INSTITUTO MILITAR DE ENGENHARIA
PÓS-GRADUAÇÃO EM ENGENHARIA DE TRANSPORTES

Algoritmos para Resolução de Problemas em Redes

Prof.ª Vânia Barcellos G. Campos

Índice

1- Definição e Conceitos Básicos sobre Grafos

Um grafo **G = (X,A)** é uma estrutura composta por um conjunto **X** de elementos chamados vértices ou nós e um conjunto **A** de pares de vértices, chamados arcos ou arestas.

A representação gráfica de um grafo é feita por pontos (vértices) e linhas (arestas) unindo estes pontos.

Quanto as características de seus arcos, um grafo pode ser :

a) Orientado e não orientado

São orientados quando seus arcos possuem uma orientação definida , ou seja , um nó do arco é definido como origem do mesmo e o outro como destino. E, não orientado, quando não existe esta noção de direção.

b) Valorado e não valorado

Um grafo é valorado, quando existem valores atribuídos a cada um dos seus arcos. Exemplo disto ocorre quando se está representando uma rede viária e se atribui a cada arco os valores correspondentes às distância entre interseções (vértices).

c) Planar e não-planar

Um grafo planar é aquele em que os arcos somente se cruzam sobre um nó, podendo consequentemente ser projetado sobre um plano, sem perder suas características. Um grafo não-planar, no entanto, quando projetado sobre um plano, apresenta interseções de arcos não coincidentes com um nó, em função de sua estrutura espacial.

1.1 - Conceitos básicos

a) Um arco que se inicia e termina nele mesmo é chamado de *laço.*

b) Um arco ou um nó é dito *incidente* a um outro nó se este é destino ou origem do arco.

c) *Grau de um nó* é medido pelo número de arcos incidentes a ele. No caso de grafo orientado existe a noção de semigrau interior -número de arcos incidentes interiormente ao nó e semigrau exterior que corresponde ao número de arcos incidentes exteriormente ao nó.

d) Dois vértices são considerados *adjacentes* um ao outro se existe um arco unindo-os.

e) Uma *cadeia* é uma seqüência de arcos (orientados ou não). O tamanho de uma cadeia é o número de arcos que a compõem.

f) Um *caminho* é uma cadeia em que todos os arcos tem a mesma direção.

g) Um *ciclo* é uma cadeia cujo vértice inicial e final é o mesmo (cadeia fechada).

h) Um *circuito* é um caminho cujo vértice inicial e final é o mesmo.

1.2 - Características Estruturais

a) Um grafo é *conexo* quando existe uma *cadeia* entre todos os pares de vértices do grafo (fracamente conexo).

b) Um grafo é *fortemente conexo* quando existe um caminho de cada nó para todos os outros nós do grafo.

c) Grafo *completo* é aquele onde cada nó está conectado com todos os outros nós do grafo.

d) Uma *árvore* é um grafo conexo sem ciclos.

e) Um *subgrafo* é uma parte de um grafo, obtido pela supressão de vértices e dos arcos adjacentes a estes vértices.

f) Um grafo *parcial* de um grafo é obtido pela supressão de arcos deste grafo.

g) Um grafo *trivial* é formado por um único nó.

2 -Conceituação de Rede

De uma forma geral uma rede é um grafo com um ou mais valores associados a cada arco e algumas vezes ao nós.

$G = \{X,A\}$ (grafo)
$R = \{X,A,\alpha\}$ (rede)

onde
X = conjunto de nós ($|X| = n$);
A = conjunto de arcos ($|A| = m$);
α = parâmetros associados aos elementos do conjunto A

Exemplos de Rede:

Nós ou Vértices	Arcos	Fluxo
interseções	rodovias	veículos
aeroportos	linhas aéreas	aviões
estações de bombeamento	dutos	fluidos

Os principais valores associados aos arcos a(i,j) são:

- a capacidade de fluxo *u(i,j)* que corresponde a capacidade máxima de fluxo que pode passar no arco;
- o custo *c(i,j)* no arco que pode ser considerado como um *valor monetário*, a *distância* percorrida ou o *tempo de viagem* no arco.
- fluxo no arco *f(i,j);*

Os problemas de otimização de redes compreendem basicamente:

- Minimização de Redes (árvore mínima)
- Caminho Mínimo
- Fluxo Máximo
- Custo Mínimo
- Roteirização

Os problemas de **Minimização de Redes** são aqueles em que se busca uma interligação de pontos de uma rede de forma que a soma total dos valores (custos) dos arcos utilizados para ligá-los seja a menor possível.

Os problemas de **Caminho Mínimo** compreendem a determinação do caminho ou rota de menor tamanho (distância, tempo ou um custo qualquer) entre dois nós de uma rede.

Os problemas de **Fluxo Máximo** se referem às situações em que se deseja avaliar a quantidade máxima de fluxo que pode ser enviada de um nó de origem a um nó de destino na rede.

Os problemas de **Custo Mínimo** visam determinar os caminhos entre um par de nós (origem e destino) pelos quais deve ser distribuído um determinado fluxo com o menor custo possível.

Os Problemas de **Roteirização** compreendem a busca de um itinerário que se inicia e termina num mesmo ponto da rede passando por determinados pontos ou arcos desta rede.

Para resolver estes problemas utilizam-se procedimentos chamados de **algoritmos de resolução,** descritos a seguir.

3 - Minimização de Redes

Os algoritmos de Minimização de Rede (Minimum Spanning Tree Problem) tratam da determinação da árvore de valor mínimo em problemas de interligação de redes de comunicação, luz, água, esgoto, dutovias, rodovias etc; com o objetivo de atender a todos os pontos de "consumo"(nós da rede) com um consumo mínimo de meios.

Em problemas deste tipo a rede é não orientada. Dois algoritmos podem ser utilizados:

- algoritmo de Kruskal
- algoritmo de Prim

3.1 Algoritmo de Kruskal

Este algoritmo compreende a cada passo a seleção de um arco, iniciando-se pelo arco de menor valor e prosseguindo com a adição de arcos em ordem crescente de valores construindo uma arvore, de modo a não formar ciclos com os arcos já selecionados. O processo se encerra quando a árvore que esta sendo construída possuir n-1 arcos conectados.

Em termos gerais, o algoritmo compreende os seguintes passos:

Passo 0 – Coloque os arcos em ordem crescente de valor. Estes arcos fazem parte de um conjunto de arcos não conectado A*. O conjunto A de arcos conectados é vazio.

Passo 1 – Selecione o menor dos arcos do conjunto A* que não forme um ciclo com os demais e coloque-o no conjunto A. Um arco forma um ciclo quando os vértices deste arco já fazem parte da árvore mínima em construção.

Passo 2 - Se o conjunto A possui n-1 arcos, então pare, os arcos deste conjunto compõem a árvore mínima, caso contrário volte para 2.

Exemplo1:

A rede a seguir representa as possíveis ligações rodoviárias entre seis regiões. Deseja-se verificar quais as ligações que deverão ser implantadas visando interligar todas as regiões, porém, com uma quilometragem total mínima de estradas construídas. Os valores sobre os arcos representam as distancias entre as regiões (x 10^2).

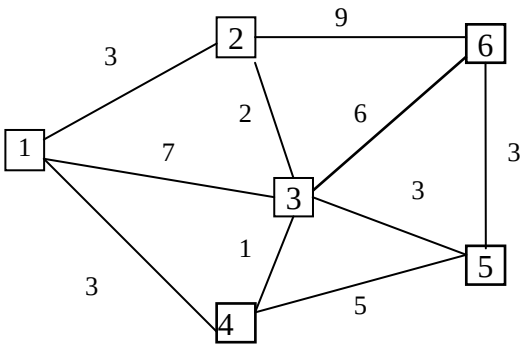


Fig1- Rede do exemplo 1

Passo 0 : Colocando os arcos em ordem crescente de tamanho tem-se:
A* = { (3,4),(3,2),(1,2),(3,5),(6,5),(1,4),(3,6),(3,1),(2,6)}
A = $\varnothing$

Passo 1 : A= {(3,4)}
A* = {(3,2),(1,2),(3,5),(6,5),(1,4),(3,6),(3,1),(2,6)}

Passo 2: Como n= 6 e o número de elementos de A é menor que n-1, então volta-se ao passo2;

Passo 1: A = {(3,4),(3,2)}
A* = { (1,2),(3,5), (1,4), (6,5), (3,6),(3,1),(2,6)}

Passo 2: $|A| \neq$ n-1, passo2
Passo 1: A = { (3,4),(3,2), (1,2)}
A* = {(3,5),(1,4),(6,5),(3,6),(3,1),(2,6)}

Passo 2: $|A| \neq$ n-1, passo2
Passo 1: A = { (3,4),(3,2), (1,2), (3,5)}
A* = {(1,4),(6,5),(3,6),(3,1),(2,6)}

Passo 2: $|A| <$ n-1, passo2
Passo 1: A = { (3,4),(3,2), (1,2), (3,5),(6,5}, o arco (1,4) não pode fazer parte do conjunto pois a sua inclusão na árvore forma um círculo.
A* = {(1,4),(3,6),(3,1),(2,6)}

Passo 2: $|A|$ = n-1, então já encontramos a árvore mínima representada na figura a seguir, que possui um total de $12x10^2$ km:

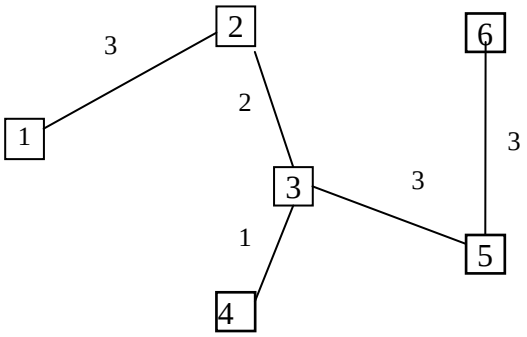


Figura 2 - Rede mínima resultante do exemplo1

3.2 - Algoritmo de Prim

Este algoritmo compreende a associação de nós da rede, buscando a cada passo o nó que esta mais próximo da composição corrente da árvore mínima. Assim como no algoritmo de Kruskal, considera-se um conjunto de nós conectados e um conjunto de nós não conectados.

O algoritmo compreende os seguintes passos:

Passo 1 - Selecione qualquer nó da rede. Identifique o nó que está mais próximo do nó selecionado e inclua este nó e respectivo arco na árvore mínima.
Os nós da árvore mínima formam o conjunto de nós conectados C e os nós restantes o conjunto de não conectados C*.

Passo 2- Identifique o nó do conjunto de nós não-conectados C* que está mais próximo a qualquer um dos nós do conjunto de nós conectados C.
Este processo é repetido até que todos os nós estejam conectados, ou seja, o conjunto C* está vazio.

4 - Algoritmos de Caminho Mínimo

Numa rede qualquer, dependendo das suas características, podem existir vários caminhos entre um par de nós, definidos como origem (*s*) e destino (***t***). Entre os vários caminhos aquele que possui o menor "peso" é chamado de caminho mínimo. Este peso representa a soma total dos valores dos arcos que compõem o caminho e estes valores conforme referenciados anteriormente podem ser: o tempo de viagem, a distância percorrida ou um custo qualquer do arco.

Assim, os algoritmos de Caminho Mínimo determinam a rota de menor tempo, distância ou custo entre um par ou pares de origem e destino.

O *Caminho ótimo* é aquele que apresenta uma seqüência de arcos conectando o nó de origem e o nó de destino de tal forma que a soma dos valores dos arcos no caminho é minimizada.

Se o caminho é mínimo entre um par de nós então qualquer caminho que é parte deste caminho também é mínimo

Existem diferentes formulações para um problema de caminho mínimo que são:

a) de um nó para outro nó;
b) de um nó para todos os outros nós da rede;
c) entre todos os nós da rede e
d) K- caminhos mínimos entre dois nós.

Existem basicamente dois tipos de estrutura de algoritmos para cálculo de caminhos mínimos : árvores e matrizes.

Nos algoritmos em árvore, determinam-se os caminhos mínimos de um nó para todos os outros nós da rede ou simplesmente entre um par de nós. Neste tipo de algoritmo, a solução é obtida construindo-se passo a passo uma árvore de caminhos mínimos. Para se obter os caminhos mínimos de todos os nós para todos os outros nós da rede, utilizam-se os algoritmos com estrutura de matriz e neste caso os caminhos mínimos entre todos os pares de nós são obtidos simultaneamente.

Na literatura pesquisada (ver bibliografia) encontramos os seguintes algoritmos mais comumente utilizados, com estrutura em árvore: Dijkstra e Ford/Moore, e os algoritmos em matriz: de Floyd, Dantzig e ” Double Sweep”.

4.1- Algoritmo de Dijkstra

Utiliza-se este algoritmo para determinar o caminho mínimo de um nó para outro nó ou para todos os outros nós da rede. É considerado um algoritmo bastante eficiente e a sua utilização só é possível quando o valor (tempo, distancia ou outros) atribuído a cada arco da rede é positivo.

Trata-se de um algoritmo iterativo que se utiliza de uma fórmula de recorrência (1) e considera que um nó é “fechado” quando se encontra o caminho mínimo da origem até este nó e aqueles nós cujos caminhos mínimos ainda não foram encontrados são considerados ativos ou “abertos”.

O conceito de fechado e aberto está associado a impossiblidade de se encontrar um caminho melhor do que o já encontrado, assim enquanto o nó não é fechado (ou rotulado) ainda é possível encontrar um caminho de menor valor da origem até este nó.

Fórmula de Recorrência:

$$\mathbf{d(x)^i = min \{ d(x)^{i-1}, d(y) + d(y,x)\}} \quad (1)$$

onde:
$d(x)^i$- tamanho do caminho da origem S até o nó x (na iteração corrente).
d(y) - tamanho do caminho da origem S até o nó fechado (y)
d(y,x)- tamanho do arco (y,x)

Estrutura do algoritmo:

Passo 1: Atribui-se um valor d(x) para cada um dos vértices do grafo sendo:
d(s)= 0 e
$d(x) = \infty$ para todo nó $x \neq s$
Considere y o último nó rotulado (fechado).
Inicialmente o nó **s** é o único rotulado e y= s.

Passo 2 : Para cada nó x não-fechado (aberto) redefine-se d(x) conforme a expressão 1. O nó "aberto" que possuir o menor valor d(x) é "fechado" e faz-se y = x.

Passo 3 : Se o nó de destino **t** foi "fechado" então pare, um caminho de **s para t** foi encontrado. Se o **t** ainda não foi "fechado" volte ao passo2.

Os nós rotulados (fechados) formam uma arborescência de raiz **s** e o caminho de **s** para qualquer nó **x** contido em qualquer caminho é um caminho mínimo de ***s*** para **x.** Assim se deseja-se saber os caminhos com origem em s para todos os demais nós da rede, tem-se que prosseguir com algoritmo até que todos os nós sejam rotulados.

O algoritmo de Dijkstra foi desenvolvido em 1959 e posteriormente Dantizg (1960) e Nicholson (1960) desenvolveram o algoritmo de "duas árvores" de Dijkstra (Dijkstra two-tree algorithm), cuja idéia é construir árvores de caminhos mínimos de um nó origem **s** e de um nó de destino **t** simultaneamente. Logo, ao se chegar a um nó comum o caminho mínimo entre **s** e **t** foi encontrado. Esta idéia mostrou ser interessante do ponto de vista de tempo computacional, conforme foi constatado posteriormente por Helgason (1988).

4.2 - Algoritmo de Ford, Bellman e Moore

Trata-se de uma generalização do algoritmo de Dijkstra, permitindo a utilização de arcos com valor negativo, a partir de uma pequena modificação, observando-se porém que, a existência de ciclos negativos, pois nestes casos, o algoritmo não chega a uma solução.

Neste algoritmo utiliza-se a fórmula de recorrência anterior (1) e não existem nós "fechados", pois a existência de arcos com valores negativos pode fazer com que os caminhos a cada iteração possam ser reduzidos, assim o algoritmo só chega a uma solução quando todos os nós forem analisados e os valores dos caminhos se repetem na iteração seguinte.

Como se trata de uma generalização do algoritmo de Dijkstra as seguintes modificações devem ser feitas neste algoritmo:

1. No passo 2, aplica-se a expressão 1 em todos os vértices, não somente nos nós não rotulados (abertos). Assim, tanto os nós já rotulados quanto os não rotulados podem ter seus valores reduzidos.

2. Se um nó rotulado teve seu valor reduzido, então desconsidere ("desrotule") o arco anteriormente incidente à este nó.

3. Termine o algoritmo somente depois que todos os vértices foram rotulados e se no não houver modificações no valor dos caminhos.

Algumas modificações foram posteriormente sugeridas por outros autores como Steenbrink, Esopo e Pape (1980) para melhorar o desempenho computacional deste algoritmo.

No método de Ford (1956), Moore(1957) e Bellman (1958), um nó pode ser considerado "pivô" inúmeras vezes pois a cada iteração um caminho é descoberto. Esta fonte de ineficiência em tempo de processamento para chegar a solução pode ser reduzida pelas seguintes heurísticas sugeridas por Esopo e Pape:

1 - Se um nó **n** qualquer nunca foi rotulado ele deve ser o último a ser considerado pivô. Caso contrário, se o nó **n** já foi rotulado ou examinado deve ser considerado como um dos primeiros a ser "pivoteado".

2 - A razão para se colocar um nó já examinado de pivô é para reexaminar aqueles que alcançam **n** imediatamente, reduzindo o tempo de chegar a solução final.

Resumo do procedimento de Esopo e Pape:

1. $d(s)= 0 \Rightarrow$ **s** é o nó rotulado (pivô), então y=s;

2. min d(s,j) $\Rightarrow$ **j** é o novo pivô, então y=j;

3. se algum nó que foi pivô é modificado então é o próximo a ser avaliado. Senão toma-se o nó de valor mínimo como pivô.

4.3 Caminho mais Confiável

Em algumas circunstâncias pode ser interessante verificar qual o caminho mais confiável a ser utilizado. Nestes casos, os valores associados a cada arco se referem a probabilidade de sucesso de um evento qualquer.

Seja R=(X,A) uma rede orientada representando uma rede de comunicação tendo-se:

$P_{i,j}$ = probabilidade de que o arco (i,j) $\in$ A estar ativo em determinado instante onde $0 \leq P_{ij} \geq 1$

C' = caminho de *s* para *t*

Considerando-se a independência dos eventos:
Prob{ C' estar operativo } = $\prod_{(i,j)\in C'} P_{ij}$

Exemplo 2:

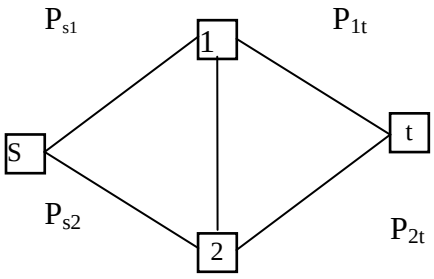


Figura 3 - Rede de comunicação (Exemplo 2)

Probabilidade C'(s,1,2.t) estar ativo = P_{s1}. P_{12}. P_{2t}

Assim determinar o caminho mais confiável de s a t em R equivale a maximizar $\prod_{(i,j)\in C'} P_{ij}$

Seja P = $\prod_{(i,j)\in C'} P_{ij}$

Fazendo-se:

$$\log P = \log \prod_{(i,j)\in C'} P_{ij} = \sum_{(i,j)\in C'} \log P_{i,j}$$

Assim, $\max P = - \min \sum_{(i,j)\in C'} \log P_{i,j}$

$$= \min \sum_{(i,j)\in C'} -\log P_{i,j}$$

Desta forma, o problema acima pode ser entendido como um problema de caminho mínimo.

Num problema de encontrar um caminho de menor risco, deve-se atribuir a cada arco a probabilidade de não ocorrência de algum tipo de evento de risco. E assim maximizar a probalidade de não ocorrência do evento.

4.4 Algoritmo de Floyd

Trata-se de um algoritmo que utiliza matrizes para determinar os caminhos mínimos entre todos os pares de nós da rede. No algoritmo de Floyd são feitas **n** iterações que corresponde ao número de nós da rede . A cada iteração corresponde uma matriz **n** x **n** cujos valores são modificados utilizando também uma fórmula de recorrência (2) .

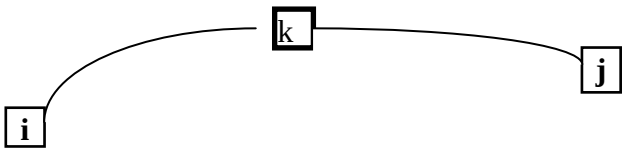
Assim trabalha-se com **K** matrizes de **n** x **n** sendo K= 1,2,3 ... n onde cada valor da matriz é definido como :

$$\mathbf{d_{ij}^{\,k} = \min \{ d_{ik}^{\,k-1} + d_{kj}^{\,k-1} , d_{ij}^{\,k-1} \}} \qquad (2)$$

onde $\mathbf{d_{ij}^{\,k}}$ é o caminho entre os nós **i** e **j** na k-ésima matriz de iteração.

Determina-se inicialmente uma matriz D^0 cujo os valores correspondem aos tamanhos dos arcos se estes existem senão os valores são **∞** . Depois calcula-se D^1 de D^0 e D^2 de D^1 até se obter D^n de D^{n-1} . A matriz D^n é a matriz final que apresenta os caminhos mínimos entre todos os nós da rede.

A idéia básica deste algoritmo é verificar a cada iteração se a inclusão de um nó **k** intermediário no caminho de i para j pode reduzir o tamanho de um caminho já determinado:



Estrutura do Algoritmo

Passo 1 - Numere os vértices do grafo de 1,2..n. Defina a matriz D^0 , cujos valores $\mathbf{d^0_{ij}}$ correspondem ao tamanho (valor) dos arcos **i,j** se existir o arco no grafo; caso contrário considere $\mathbf{d_{ij} = \infty}$, e faça os elementos da diagonal da matriz, $\mathbf{d_{ii}=0}$ para todo i.

Passo 2 - Para cada **k = 1..n** determine sucessivamente os elementos da matriz D^k a partir dos elementos da matriz D^{k-1} utilizando a expressão 2.

Este processo é repetido até **k=n,** e neste caso o valor do caminho mínimo de todos os pares i,j do grafo estão definidos na matriz $\mathbf{D^n}$.

Note que $\mathbf{d_{ii}^{\kappa} = 0}$ para todo i e para todo k, consequentemente, os elementos da diagonal das matrizes não necessitam ser calculados. Além disso, $\mathbf{d_{ik}^{k-1} = d_{ik}^{k}}$ e $\mathbf{d_{ki}^{k-1} = d_{ki}^{k}}$ para todo i= 1,2..n. Ou seja, os elementos da linha e da coluna k da matriz $\mathbf{D^k}$ são iguais ao da matriz D^{k-1}, isto acontece porque o vértice/nó k não pode ser um nó intermediário de um caminho que se inicia ou termina nele mesmo, desde que não existam ciclos negativos. Assim, em cada matriz D^k somente (n-1)(n-2) elementos que não estão nem na diagonal nem na k-ésima linha e coluna precisam ser calculados.

Para se identificar os nós que fazem parte dos caminhos cujos valores são dados pela matriz D^n , tem-se por opção guardar a cada iteração os arcos do caminho ou utilizar uma matriz que guarda a cada iteração o penúltimo nó que forma aquele caminho, Logo, a partir desta matriz por um processo de roteamento pode-se identificar os nós que formam aquele caminho.

Matriz de Roteamento:

Também chamada de matriz de uni-roteamento, a Matriz de Roteamento permite a descrição do caminho mínimo entre cada par de vértice, e baseia-se no princípio de que um vértice k pertence a um caminho dij se e somente se $d_{ik} + d_{kj} = d_{ij}$.

Considerando-se que a matriz de roteamento é formada pelos penúltimos vértice dos caminhos entre os pares de vértice do grafo e definido-se este elemento como k , faz-se:
r_{ij} =k (elemento da matriz de roteamento)

d_{ij} = ?
r_{ik} = ...r_{ikm} =i

Ou seja, vai-se substituindo o penúltimo vértice a cada caminho até que se chega ao nó inicial ***i*** do caminho procurado.

Se considerarmos que um caminho tem **m** nós intermediários, iniciando-se pelo penúltimo k, vamos chegar ao nó de origem , ao encontrarmos o elemento k_m.

Centro de um Grafo
O centro de um grafo corresponde ao vértice **x** do grafo que apresenta a menor das distâncias máximas aos demais vértices do grafo. Este centro é definido a partir do resultado do algoritmo de Floyd, verificando-se na ultima matriz (D^n **) para cada vértice** qual o vértice que tem o menor valor da maior distância aos demais vértices do grafo.

Mediana de um grafo
A mediana de um grafo corresponde a um vértice **x** com o menor somatório das menores distancias aos demais vértices do grafo. Para obtê-la utiliza-se o algoritmo de Floyd somando-se os valores de cada linha da matriz final do algoritmo e o vértice que corresponde a mediana é aquele que tem a menor soma.

4.5 Algoritmo de Dantzig

O algoritmo de Dantzig apresenta as mesmas características do algoritmo anterior, muito embora trabalhe a cada iteração com matrizes de tamanhos diferentes.

Neste algoritmo utiliza-se **n** matrizes para se chegar ao resultado final, porém as matrizes são de tamanhos **k** x **k** e, como no algoritmo anterior a matriz final $\mathbf{D^n}$, ou seja, para **k = n** , o i,j-ésimo elemento determina o tamanho do menor caminho do nó **i** ao nó **j.**

Estrutura do algoritmo:

Passo 1: Numere os vértices do garfo de 1,3..n. Defina a matriz $\mathbf{D^0}$, conforme descrito para o algoritmo de Floyd.

Passo 2 : Para k = 1..n determina-se a cada iteração os elementos de uma matriz $\mathbf{D^k}$ = **k x k** da seguinte forma:

$d^k_{kj} = \min \{ d^0_{ki} + d^{k-1}_{ij} \}$ para: i= 1,2..k-1 e j = 1,2..k-1 (3)
$d^k_{ik} = \min \{ d^{k-1}_{ij} + d^0_{jk} \}$ para: i= 1,2..k-1 e j = 1,2..k-1 (4)
$d^k_{ij} = \min \{ d^k_{ik} + d^k_{kj} , d^{k-1}_{ij} \}$ para i= 1,2..k-1 e j = 1,2..k-1 (5)

Para identificar os nós que compõem cada caminho, utiliza-se o mesmo processo de roteamento definido no algoritmo de Floyd.

4.6 Algoritmo de k-Caminhos Mínimos

O objetivo deste algoritmo é definir mais de um caminho mínimo entre uma origem e um destino. Assim podemos obter k- caminhos mínimos em ordem crescente de tamanho. Este tipo de problema é importante na medida em que algumas vezes desejamos verificar mais de uma opção de caminho para um determinado transporte .

Estes algoritmos também trabalham com matrizes e como nos anteriores utilizam as operações de adição e minimização, efetuadas com conjuntos (vetores) de k distintos números representando o conjunto de caminhos entre dois vértices.

Entre os algoritmos existentes para k-caminhos mínimos podemos citar: Double-Sweep, os de Floyd e Dantzig generalizados .

4.7- Desempenho dos Algoritmos de Caminhos Mínimos

Os algoritmos que utilizam matrizes têm a vantagem de serem geralmente mais fáceis de serem codificados num programa de computador. Além disso, a eficiência no tempo computacional é exatamente determinável e independente da estrutura da rede.

É bastante difícil comparar exatamente a eficiência no tempo computacional entre algoritmos em árvore e matriz.

Quando se deseja o caminho entre todos os nós e quando todos os nós estão diretamente conectados com todos os outros nós, os algoritmos em matriz são mais eficientes que os algoritmos em árvore. Mas quando os nós estão diretamente conectados com poucos nós e quando somente o caminho entre um subconjunto de nós é desejado, então os algoritmos em árvore são mais eficientes quanto ao tempo computacional. Em geral os algoritmos em árvore parecem ser mais úteis que os algoritmos em matriz para redes de transporte.

Conforme se verificou todos os algoritmos de caminho mínimo consistem essencialmente de duas operações: adição e minimização. Para analisar a complexidade computacional destes algoritmos necessita-se de alguns meios para comparar operações de adição com operações de minimização. Certamente que estas operações variam com os computadores utilizados, por conveniência assume-se que estas duas operações têm equivalentes tempos computacionais.

Pode-se concluir que dado o número de operações necessárias os algoritmos apresentados anteriormente tem a seguinte ordem de complexidade:

Dijkstra $=3n^2/2$
Ford $= 1\ ½\ n^3$
Floyd $= 2n^3$
Dantzig$= 2n^3$

5 - Algoritmos de Fluxo Máximo

A utilização destes algoritmos tem por objetivo verificar a capacidade máxima de fluxo em uma rede a partir de um nó origem a um nó de destino. Nestes casos, cada arco possui um valor que indica a capacidade máxima de fluxo que pode passar por ele (limite superior) e, dependendo da rede ou do objetivo da análise, há um outro valor que indica o fluxo mínimo (limite inferior) que deve passar pelo arco .

Um dos primeiros algoritmos se deve a Ford e Fulkerson (1956), os algoritmos que surgiram posteriormente visavam melhorar o desempenho computacional deste algoritmo. Além disso, algumas modificações foram também introduzidas na forma de trabalhar a rede como foi o caso do algoritmo de Dinic (1970) que introduziu o conceito de redes em camadas e Mallhotra (1978) que introduziu o conceito de potencial de um nó.

A idéia básica de um algoritmo de fluxo máximo é encontrar "caminhos de aumento de fluxo" de uma origem **S** para um nó de destino **T** e alocar nestes caminhos a maior quantidade de fluxo possível.

5.1 Conceitos Básicos

Para resolução de um problema de fluxo máximo, algumas considerações conceituais se fazem necessárias:

1- *Lei de Kirschhoff ou lei de Conservação de Fluxos :* " a soma dos fluxos que entram em um nó é igual a soma dos fluxos que saem deste nó".

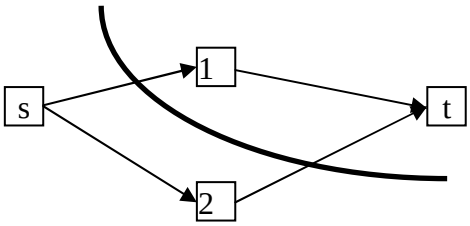
2- *Fluxo viável* é um fluxo que obedece a lei acima e as restrições de capacidade dos arcos.

3 -*Corte é* definido como qualquer conjunto de arcos orientados contendo no mínimo um arco de cada um dos caminhos possíveis entre um nó de origem e um nó de destino num grafo, de tal forma que se este conjunto de arcos for retirado do grafo conexo divide-o em duas componentes conexas.

O valor de qualquer corte, é dado pelo somatório das capacidades dos arcos do corte na **direção s→t**. O valor de um corte representa um limite superior para o Fluxo máximo, e o corte de valor mínimo define o Fluxo máximo no grafo.

Um corte C em **G(X,A)** separando **s** e **t** é um conjunto de arcos **(x,x')** onde **s ∈ x** e **t ∈ x'**. A capacidade do corte (x,x') é c(x,x').

Por exemplo, na figura abaixo o conjunto de arcos C = {(s,1) e (2,t)} representam um corte onde x = {s,2} e x'={1,t}.



Com base nos conceitos acima, os problemas de Fluxo Máximo podem ser traduzidos em um problema de Programação Linear numa rede (X,A,c) da seguinte forma:

max V

Sujeito à:

Para o nó origem :

$$\sum_i f_{si} - \sum_i f_{is} = V$$

Para o nó de destino:

$$\sum_i f_{ti} - \sum_i f_{it} = -V$$

Para qualquer nó ≠ s ou t:

$$\sum_i f_{ji} - \sum_i f_{ij} = 0$$

Para todo arco:

$$0 \leq f(i,j) \leq c(i,j)$$

Onde:
f(i,j) - fluxo no arco (i,j)
c(i,j) - capacidade de fluxo do arco(i,j)
V - fluxo máximo na rede

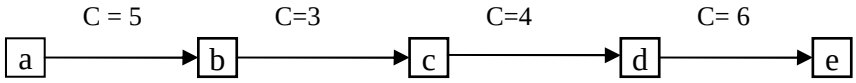
As três primeiras restrições referem-se à lei de conservação de fluxo e a terceira a de viabilidade de passagem de um fluxo num arco qualquer do grafo.

O princípio geral de um algoritmo de Fluxo Máximo é a determinação de caminhos de aumento de fluxo, conforme veremos a seguir.

5.2 - Algoritmo de Aumento de Fluxo

O procedimento básico deste algoritmo consiste no desenvolvimento de uma árvore de caminhos (uma arborescência), ou seja, vários caminhos a partir de uma origem **S ,** ao longo dos quais os fluxos nos arcos podem ser aumentados. Se o destino **T** está incluído nesta arborescência, então o caminho que inclui **T** será um caminho de aumento de fluxo. Se **T** não está incluído na arborescência, então não existe possibilidade de aumentar o fluxo de **S** para **T.**

Encontrar um caminho de aumento de fluxo entre S e T significa encontrar um caminho no qual se possa alocar uma quantidade de fluxo entre estes dois nós. Esta quantidade de fluxo viável no caminho encontrado é definida pela capacidade do "arco gargalo" deste caminho, ou seja, pela capacidade do arco de menor capacidade do caminho No grafo abaixo, a capacidade do caminho {a,b,c,d,e} é de 3 unidades definida pelo arco gargalo (b,c).



Ao se alocar um fluxo num caminho da rede a capacidade dos arcos utilizados é modificada _ passa a ser uma capacidade "residual" dada pela diferença entre a capacidade real do arco e o fluxo corrente no arco. No caso do *arco gargalo*, como o fluxo no caminho é dado pelo valor da sua capacidade, diz-se que ele está saturado e, no caso a sua capacidade residual é zero.

Portanto, para utilização de um algoritmo de aumento de fluxo faz-se necessário entender o desenvolvimento de uma rede residual.

Rede Residual

Intuitivamente, dada uma rede e um fluxo passando nesta rede, a rede residual consiste dos arcos que ainda admitem uma quantidade de fluxo, ou seja, que não foram saturados.

Os arcos numa rede capacitada podem ser divididos em dois conjuntos:

I – conjunto dos arcos que podem ter seu fluxo aumentado;
R- conjunto dos arcos nos quais o fluxo pode ser reduzido;

Conceitualmente, temos:

- conjunto **I** é formado pelos arcos que possuem **f(x,y)< c(x,y),** pois estes arcos podem ter seu fluxo aumentado de **r(x,y)= c(x,y)-f(x,y),** que é a capacidade residual do arco.

- conjunto **R** é formado pelos arcos que possuem **f(x,y)>0**, também chamados de reduzíveis.

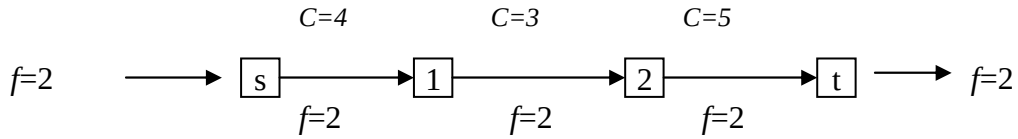
Um arco **a(x,y)** que tenha **0< f(x,y) < c(x,y**) ,pertence a ambos os conjuntos **I e R.**

Dentro destes conceitos uma rede residual é construída da seguinte maneira:

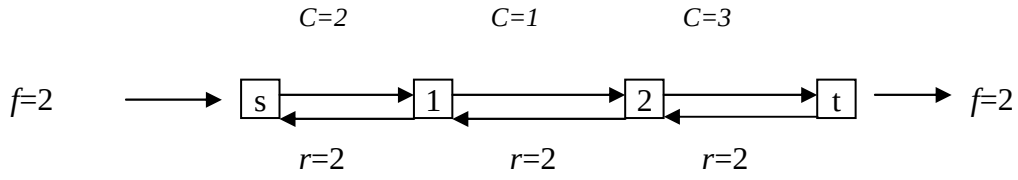
- se um arco **a(x,y)∈I**, então construa este arco na rede residual rotulando-o com a sua capacidade residual **r(x,y)= c(x,y)-f(x,y)** ;

- se **a(x,y) ∈ R**, então construa um arco reverso a(y,x) na rede residual rotulando-o com **r(y,x)= f(x,y),** a capacidade de redução.

Exemplo 3:
Considere a Rede a seguir com um fluxo corrente de f=2



Em função do fluxo corrente a rede residual correspondente é:



Estrutura do algoritmo de aumento de fluxo:

Este algoritmo utiliza um processo de rotulação em que cada nó recebe dois rótulos:

e(x) - representa a quantidade máxima de fluxo adicional no caminho pesquisado de S para o nó x.

p(x) – é o nó antecessor de x no caminho pesquisado.

Quando se encontra o nó de destino **t** o valor de **e(t)** representa a quantidade máxima de fluxo no caminho encontrado.

Passo1 - Construa uma rede residual correspondente ao fluxo corrente (fluxo atual viável).
Comece rotulando s com: **e(s)= ∞ e p(s) = 0.**
Todos os outros nós são inicialmente "não-rotulados" e todos os arcos são "não-marcados".

Passo 2 – Selecione um nó **x** que ainda não foi rotulado. Se não existe nenhum então pare, neste caso não existe possibilidade de encontrar caminhos de aumento de fluxo de ***s* para *t*.** Caso contrário vá para o **passo 3.**

Passo 3 - Se o arco a(x,y) é um arco que pertence ao conjunto I então rotule y com **e(y) = min{e(x),r(x,y)} e p(y)=x.** Se o nó t é rotulado, então um caminho de aumento de fluxo de *s* para *t* foi encontrado, caso contrário volte ao passo2.

5.3 - Algoritmo de Ford / Fulkerson

Trata-se de um algoritmo que aplica um processo de rotulação para definir rotas (de S para T) com a possibilidade de aumento de fluxo. Inicia-se o processo com um fluxo viável (igual a zero, quando não conhecido) de **S** para **T** e procura-se um caminho de aumento de fluxo. Se este caminho é encontrado então envia-se tantas unidades de fluxo quantas forem possíveis por este caminho. Procura-se, então, novamente um outro caminho de aumento de fluxo de **S** para **T** assim sucessivamente, até que não haja nenhum outro caminho e, neste caso, o fluxo corrente é máximo.

Estrutura do algoritmo:

Passo 1 – Considere **s** o nó fonte (origem do fluxo) e **t** o nó de destino. Selecione qualquer fluxo viável de **s para t**, isto é qualquer conjunto de valores de f(x,y), que satisfaça as restrições de capacidade e de continuidade de fluxo (ver pag. 16). Se tal fluxo inicial não é conhecido faça o fluxo inicial f(x,y)=0 para todo arco a(x,y).

Passo 2 - Construa a rede residual relativa ao fluxo corrente.

Passo 3 - Utilize o algoritmo de aumento de fluxo. Se nenhum caminho entre **s e t** for encontrado pare; o fluxo corrente é máximo. Caso contrário aloque este fluxo no caminho encontrado, apague todos os rótulos e volte ao passo 2.

Modificações devidas a Edmonds -Karp

Para assegurar que o algoritmo de Fluxo Máximo chega a uma solução ótima após um número finito de caminhos de aumento de fluxo, Edmond e Karp sugeriram que no passo 3 para encontrar um caminho de aumento de fluxo, os nós fossem inicialmente

pré-rotulados a partir da origem na ordem em que fossem aparecendo. Assim, a origem recebe o rótulo 1, os nós incidentes a esta recebem rótulo 2 e assim por diante.

Se o processo de rotulação é feito desta forma, o caminho de nós marcados (primeiro rotulado, primeiro considerado) conectando qualquer nó de origem para o destino conterá tão poucos arcos quanto possível.

5.4 - Algoritmo de Dinic

Utiliza-se neste algoritmo o conceito de redes em camadas (rede acíclica). O grafo inicial é dividido em níveis de nós a partir da origem **s** que pertence a camada de nível **1** até a última camada a qual pertence o destino **t.** A rede estruturada desta forma não apresenta arcos entre os nós da mesma camada. O objetivo é começar enviando uma quantidade de fluxo de **s** para **t** em caminhos com o menor número de arcos, procurando saturar os arcos até que não exista mais caminhos de aumento de fluxo entre estes nós**.** Neste caso, a rede é então reestruturada, revertendo o sentido dos arcos saturados e definindo-se novamente os níveis dos nós. Inicia-se novamente o processo de enviar fluxos saturando arcos e, assim sucessivamente, até que não se consiga mais restruturar a rede.

Um arco **a(x,y)** numa rede é saturado quando **f(x,y)= c(x,y).** Um caminho P de $s \rightarrow t$ é saturado se no mínimo um dos arcos de P é saturado.

Uma rede, é considerada saturada se todos os caminhos de **s→t** estão saturados. O fluxo total numa rede saturada é um fluxo de saturação, que é o fluxo máximo na rede.

Rede em Níveis

Uma rede em níveis $R_N = (X_N, A_N)$ é uma rede acíclica obtida da rede original, na qual todos os nós X_L são particionados em níveis (ou camadas) $X_1, X_2 .. X_N$ **.**

O primeiro nível, X_1, contém apenas a fonte **s** , ou seja, $\mathbf{X_1 = \{s\}}$.O segundo nível, X_2, contém todos os nós que são imediatamente sucessores de s. O terceiro nível consiste de todos os nós que são imediatamente sucessores dos nós do segundo nível. E assim sucessivamente, o i-ésimo nível X_i consiste de nós que estão à uma distância de i-1 níveis de s.

O nível $\mathbf{X_n}$ corresponde ao nó de destino t, ou seja, $\mathbf{X_n = \{t\}}$.

Assim todo nó de uma rede em níveis está num caminho de **s** para **t** e todos os caminhos são do mesmo cumprimento (em termos do número de arcos).

Estrutura do Algoritmo de Dinic

Passo 1 -Definir uma rede em níveis considerando o fluxo corrente. Se este fluxo existir considera-se a rede residual correspondente. Caso contrário o fluxo inicial é zero.
Se não for possível definir uma rede em níveis, pare; a rede já foi saturada e **F** é o fluxo máximo e vá para o passo 3.

Passo 2 – Saturar a rede em níveis, ou seja, saturar todos os caminhos possíveis de **s→t**, definindo-se um fluxo de saturação $\mathbf{F^r}$ (r= iteração). Volte ao passo 1.

Passo 3 – Somar os fluxos de cada iteração $\mathbf{F = F^1 + F^2}$ ....

5.5 Algoritmo de Dinic/Malhottra/Kumar/Maheshwari

Trata-se do algoritmo de Dinic onde se acrescentou o conceito de potencial do nó. Definida a rede em camadas verifica-se qual nó possui o menor potencial. O potencial de um nó é definido como o menor valor entre a soma da capacidade dos arcos que saem deste nó e a soma da capacidade dos arcos que incidem sobre este nó . O fluxo referente ao menor potencial é definido então como o fluxo a ser "puxado" da origem **s** e o mesmo valor é "empurrado" para o destino **t,** e este passa a ser o fluxo corrente. Desta forma, elimina-se pelo menos um arco saturado, redefinindo-se a rede em camadas e dando início novamente o processo.

Procedimento para encontrar um Fluxo de Saturação na Rede em Camadas

1. Para cada nó **v** da rede em camadas verifica-se o fluxo máximo **(inpot (*v*))** que chegar ao nó e a quantidade máxima de fluxo que pode sair do mesmo (**outpot (*v*)).**

2. potencial de cada nó é então definido como o menor entre o total de entrada (**inpot (v))** e o total de saída (**outpot (v)).**

3. Identifica-se o nó r de menor potencial (**poten (r)**). E toma-se **r** como o nó de referência.

4. Distribui-se nos arcos de saída (sucessores) do nó **r** a quantidade de **fluxo= poten(r)** e puxamos dos arcos anteriores (antecessores) do nó r a mesma quantidade de modo a alcançar os nós **s** e **t**. Feito isto estabelecemos um fluxo de saturação.

5. Retira-se da rede todos os arcos saturados da rede em níveis. O efeito da retirada será a redução do potencial de vários nós da rede e número de arcos. Atualiza-se os valores do potencial dos nós e inicia-se uma nova iteração.

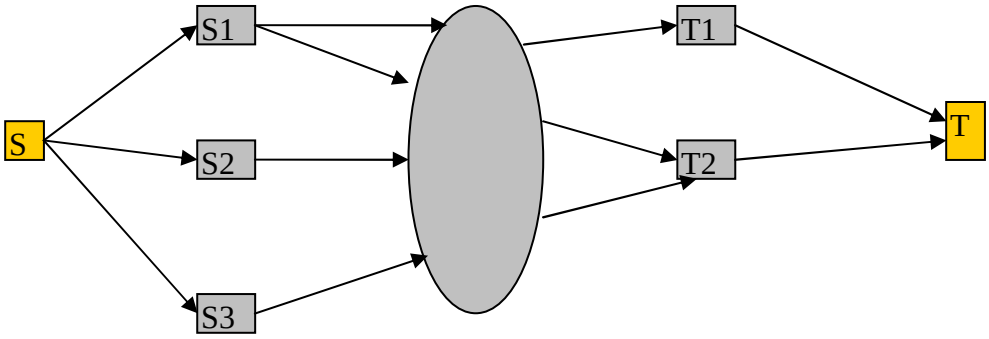
Os procedimentos definidos por Dinic, Malhotra e outros visam otimizar o tempo computacional do algoritmo de Fluxo Máximo.

5.6 Modificações na Estrutura das Redes

Em alguns casos em função do problema em estudo fazem-se necessárias algumas modificações na estrutura das redes de tal forma que se possa aplicar um algoritmo de Fluxo Máximo. Dentre estes se destaca:

1- Várias Fontes e/ou vários destinos:

Considera-se uma fonte única fictícia ligando-se através de arcos também fictícios às fontes (origens) da rede real, o mesmo se faz quando existem vários destinos criando-se um destino único , como na figura abaixo:

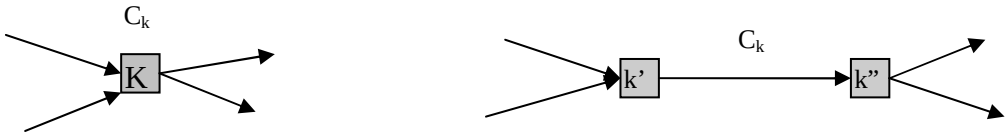


A capacidade dos arcos que ligam a fonte fictícia às fontes reais é igual a capacidade destas fontes quando as mesmas forem centros produtores; o mesmo se define para o destino no caso de serem centros consumidores. Se não houver estes valores então a capacidade destes arcos pode ser considerada como infinita.

2 - Nó com Capacidade

Na maioria dos problemas de Fluxo Máximo as capacidades são atribuídas aos nós, porém podem existir problemas nos quais os nós tenham capacidades, ou mesmo,

custos. Neste caso, estes nós devem ser substituídos por um arco com capacidade igual ao do nó.



3- Redes com limites inferiores

Em algumas situações existem limites inferiores **(l_{ij})** de capacidade associados aos arcos, significando que no mínimo l_{ij} **unidades de fluxo** devem passar do nó **i** para o nó **j**. Isto complica a utilização do algoritmo pois inicialmente não se tem um fluxo viável igual a zero.

Para encontrar um fluxo inicial viável, de tal forma que se possa utilizar o algoritmo de fluxo máximo, estende-se a rede utilizando da seguinte maneira:

1. acrescente a rede dois nós **(u** e **v**);
2. acrescente um arco de **t** para **s** tendo capacidade infinita;
3. seja **a(i,j**) um arco com limite inferior diferente de zero, l_{ij}**;**
4. crie um arco de **u** para **j** com capacidade l_{ij} e um arco de i para **v** com a mesma capacidade;
5. considere a capacidade do arco **a(i,j) = c_{ij} - l_{ij};**

como todos os arcos tem na rede estendida limite inferior igual a zero, o algoritmo de fluxo máximo pode ser aplicado. Resolvendo-se o fluxo máximo de **u** para **v** obtém-se um fluxo viável inicial.

5.7 - Análise dos Algoritmos de Fluxo Máximo

Em termos de redes de transporte os algoritmos de Fluxo Máximo são importantes para uma avaliação da capacidade de uma rede em absorver um determinado fluxo de veículos, ou seja, pode servir para uma primeira análise das condições da rede inclusive para identificar gargalos na mesma.

Pelos algoritmos apresentados podemos observar que as modificações introduzidas foram trazendo uma melhora em termos de desempenho computacional para estes algoritmos. O algoritmo de Ford/Fulkerson apresentava problemas, podendo tomar infinitos passos para convergir para uma solução ótima no pior dos casos. Posteriormente uma modificação foi introduzida por Edmond e Karp (1972) fazendo com que o algoritmo passasse a ter um desempenho computacional de ordem $O(nm^2)$. Edmond e Kasp mostraram que se procurarmos os caminhos de aumento de fluxo

através de caminhos com um número mínimo de arcos, então estaremos dimuindo o tempo de procura.

As modificações introduzidas por Dinic fizeram com que o algoritmo original passasse a ter um desempenho de ordem $O(n^2m)$, tornando-o mais simples e mais eficiente particularmente quando a rede é densa. Posteriormente a modificação introduzida por Malhotra et ali, fez com o algoritmo passasse a ter um desempenho de ordem $O(n^3)$.

6 - Algoritmos de Custo Mínimo

Nos problemas anteriores tratou-se de verificar a quantidade máxima de fluxo que poderia ser enviada de uma origem **s** para um destino **t,** não havendo custo envolvido. Os algoritmos de custo mínimo tratam do problema de enviar uma quantidade qualquer de fluxo ***v*** de **s** para **t** numa rede na qual todos os arcos a(i,j) tem uma capacidade ou limite superior **u(i,j)** tanto quanto um custo **c(i,j)** associado a estes arcos. Este custo pode ser de vários tipos : tempo ,distância, consumo de combustível ou até mesmo uma combinação destes.

Se o fluxo ***v,*** que se deseja alocar é menor que o fluxo máximo **F** para uma determinada rede, então podem existir diferentes formas de distribuir este fluxo na rede. O objetivo dos algoritmos de custo mínimo é , então, encontrar os caminhos de fluxo que minimizam o custo total.

Para resolver um problema desta natureza encontramos os seguintes algoritmos: Ford/Fulkerson ,"Out of Kilter" e o de Busacker e Gowen.

6.1 - Algoritmo de Busacker e Gowen

Trata-se de um procedimento iterativo, que busca alocar uma quantidade de fluxo **V** com o menor custo possível utilizando, a cada iteração, um algoritmo de caminho mínimo para definir o caminho de custo mínimo onde é alocado o máximo de fluxo possível respeitando as restrições de capacidade de cada arco. Após a alocação deste fluxo utiliza-se um procedimento de modificação da rede para definir um novo caminho mínimo no qual será alocada uma nova quantidade de fluxo máximo possível e assim sucessivas vezes, até que se tenha atingido o fluxo **V** desejado, ou até que não se encontre nenhum caminho para se alocar o fluxo que ainda não foi alocado.

Etapas do algoritmo:

Passo1 - Encontra-se um caminho mínimo de **s** para **t** utilizando um algoritmo de caminho mínimo e vá pra o passo2. Se não for possível encontrar um caminho - pare, não há solução para o problema.

Passo 2 - Envia-se tantas unidades de fluxo quantas forem possíveis de **s** para **t** neste caminho. Se o fluxo **v** foi atingido - pare. Caso contrário, vá para o passo 3

Passo 3 - Modifica-se a rede em função do fluxo alocado em cada caminho encontrado e volte ao passo 1.

Rede Modificada

Chama-se de Rede modificada a rede **G*** com a mesma estrutura de G mas com as capacidades modificadas $\mathbf{u^*_{ij}}$ e os custos modificados $\mathbf{C^*_{ij}}$ definido como segue.

1- Se existe um fluxo $\mathbf{f_{ij} \neq 0}$ **alocado** num arco qualquer **a(i,j),** constrói-se um arco fictício de sentido inverso com capacidade igual ao fluxo f_{ij} ; isto é :

$$\mathbf{u^*_{ji} = f_{ij} \qquad se\ f_{ij} > 0;}$$

2- Associa-se à este arco fictício um custo $\mathbf{-c_{ij}}$, ou seja, o mesmo custo do arco existente porém com valor negativo. Este custo é negativo porque a utilização de um arco de sentido inverso significa uma redução do fluxo que passa no arco original e, portanto trata-se de uma redução de custo.

$$\mathbf{c^*_{ji} = -c_{ij} \qquad se\ f_{ij} > 0}$$

3- Se num arco qualquer está alocado um fluxo $\mathbf{f_{ij},}$ e este não está saturado, então, a capacidade deste arco passa a ser capacidade residual $\mathbf{u^*_{ij} = u_{ij} - f_{ij}}$ **.** O custo de enviar uma unidade de fluxo no arco não saturado é o mesmo custo do arco original. Isto é:

$$\mathbf{c^*_{ij} = c_{ij} \qquad se\ f_{ij} \geq 0}$$

Esta forma "incremental" de alocar fluxos foi originalmente proposta por Busacker e Gowen. Este procedimento é também chamado de método dual de resolver problema de custo mínimo, uma vez que a primeira solução viável obtida é uma solução ótima.

6.2 - Análise dos Algoritmo de Custo Mínimo

Em termos de planejamento de sistemas de transporte, os algoritmos de custo mínimo podem ser utilizados para definir a melhor distribuição de tráfego considerando que sob o ponto de vista de um conjunto de usuários de uma rede, ter-se-ia um ganho no custo médio se o volume fosse distribuído conforme definido pelo algoritmo.Também,

sob o ponto de vista de uma indústria, este algoritmo poderia ser utilizado para definir a melhor forma de transporte numa rede (unimodal ou multimodal) para chegar com o produto ao destino (centro consumidor/depósito)

7 – Roteirização

Um problema de roteirização compreende a definição de um itinerário a partir de um ponto de uma rede passando por vários outros pontos desta rede e retornando ao ponto inicial.

O ponto inicial pode ser entendido como um depósito de uma indústria ou transportadora e, os outros pontos da rede como sendo os clientes (pontos de entrega ou coleta). Assim, um caminhão sai do depósito com mercadorias que devem ser entregues aos clientes a partir de um itinerário programado que compreende a seqüência de clientes/entregas a serem feitas e retorna ao depósito inicial.

Existem diferentes métodos de roteirização que compreendem heurísticas para se chegar a uma solução próxima da ótima, um destas heurísticas é conhecida como o problema do caixeiro viajante.

7.1 Problema do Caixeiro Viajante

Um itinerário ótimo é aquele que minimiza a distância ou o tempo total percorrido. Para se obter este itinerário/ roteiro utiliza-se um algoritmo conhecido como Problema do Caixeiro Viajante. Trata-se de uma heurística de resolução do problema que compreende a construção de uma "tour" através da inserção de pontos a cada iteração.

Passos do Algoritmo:

Passo 0 - *Selecione um vértice **i** qualquer da rede e identifique o vértice **j** mais próximo a este e forme uma "subtour": **i - j – i**.*

Passo 1 - *A cada iteração encontre um vértice **k** que não esteja na "subtour" e que está mais próximo a qualquer vértice da "subtour". Se houver mais de uma possibilidade de inserção deste vértice k na " subtour" vá para o passo 2.*

Passo 2 – *Identifique o arco (i,j) na "subtour"que minimiza a relação : d(i,k) + d(k,j) – d(i,j). Insira este vértice k entre i e j _ ou seja , substituindo o arco (i,j) pelos arcos (i,k) e (k,j), e volte ao Passo 1. Repita este processo até que todos os vértices (clientes) façam parte da " tour".*

Observe que no passo 2 (iterativo) a inserção do nó cada se faz removendo o arco (i,j) de forma a minimizar o total de acréscimo com a inclusão de (i,k) e (k,j) na tour.

Bibliografia:

BOAVENTURA NETTO, P. O. (1996) *Grafos: Teoria, Modelos, Algoritmos* , Editora Edgar Blucher Ltda, São Paulo.

BAZARAA M.S., JARVIS J.J., SHERALI H. D., 1990, *Linear Programming and Network Flows*, 2nd. Ed., John Wiley & Sons.

EDMONDS,J. E KARP, R.M., 1972, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", J.ACM 19 ,pp248-264.

KENNINGTON J. L., HELGASON R.V., 1988, *Algorithms for Network Programming*, John Wiley & Sons.

MINIEKA E.(1990)*Optimization Algorithms for Networks and Graphs*, Ed. Marcel Dekker Inc. 1st and 2nd ed.

NEWELL G.F., 1980, *Traffic Flow On Transportation Network*, MIT Press.

POTTS R.B. , OLIVER R.M., 1972, *Flows in Transportation Networks*, Academic Press , New York.

SHEFFI Y., 1985, *Urban Transportation Network : Equilibrium Analysis with Mathematical Programming Models*, Prentice Hall Inc., Englewood Cliffs, N.J.

SHIER D. R., 1976, "Interactive Methods for Determining the K- Shortest Paths in a Network", Networks, No 6,pp 205-229.

STEENBRINK P.A., 1974, *Optimization of Transport Networks*, Ed. John Wiley & Sons.

SYSLO M.M., NARSINGH D., KOWALIK J.S. 1983, Discrete Optimization Algorithms With Pascal Programs, Prentice Hall, Inc. Englewood Cliffs..

TAAFFE E. J., GAUTHIER H.L., 1973, *Geography of Transportation*, Foundation of Economic Geography Series, Prentice Hall, Inc., Englewood Cliffs, N.J.

YEN J.Y., 1971, "Finding the K- shortest Loopless Paths in a network", Management Science, vol.17, No 11, Jul.