25, 7, 643-648 (1979).

19. - PAPADIMITRIOU, C.H., "On the Complexity of Edge Traversing". J. ACM 23, 3, 544-554 (1976).

20. - SANTOS, A., "Uma Solução Heurística Para o Problema do Carteiro Chines num Grafo Misto, Com Aplicação à Distribuição de Bens e Serviços Públicos", tese de mestrado (1981).