Marcelo Mazetto Moala

Uma Implementação do Algoritmo Simplex para Problemas de Transporte Capacitados

UMA IMPLEMENTAÇÃO DO ALGORITMO SIMPLEX PARA PROBLEMAS DE TRANSPORTE CAPACITADOS

Marcelo Mazetto Moala

Dissertação de Mestrado

Pós-Graduação em Matemática Aplicada e Computacional
MAC-010

Uma Implementação do Algoritmo Simplex para Problemas de Transporte Capacitados

Marcelo Mazetto Moala

Orientador Prof. Dr. Geraldo Nunes Silva

Dissertação apresentada ao DCCE/IBILCE/UNESP como parte dos requisitos para obtenção do título de Mestre em Ciências Matemáticas-Área de Concentração Matemática Aplicada e Computacional.

São José do Rio Preto-1996

À Cilene,
meus irmãos,
meus pais e meu
Orientador Geraldo

Agradecimentos

Ao Prof. Dr. **Geraldo Nunes Silva**, pela amizade, pelo incentivo, pela confiança e pela orientação dada a mim durante a realização deste trabalho.

Ao Prof. Dr. **Adalberto Spezamiglio** e ao Prof. Dr. **Eurípides Alves da Silva**, pelas cartas de recomendação que, de alguma maneira, contribuíram para o meu ingresso neste curso.

Ao Prof. Dr. **José Antonio Cordeiro** e ao Prof. Dr. **Manoel Ferreira Borges Neto** pela confiança que depositaram em mim durante todo meu período como mestrando nesta Instituição.

A minha esposa **Cilene**, pelo amor, pelo companheirismo e pelo carinho dedicados a mim desde o meu ingresso na universidade.

Aos meus pais **Waldomiro** e **Jandyra**, pelo amor e pela dedicação que me propiciaram uma vida voltada exclusivamente aos estudos fazendo com que eu possa, do fundo do meu coração, ter orgulho de ter os irmãos que tenho e de ser filho deste casal que amo tanto.

Aos meus irmãos **Fernando**, **Francisco**, **Wladomiro** e ao saudoso **Paulo**, pela compreensão, pelo apoio e pelo incentivo que me deram, contribuindo em muito na minha formação moral e intelectual.

À Da. **Maria**, ao **Luiz** e à saudosa Da. **Cida**, pelo incentivo que depositaram em mim desde que conheci minha esposa.

Aos Professores e Amigos do **DCCE**, e Funcionários da **SPG/IBILCE/UNESP**-São José do Rio Preto-SP, pelo apoio.

Aos Funcionários das bibliotecas do **IBILCE/UNESP**, **ICMSC/USP**, **PUC/RIO**, **UNICAMP** e **UFRGS** pela dedicação e pelo desempenho em adquirirem os artigos solicitados por mim.

À **CAPES** pelo apoio financeiro dado não só a mim como também a todos aqueles que de alguma forma contribuem para o desenvolvimento científico e tecnológico do nosso país.

A todos que direta ou indiretamente contribuíram para a realização deste trabalho.

Resumo

A descrição completa de uma implementação eficiente do Algoritmo Simplex Primal para Problemas de Transporte Capacitados é apresentada. Uma relação explícita entre o Algoritmo Simplex Primal e o Algoritmo de Transporte é estabelecida. Atenção especial é dada às estruturas de dados que não só representam a base, mas que também tornam as operações do Método Simplex mais fáceis de serem realizadas; além de economizar significativamente espaço de memória para armazenagem dos dados. O programa implementado neste trabalho resolve eficientemente Problemas de Transporte esparsos ou densos, com ou sem arcos capacitados, balanceados ou não. O código foi escrito em linguagem de programação Pascal e foi submetido a vários testes resolvendo problemas gerados aleatoriamente.

Palavras Chaves: Problema de Transporte, Otimização, Fluxo em Redes.

Abstract

The complete description of an efficient implementation of the Primal Simplex Algorithm for Capacitated Transportation Problem is presented. An explicit relationship between the Primal Simplex Algorithm and the Transportation Algorithm is established. Special attention is given to data structures which represent the base and make much easier the executions of operations of Simplex Method; besides reducing significantly memory space for data storage. The program implemented at this work efficiently handles sparse or dense Transportation Problems, with capacitated or uncapacitated arcs, balanced or not. The code was written on Pascal programming language and it was tested on a number of randomly generated problems.

Keywords: Transportation Problem, Otimization, Network Flows.

Índice

Introdução

O Problema de Transporte Capacitado consiste em distribuir uma certa quantidade de um produto a partir de *m* pontos de origens para *n* pontos de destinos, onde cada origem *i* tem uma oferta de p_i unidades do produto, cada destino j tem uma demanda de q_j unidades do produto, e o custo de transportar uma unidade do produto da origem *i* para o destino j é de c_{ij} unidades monetárias, com a quantidade y_{ij} do produto a ser enviada da origem *i* para o destino j limitada inferiormente por α_{ij} e superiormente por β_{ij}. Assim, o objetivo de um Problema de Transporte Capacitado é determinar como a distribuição do produto deve ser feita de modo a minimizar os custos.

São inúmeros os problemas que podem ser resolvidos com a aplicação de Problemas de Transporte Capacitados, além do que diversos problemas de Pesquisa Operacional podem ser reformulados como Problemas de Transporte Capacitados. Na prática, os problemas possuem algumas origens (fornecedores, depósitos, etc) e diversos destinos (lojas, consumidores, etc). Para maiores discussões dessas aplicações veja as referências Bradley [10], Busacker e Saaty [12], Charnes e Cooper [13], Dantzig [19], Elmaghraby [22], Ford e Fulkerson [23] e Fulkerson [24].

O modelo de transporte foi originalmente proposto por Hitchcock [29] e por Koopmans [33] cujo método computacional, proposto por George B.Dantzig [1947], é chamado *Simplex Primal*. Hitchcock [29] mostra não só que uma solução ótima se encontra num *ponto extremo*, mas também como obter iterativamente tal solução que, segundo ele, pode ser do tipo *alternativa* e/ou *degenerada*. Já, Koopmans [33], desenvolve os chamados *multiplicadores simplex*, também conhecidos por *nós potenciais*, e descreve o critério de otimalidade mostrando ainda que um ponto extremo é equivalente a uma árvore. Dantzig [18] desenvolve uma variante especial do Algoritmo Simplex Primal para o Problema de Transporte.

Além de algoritmos baseados no Simplex Primal outros algoritmos foram propostos, entre eles citamos o dual (Balas e Hammer [3]), "generic auction" (Bertesekas e Castanon

[7]), "scaling" (Edmonds e Karp [21]), primal-dual (Ford e Fulkerson [23]), "out-of-kilter" (Fulkerson [24]) e ciclo negativo (Klein [31]).

Durante a década de sessenta acreditava-se que implementações de algoritmos para redes baseados no "out-of-kilter" eram melhores (Bradley, Brown e Graves [11]). A implementação de Clasen [14] era a mais conhecida na época. Entretanto, no início da década de setenta aumentou-se o interesse por algoritmos simplex primais para redes e logo ficou evidenciado a superioridade destes algoritimos para redes (Barr, Glover e Klingman [5], Glover, Karney e Klingman [27]).

As implementações recentes do Algoritmo Simplex Primal em rede estão baseadas numa representação compacta da base, determinação do arco que sai da base e técnicas eficientes para atualização dos multiplicadores simplex em cada pivô. Glicksman, Johnson e Eselson [26] descreve um Problema de Transporte com poucas origens e muitos destinos, onde suas estruturas de dados de armazenagem da base são vistas como uma árvore, e utiliza tais estruturas para determinar eficientemente o arco sai da base. Johnson [30] também representa a base como uma árvore e descreve um esquema de rotulação em três índices que permite que os multiplicadores simplex sejam atualizados eficientemente.

Um aspecto importantíssimo atualmente para as pesquisas em redes tem sido os testes computacionais de diferentes algoritmos em problemas testes de grande porte. Um estudo comparativo de algoritmos pode ser visto em Barr, Glover e Klingman [4], Glover, Karney e Klingman [27], onde tais algoritmos são submetidos a um conjunto variado de problemas testes gerados em Barr, Glover e Klingman [4].

O objetivo deste trabalho é descrever um pacote computacional, *TRANSMAX*, que foi escrito em linguagem de programação Pascal que resolve eficientemente Problemas de Transporte Capacitados. Uma das vantagens desse pacote é sua aplicação no Brasil em microcomputadores PC's com pouca memória RAM.

O programa *TRANSMAX* é uma implementação do Algoritmo Simplex Primal para o Problema de Transporte Capacitado baseado nos estudos desenvolvidos, principalmente por Bradley, Brown e Graves [11], onde as bases são representadas por árvores geradoras

fortemente viáveis utilizando-se vetores que armazenam, para cada nó, o nível, o antecessor, o último sucessor entre outras estruturas de dados que podem ser vistas ao longo deste trabalho. A finalidade é determinar tanto o arco que entra na árvore como também o arco que sai de forma eficiente, além de facilitar a atualização dos multiplicadores simplex.

O *TRANSMAX* foi testado em Problemas de Transporte Capacitados ou não, esparsos ou densos, balanceados ou não, os quais foram gerados aleatoriamente.

No capítulo 1 é apresentada uma classe de problemas chamada *Problemas de Programação Linear com Variáveis Limitadas*. Neste tipo de problema temos um número k de restrições de capacidade e um número m de outras restrições, que para ser resolvido, bastaria considerar todas as restrições como um único conjunto de restrições. Mas, tal procedimento aumentaria o tamanho do problema inviabilizando sua solução em tempo razoável.

Assim, para contornar esta situação, Dantzig [19] desenvolveu uma técnica especial que torna possível resolver o problema em questão utilizando-se somente bases de ordem m. O método é uma generalização do Método Simplex chamado *Método Simplex com Variáveis Limitadas*. O algoritmo move-se de uma solução básica viável para uma solução básica viável melhor até que a otimalidade seja atingida ou se verifica que o valor da função objetivo é ilimitado.

Neste capítulo também discuti-se as dificuldades associadas com degenerescências, uma vez que o Método Simplex com Variáveis Limitadas converge num número finito de passos na ausência de degenerescência. A regra adotada para prevenir ciclagem no Método Simplex com Variáveis Limitadas é a regra da Lexicografia a qual garante que nenhuma das bases visitadas por este método sejam repetidas. Em vista do número finito de bases, isto garante automaticamente o término do algoritmo em um número finito de iterações. Uma breve discussão sobre o fenômeno "stalling" é também apresentada.

O capítulo 2 descreve uma especialização do Método Simplex Primal com Variáveis Limitadas para os Problemas de Transporte Capacitados ou de Hitchcock. Esta classe especial de Problemas de Programação Linear tem sua estrutura dada através de redes, e

portanto, para um melhor entendimento, mostramos alguns conceitos da teoria dos grafos. Mostramos ainda como obtermos uma solução básica viável inicial para um problema desta natureza, assim como uma solução básica viável numa iteração qualquer pode ser representada por uma árvore geradora enraizada. Tal fato contribui, e muito, para que possamos perceber que os passos deste algoritmo, como por exemplo, a determinação dos multiplicadores simplex ou o movimento de uma árvore geradora enraizada para outra, são especializações dos passos usuais do Método Simplex com Variáveis Limitadas.

Uma vez que qualquer rede tem um número finito de arvores geradoras, e que cada árvore geradora corresponde a uma solução básica viável, o algoritmo deveria terminar em número finito de iterações. Mas, podem ocorrer pivôs degenerados, fazendo com que o algoritmo entre em um processo de ciclagem. A convergência finita do algoritmo está garantida quando se tem árvores geradoras fortemente viáveis em cada iteração.

Mesmo com a garantia da convergência finita do algoritmo, um outro fenômeno chamado "stalling" pode ocorrer, fazendo com que a convergência do algoritmo seja lenta. Este fenômeno também é discutido.

O capítulo 3 descreve a implementação de *TRANSMAX*, para a otimização de redes de transporte utilizando-se do fato das bases dos Problemas de Transporte serem representadas por árvores geradoras enraizadas fortemente viáveis. O programa resolve problemas esparsos ou densos, com ou sem arcos capacitados, e gera automaticamente nós e arcos artificiais no caso de problemas não-balanceados.

No Capítulo 4 o desempenho de *TRANSMAX* é apresentado nas tabelas contendo os resultados obtidos após submetê-lo a vários testes gerados aleatoriamente. Discute-se os resultados obtidos e as direções de trabalho futuro.

CAPÍTULO 1

MÉTODO SIMPLEX COM VARIÁVEIS LIMITADAS

1.1 INTRODUÇÃO

Neste capítulo, apresentaremos uma classe de Problemas de Programação Linear cujas variáveis são limitadas. Problemas como estes são chamados *Problemas de Programação Linear com Variáveis Limitadas*, e para resolvê-los, descreveremos uma generalização do Método Simplex chamada *Método Simplex com Variáveis Limitadas*, o qual é padrão na literatura podendo ser encontrado em diversos livros-texto, como por exemplo, nas referências Ahuja, Magnanti, Orlin [1], Bazaraa, Jarvis, Sherali [6], Murty [39].

O algoritmo apresentado aqui, move-se de uma solução básica viável para uma solução básica viável melhor até que a otimalidade seja atingida ou se verifica que a solução é ilimitada. Trataremos somente de problemas de minimização uma vez que para os casos de maximização basta multiplicarmos os coeficientes da função objetivo por (-1), e após completada a otimização, o valor da função objetivo é (-1) vezes o valor do novo problema.

O Problema de Programação Linear com Variáveis Limitadas pode ser formulado da seguinte forma:

$$\begin{aligned} &\text{Minimizar} && cy \\ &\text{sujeito a:} && Ay = d \\ & && l \le y \le s \end{aligned} \qquad (1.2.1)$$

onde A é uma matriz de ordem $m \times n$ de posto m, $c \in R^n$ é o vetor de custos, $y \in R^n$ é a variável de decisão, e l e s , com $0 \le l < s$, são os limitantes inferiores e superiores, respectivamente, da variável de decisão y. Note aqui que, as relações de desigualdades entre

vetores têm significado similar quando no caso com números, ou seja, dados dois vetores $x, y \in R^n$, temos $x \geq y$ se, e somente se, $x_i \geq y_i, \forall i \in \{1,2,...,n\}$.

Se, no problema (1.2.1), substituirmos a restrição $l \leq y \leq s$ por simplesmente $y \geq 0$, teremos um Problema de Programação Linear usual, com as restrições de não-negatividade e sem o limitante superior.

A discussão do Método Simplex com Variáveis Limitadas será feita para o caso em que o limitante inferior $l \in R^n$ na restrição de desigualdade do problema (1.2.1) é o vetor nulo. Para isto, basta fazer a seguinte mudança de variável:

$$x = y - l$$

Desta maneira, o problema reformulado torna-se:

$$\begin{aligned} &\text{Minimizar} && z(x) = cx \\ &\text{sujeito a:} && Ax = b \\ & && 0 \leq x \leq u \end{aligned} \qquad (1.2.2)$$

onde $b = d - Al$, $u = s - l$.

Observe que a constante cl foi eliminada da função objetivo, e que, se pode supor ainda que $b \geq 0$. Nestas condições, dizemos que o problema (1.2.2) está na *forma padrão*.

1.2 SOLUÇÃO BÁSICA VIÁVEL

Considere o problema (1.2.2). Particione a matriz A em $[B, L, S]$ e o vetor x em (x_B, x_L, x_S), onde as variáveis em x_B correspondentes às colunas da matriz B são chamadas *variáveis básicas*, e as variáveis em x_L e x_S correspondentes às colunas das matrizes L e S são chamadas *variáveis não-básicas* em suas capacidades inferiores e superiores, respectivamente. Nestas condições, a matriz B é chamada *matriz básica de trabalho* e a solução $x = (x_B, x_L, x_S)$ é chamada *solução básica*. Todas as variáveis $x_i \in x_L$ têm seus valores iguais as suas capacidades inferiores, zero, na solução básica, enquanto que todas as variáveis $x_i \in x_S$ têm seus valores iguais as suas capacidades superiores, u_i, na solução básica.

Após substituir as variáveis não-básicas pelos respectivos valores em x_L e x_S, os valores das variáveis em x_B são obtidos de maneira única através do sistema $Ax = b$, e a solução básica do problema (1.2.2) correspondente à partição $[B, L, S]$ é $\bar{x}$, onde:

$$\bar{x}_j = \begin{cases} 0 & \text{para } x_j \in x_L \\ u_j & \text{para } x_j \in x_S \\ B^{-1}(b - \sum_{x_i \in x_S} a_i u_i) & \text{para } x_j \in x_B \end{cases}$$

onde a_i é a i-ésima coluna da matriz S.

Se $0 \leq \bar{x}_j \leq u_B$ para $x_j \in x_B$, onde u_B representa o vetor das capacidades superiores das variáveis básicas, então $\bar{x}$ é chamada *solução básica viável* e a partição $[B, L, S]$ é chamada *partição básica viável*. Se $0 < \bar{x}_j < u_B$ para $x_j \in x_B$, então $\bar{x}$ é chamada *solução básica viável não-degenerada* e a partição $[B, L, S]$ é chamada *partição básica viável não-degenerada*. Finalmente, se $\bar{x}_j = 0$ ou $\bar{x}_j = u_j$ para pelo menos uma variável $x_j \in x_B$, então $\bar{x}$ é chamada *solução básica viável degenerada* e a partição $[B, L, S]$ é chamada *partição básica viável degenerada*.

1.3 CRITÉRIO DE OTIMALIDADE

Suponha que a solução básica viável atual seja $\bar{x}$, onde as variáveis não-básicas em $\bar{x}_L$ e $\bar{x}_S$ tomam valores nas suas capacidades inferiores e superiores, respectivamente. Ao decompormos o vetor x em (x_B, x_L, x_S), conseqüêntemente decompomos o vetor de custos c em (c_B, c_L, c_S).

Após algumas operações algébricas, as variáveis básicas e a função objetivo podem ser escritas em termos das variáveis não-básicas em $\bar{x}_L$ e $\bar{x}_S$ como segue:

$$A\bar{x} = b$$

$$[B, L, S](\bar{x}_B, \bar{x}_L, \bar{x}_S) = b$$

$$B\bar{x}_B + L\bar{x}_L + S\bar{x}_S = b$$

$$\bar{x}_B = B^{-1}b - B^{-1}L\bar{x}_L - B^{-1}S\bar{x}_S \qquad (1.3.1)$$

$$z = c\bar{x}$$

$$z = (c_B, c_L, c_S)(\bar{x}_B, \bar{x}_L, \bar{x}_S)$$

$$z = c_B\bar{x}_B + c_L\bar{x}_L + c_S\bar{x}_S$$

$$z = c_B(B^{-1}b - B^{-1}L\bar{x}_L - B^{-1}S\bar{x}_S) + c_L\bar{x}_L + c_S\bar{x}_S$$

$$z = c_B B^{-1}b + \sum_{j\in I_L}(c_j - z_j)\bar{x}_j + \sum_{j\in I_S}(c_j - z_j)\bar{x}_j$$

onde $z_j = c_B B^{-1}a_j$, I_L e I_S são os conjuntos de índices das variáveis não-básicas em suas capacidades inferiores e superiores, respectivamente.

Agora, fazemos:

$$\psi B - c_B = 0 \quad \text{ou} \quad \psi B = c_B$$

onde referimos a $\psi = c_B B^{-1}$ como vetor dos *multiplicadores simplex* associados com a base B, e referimos a $\pi_j = c_j - z_j$ como o *coeficiente de custo reduzido* da variável x_j. Logo, a função objetivo é reescrita em termos das variáveis não-básicas da seguinte maneira:

$$z = c_B B^{-1}b + \sum_{j\in I_L}\pi_j\bar{x}_j + \sum_{j\in I_S}\pi_j\bar{x}_j \qquad (1.3.2)$$

O valor de uma única variável não-básica é modificado, enquanto que o valor das demais variáveis não-básicas ficam fixos. O índice w desta variável não-básica que tem seu valor modificado está associado ao seguinte número:

$$\pi_w = \text{Máximo}\left\{\underset{j\in I_L}{\text{Máximo}}\{\pi_j\}, \underset{j\in I_S}{\text{Máximo}}\{-\pi_j\}\right\}$$

Se $\pi_w < 0$, então $\pi_j < 0$ para algum $j \in I_L$ e $\pi_j > 0$ para algum $j \in I_S$, e portanto, (1.3.2) sugere que x_w, para $w \in I_L$, deve ser aumentado de seu valor atual 0, e x_w, para

$w \in I_S$, deve ser diminuído de seu valor atual u_w. Entretanto, se $\pi_w \geq 0$, então $\pi_j \geq 0$ para todo $j \in I_L$, e $\pi_j \leq 0$ para todo $j \in I_S$, o que nos leva a concluir novamente de (1.3.2) que a solução atual é ótima.

Logo, os critérios de otimalidade são dados por:

$$\pi_j \geq 0, \text{ para } j \in I_L$$

$$\pi_j \leq 0, \text{ para } j \in I_S$$

Portanto, se a solução básica viável satisfaz estes critérios de otimalidade, a função objetivo atinge seu limite inferior $z = c_B B^{-1} b + \sum \pi_j \bar{x}_j$ com $j \in I_S$ (note que $\bar{x}_j = 0$ para $j \in I_L$), e assim, a solução básica viável atual é ótima.

Reservamos o final desta seção para fazermos uma breve, mas importante, discussão sobre as várias maneiras com que a variável que entra na base pode ser selecionada.

A regra utilizada anteriormente para selecionar a variável que entra na base é chamada *Regra de Dantzig* e foi desenvolvida por George B. Dantzig em 1947. Resultados computacionais indicam que esta maneira de introduzir uma nova variável na base, tende a produzir um decrescimento relativamente grande no valor da função objetivo de uma iteração para outra. Entretanto, a desvantagem desta prática é o excesso de tempo consumido na identificação da nova variável que entrará na base, uma vez que devemos percorrer todas as variáveis não-básicas. Existem muitas outras regras para selecionar a variável que entra na base. Veja a seguir algumas dessas regras que também podem ser encontradas em Murty [39].

Uma outra regra também utilizada com o objetivo de selecionar a variável que entra na base é a *Regra da Primeira Variável Candidata*. Dispondo todas as variáveis numa seqüência qualquer, esta regra consiste em selecionar a primeira variável não-básica desta seqüência que contrariar o critério de otimaliadade. Se a *k*-ésima variável da seqüência foi escolhida a entrar na base numa iteração *i*, então, na iteração *i*+1 a pesquisa em busca da candidata a entrar na base é realizada à partir da $(k+1)$-ésima variável da seqüência. Atingido o final desta seqüência, a procura de variáveis candidatas a entrar na base é

realizada novamente partindo-se do início da seqüência. A vantagem desta regra é a rápida identificação da nova variável básica. Todavia, o algoritmo efetua mais iterações com esta regra do que com outras devido ao decrescimento relativamente pequeno no valor da função objetivo de uma iteração para outra.

Finalmente, uma outra regra que tem por objetivo selecionar a variável que entra na base é a chamada *Regra da Lista de Candidatas*. Esta regra consiste em utilizar um procedimento dotado de duas fases. A primeira fase contém um número maior de iterações e tem como finalidade construir uma lista composta de variáveis candidatas a entrar na base. Um número menor de iterações é realizado na segunda fase onde cada uma destas iterações tem o compromisso de determinar a variável que realmente entrará na base dentre todas as variáveis na lista construída na fase anterior. A lista é atualizada removendo-se aquelas variáveis que não são mais candidatas. Esgotada a lista ou atingido um número específico de iterações na segunda fase, reconstruímos a lista de variáveis candidatas reiniciando a primeira fase contendo um número de iterações maior. Note que a Regra de Dantzig e a Regra da Primeira Candidata são casos particulares dessa última regra.

O Capítulo 3 mostra a implementação da Regra da Lista de Candidatas para o Problema de Transporte Capacitado, onde sua eficiência pode ser notada com relação ao número de iterações efetuadas para se obter a solução.

1.4 ALGORITMO

O Método Simplex seleciona uma variável não-básica x_w para entrar na base, e tenta aumentar ou diminuir seu valor o máximo posssível de modo a fazer com que alguma variável básica $x_{w'}$ atinja sua capacidade inferior ou superior, enquanto as demais variáveis não-básicas são mantidas em suas capacidades inferiores ou superiores. Em seguida, o Método Simplex substitui a variável básica $x_{w'}$ pela variável não-básica x_w, definindo uma nova base. Então, o método atualiza as demais variáveis básicas através da matriz inversa da base e repete os cálculos novamente.

Critério para Entrada

O Método Simplex com Variáveis Limitadas considera qualquer variável não-básica x_w como candidato a entrar na base se $\pi_w < 0$ para $w \in I_L$, e $\pi_w > 0$ para $w \in I_S$.

Critério de Saída

Uma vez que x_w é selecionada como a variável que entra na base, o Método Simplex com Variáveis Limitadas tenta aumentar o seu valor o máximo possível quando esta estiver na sua capacidade inferior, e tenta diminuir o máximo possível o seu valor quando esta estiver na sua capacidade superior, sempre respeitando os valores das variáveis básicas e o valor de x_w que devem permanecer dentro dos seus limites de capacidades.

Primeiramente, suponhamos que x_w esteja em sua capacidade inferior.

Substituindo as variáveis não-básicas em x_L e x_S pelas suas respectivas capacidades inferiores (zero) e superiores, e denotando por $\bar{b}$ o valor atual das variáveis em x_B de (1.3.1), obtemos o seguinte resultado:

$$\bar{b} = B^{-1}b - B^{-1}Su_S \tag{1.4.1}$$

onde u_S é o vetor das capacidades superiores para as variáveis não-básicas que estão em suas capacidades superiores.

Se aumentarmos o valor da variável que entra x_w para $\Delta_w \geq 0$ e mantivermos todas as outras variáveis não-básicas em suas capacidades inferiores e superiores, então, de acordo com (1.3.1) e (1.4.1), as variáveis básicas em x_B são dadas por:

$$\bar{x}_B = B^{-1}b - B^{-1}Su_S - B^{-1}a_w\Delta_w$$
$$\bar{x}_B = \bar{b} - y_w\Delta_w$$

onde $y_w = B^{-1}a_w$ e a_w é a coluna da matriz L correspondente à variável não-básica x_w.

Mais explicitamente, temos:

$$\bar{x}_{B_i} = \bar{b}_i - y_{iw}\Delta_w, \text{ para } i \in I_B \tag{1.4.2}$$

onde I_B é o conjunto dos índices das variáveis básicas.

Denotando por $\bar{z}$ o valor da função objetivo de (1.3.2) antes do aumento da variável não-básica x_w, e mantendo as demais variáveis não-básicas em suas capacidades inferiores e superiores, obtemos a seguinte relação:

$$z = c_B B^{-1} b + \sum_{j \in I_S} \pi_j \bar{x}_j + \pi_w \Delta_w$$

$$z = \bar{z} + \pi_w \Delta_w \qquad (1.4.3)$$

Uma vez que $\pi_w < 0$, então aumentando o máximo possível o valor de Δ_w, (1.4.3) implica que o valor da função objetivo é decrescido.

Nestas condições, teremos três casos a serem analisados.

Caso 1: *Uma variável básica é reduzida à sua capacidade inferior*

Seja x_{B_r} a variável básica que é reduzida à sua capacidade inferior. Denotemos por Ω_1 o valor de Δ_w que faz com que a variável básica x_{B_r} seja reduzida à sua capacidade inferior quando a variável x_w é aumentada da sua capacidade inferior.

De (1.4.2) temos $0 \le \bar{x}_{B_r} = \bar{b}_r - y_{rw}\Delta_w$, implicando que, se $y_{rw} \le 0$, então poderemos atribuir um valor arbitrariamente grande a Δ_w sem que o valor da variável básica x_{B_r} seja menor do que a sua capacidade inferior, ou seja, teremos $\Omega_1 = \infty$, e neste caso, nenhuma variável básica será reduzida à sua capacidade inferior. Caso contrário, Ω_1 é dado pelo seguinte teste da razão mínima:

$$\underset{1 \le i \le m}{\text{Mínimo}} \left\{ \frac{\bar{b}_i}{y_{iw}} : y_{iw} > 0 \right\} = \frac{\bar{b}_r}{y_{rw}}$$

onde o índice *i* indica a variável x_{B_i} que sai da base.

Logo, da discussão acima concluímos que:

$$\Omega_1 = \begin{cases} \text{Mínimo}_{1 \le i \le m} \left\{ \dfrac{\bar{b}_i}{y_{iw}} : y_{iw} > 0 \right\}, & \text{se } y_w \not\le 0 \\ \infty, & \text{se } y_w \le 0 \end{cases}$$

e portanto, uma variável básica que for reduzida à sua capacidade inferior será uma candidata a sair da base.

Caso 2: *Uma variável básica atinge a sua capacidade superior*

Seja x_{B_r} a variável básica que atinge a sua capacidade superior. Denotemos por Ω_2 o valor de Δ_w que faz com que a variável básica x_{B_r} atinja a sua capacidade superior quando a variável x_w é aumentada da sua capacidade inferior.

De (1.4.2) temos $\bar{x}_{B_r} = \bar{b}_r - y_{rw}\Delta_w \le u_{B_r}$, implicando que, se $y_{rw} \ge 0$, então poderemos atribuir um valor arbitrariamente grande a Δ_w sem que o valor da variável básica x_{B_r} seja maior do que a sua capacidade superior, ou seja, teremos $\Omega_2 = \infty$, e neste caso, nenhuma variável básica atingirá sua capacidade superior. Caso contrário, Ω_2 é dado pelo seguinte teste da razão mínima:

$$\text{Mínimo}_{1 \le i \le m} \left\{ \frac{\bar{b}_i - u_{B_i}}{y_{iw}} : y_{iw} < 0 \right\} = \frac{\bar{b}_r - u_{B_r}}{y_{rw}}$$

onde o índice *i* indica a variável x_{B_r} que sai da base.

A discussão acima acarreta a seguinte conclusão:

$$\Omega_2 = \begin{cases} \text{Mínimo}_{1 \le i \le m} \left\{ \dfrac{\bar{b}_i - u_{B_i}}{y_{iw}} : y_{iw} < 0 \right\}, & \text{se } y_w \not\ge 0 \\ \infty, & \text{se } y_w \ge 0 \end{cases}$$

e portanto, uma variável básica que atingir a sua capacidade superior será uma candidata a sair da base.

Caso 3: *A própria variável x_w atinge a sua capacidade superior*

Neste caso, a análise a ser feita consiste simplesmente em notar que o valor de Δ_w no qual a variável não-básica x_w atinge a sua capacidade superior é u_w.

Portanto, destes três casos concluímos que Δ_w é dado por:

$$\Delta_w = \text{Mínimo}\{\Omega_1, \Omega_2, u_w\}$$

Se $\Delta_w = \infty$, então o aumento em x_w é arbitrariamente grande, e de acordo com (1.4.3), vemos que o valor da função objetivo é ilimitado. No entanto, se $\Delta_w < \infty$, então uma nova solução básica viável é obtida onde $x_w = \Delta_w$ e as variáveis básicas são atualizadas de acordo com a relação $\bar{x}_B = \bar{b} - y_w \Delta_w$.

Agora, suponhamos que x_w esteja em sua capacidade superior.

Se diminuirmos o valor da variável que entra x_w para $\Delta_w \geq 0$ e mantivermos todas as outras variáveis não-básicas em suas capacidades inferiores e superiores, então teremos $x_w = u_w - \Delta_w$, e de acordo com (1.3.1) e (1.4.1), as variáveis básicas em x_B mudam na seguinte maneira:

$$\bar{x}_B = B^{-1}b - B^{-1}Su_S + B^{-1}a_w\Delta_w$$

$$\bar{x}_B = \bar{b} + y_w\Delta_w$$

onde $y_w = B^{-1}a_w$ e a_w é a coluna da matriz L correspondente à variável não-básica x_w.

Mais explicitamente, temos:

$$\bar{x}_{B_i} = \bar{b}_i + y_{iw}\Delta_w, \text{ para } i \in I_B \qquad (1.4.4)$$

onde I_B é o conjunto dos índices das variáveis básicas.

Denotando por $\bar{z}$ o valor da função objetivo dada em (1.3.2) antes do decrescimento da variável não-básica x_w, e mantendo as demais variáveis não-básicas em suas capacidades inferiores e superiores, obtemos a seguinte relação:

$$z = c_B B^{-1} b + \sum_{j \in I_S} \pi_j \bar{x}_j - \pi_w \Delta_w$$

$$z = \bar{z} - \pi_w \Delta_w \qquad (1.4.5)$$

Uma vez que $\pi_w > 0$, então aumentando o máximo possível o valor de Δ_w em (1.4.5) implica que o valor da função objetivo é decrescido.

Novamente, teremos três subcasos a serem analisados.

Caso 1: *Uma variável básica é reduzida à sua capacidade inferior*

Seja x_{B_r} a variável básica que é reduzida à sua capacidade inferior. Denotemos por Ω_1 o valor de Δ_w que faz com que a variável básica x_{B_r} seja reduzida à sua capacidade inferior quando a variável x_w é decrescida da sua capacidade superior.

De (1.4.4) temos $0 \le \bar{x}_{B_r} = \bar{b}_r + y_{rw} \Delta_w$, implicando que, se $y_{rw} \ge 0$, então poderemos atribuir um valor arbitrariamente grande a Δ_w sem que o valor da variável básica x_{B_r} seja menor do que a sua capacidade inferior, ou seja, teremos $\Omega_1 = \infty$, e neste caso, nenhuma variável básica será reduzida à sua capacidade inferior. Caso contrário, Ω_1 é dado pelo seguinte teste da razão mínima:

$$\underset{1 \le i \le m}{\text{Mínimo}} \left\{ \frac{\bar{b}_i}{-y_{iw}} : y_{iw} < 0 \right\} = \frac{\bar{b}_r}{-y_{rw}}$$

Da discussão acima concluímos que:

$$\Omega_1 = \begin{cases} \underset{1 \le i \le m}{\text{Mínimo}} \left\{ \dfrac{\bar{b}_i}{-y_{iw}} : y_{iw} < 0 \right\}, & \text{se } y_w \not\geq 0 \\ \infty, & \text{se } y_w \ge 0 \end{cases}$$

e portanto, uma variável básica que for reduzida à sua capacidade inferior será uma candidata a sair da base.

Caso 2: *Uma variável básica atinge a sua capacidade superior*

Seja x_{B_r} a variável básica que atinge a sua capacidade superior. Denotemos por Ω_2 o valor de Δ_w que faz com que a variável básica x_{B_r} atinja a sua capacidade superior quando a variável x_w é decrescida da sua capacidade superior.

De (1.4.4) temos $\bar{x}_{B_r} = \bar{b}_r + y_{rw}\Delta_w \leq u_{B_r}$, implicando que, se $y_{rw} \leq 0$, então poderemos atribuir um valor arbitrariamente grande a Δ_w sem que o valor da variável básica x_{B_r} seja maior do que a sua capacidade superior, ou seja, teremos $\Omega_2 = \infty$, e neste caso, nenhuma variável básica atingirá sua capacidade superior. Caso contrário, Ω_2 é dado pelo seguinte teste da razão mínima:

$$\underset{1 \leq i \leq m}{\text{Mínimo}}\left\{\frac{u_{B_i} - \bar{b}_i}{y_{iw}} : y_{iw} > 0\right\} = \frac{u_{B_r} - \bar{b}_r}{y_{rw}}$$

onde o índice *i* indica a variável x_{B_r} que sai da base.

A discussão acima acarreta a seguinte conclusão:

$$\Omega_2 = \begin{cases} \underset{1 \leq i \leq m}{\text{Mínimo}}\left\{\dfrac{u_{B_i} - \bar{b}_i}{y_{iw}} : y_{iw} > 0\right\}, & \text{se } y_w \not\leq 0 \\ \infty, & \text{se } y_w \leq 0 \end{cases}$$

Caso 3: *A própria variável x_w atinge a sua capacidade inferior*

Outra vez, a análise a ser feita consiste simplesmente em notar que o valor de Δ_w no qual a variável não-básica x_w atinge a sua capacidade inferior é u_w.

Portanto, destes três casos também concluímos que Δ_w é dado por:

$$\Delta_w = \text{Mínimo}\{\Omega_1, \Omega_2, u_w\}$$

Se $\Delta_w = \infty$, então o decrescimento de x_w é arbitrariamente grande, e de acordo com (1.4.5), vemos que o valor da função objetivo é ilimitado. No entanto, se $\Delta_w < \infty$, então uma nova solução básica viável é obtida onde $x_w = u_w - \Delta_w$ e as variáveis básicas são atualizadas de acordo com a relação $\bar{x}_B = \bar{b} + y_w\Delta_w$.

Resumidamente, o algoritmo do Método Simplex para Variáveis Limitadas segue os seguintes passos:

Passo 0: Encontre uma solução básica viável inicial;

Passo 1: Se $z_j - c_j \geq 0$ para as variáveis não-básicas nas suas capacidades inferiores e $z_j - c_j \leq 0$ para as variáveis não-básicas nas suas capacidades superiores, então a solução atual é ótima. Caso contrário, se uma destas condições for violada para o índice *w*, então vá para o Passo 2 se x_w está em sua capacidade inferior e para o Passo 3 se x_k estiver em sua capacidade superior.

Passo 2: A variável x_w é aumentada de seu valor atual de 0 para Δ_w.

O valor de Δ_w é dado por:

$$\Delta_w = \text{Mínimo}\{\Omega_1, \Omega_2, u_w\}$$

onde:

$$\Omega_1 = \begin{cases} \underset{1 \leq i \leq m}{\text{Mínimo}} \left\{ \dfrac{\bar{b}_i}{y_{iw}} : y_{iw} > 0 \right\}, & \text{se } y_w \not\leq 0 \\ \infty, & \text{se } y_w \leq 0 \end{cases}$$

$$\Omega_2 = \begin{cases} \underset{1 \leq i \leq m}{\text{Mínimo}} \left\{ \dfrac{\bar{b}_i - u_{B_i}}{y_{iw}} : y_{iw} < 0 \right\}, & \text{se } y_w \not\geq 0 \\ \infty, & \text{se } y_w \geq 0 \end{cases}$$

Se $\Delta_w = \infty$, então pare; a solução é ilimitada. Caso contrário, atualize a nova solução básica viável de acordo com a relação $\bar{x}_B = \bar{b} - y_w \Delta_w$. Repita o Passo 1.

Passo 3: A variável x_w é diminuída de seu valor atual de u_w para $u_w - \Delta_w$.

O valor de Δ_w é dado por:

$$\Delta_w = \text{Mínimo}\{\Omega_1, \Omega_2, u_w\}$$

onde:

$$\Omega_1 = \begin{cases} \text{Mínimo}\left\{ \dfrac{\bar{b}_i}{-y_{iw}} : y_{iw} < 0 \right\}, & \text{se } y_w \not\leq 0 \\ \infty, & \text{se } y_w \geq 0 \end{cases}$$

$$\Omega_2 = \begin{cases} \text{Mínimo}\left\{ \dfrac{u_{B_i} - \bar{b}_i}{y_{iw}} : y_{iw} > 0 \right\}, & \text{se } y_w \not\leq 0 \\ \infty, & \text{se } y_w \leq 0 \end{cases}$$

Se $\Delta_w = \infty$, então pare; a solução é ilimitada. Caso contrário, atualize a nova solução básica viável de acordo com a relação $\bar{x}_B = \bar{b} + y_w \Delta_w$. Repita o Passo 1.

Se nenhuma solução básica viável estiver disponível, podemos iniciar o Método Simplex com Variáveis Limitadas introduzindo-se *variáveis artificiais*, onde tais variáveis artificiais podem ser compreendidas do modo como segue. Suponha que o sistema $Ax = b$, $x \geq 0$, seja alterado pela introdução de um vetor x_a de variáveis artificiais de modo a obtermos o sistema $Ax + x_a = b$, $x \geq 0$, $x_a \geq 0$. Logo, uma solução básica viável inicial pode ser obtida deste novo sistema da seguinte maneira:

(*i*) atribua a cada variável original x uma de suas capacidades inferior ou superior;

(*ii*) se $b_j \leq 0$, então multiplique a j-ésima linha do novo sistema por (-1), de modo a obter $b_j \geq 0$;

(*iii*) determine o valor da variável artificial x_a de acordo com o valor de x dado em (*i*).

Agora, podemos aplicar o Método das Duas Fases, ou o Método do M-Grande, para retirar as variáveis artificiais da base.

1.5 CONVERGÊNCIA

Antes de discutirmos a convergência do Método Simplex com Variáveis Limitadas, vejamos algumas definições preliminares.

Chamamos de *operação pivô* ou *passo pivô* do Método Simplex sobre um sistema de equações a operação que consiste em determinar a variável que sai e a variável que entra na base. O *elemento pivô* é dado pelo elemento y_{iw} do teste da razão mínima (veja seção 1.4), de tal modo que $i \equiv r$, onde r é o índice correspondente à variável que sai da base. O elemento pivô deve ser não-nulo pois a operação requer operações com o elemento pivô como denominador.

Um passo pivô é chamado *passo pivô não-degenerado* se o valor da variável que entra na base altera a solução neste passo, caso contrário, o passo é chamado *passo pivô degenerado*. Durante cada passo pivô não-degenerado do Algoritmo Simplex, a solução muda e o valor da função objetivo sofre um decrescimento estrito. No entanto, durante um passo pivô degenerado, não existe mudança na solução nem no valor da função objetivo, mas somente a base de trabalho B e a partição da matriz A em $[B,L,S]$ mudam.

A obtenção de um passo pivô degenerado pode fazer com que o algoritmo entre num processo chamado *ciclagem*, ou seja, num processo que se repete infinitamente fazendo com que o algoritmo não possa ser finalizado.

Na ausência de degenerescência, o Algoritmo Simplex com Variáveis Limitadas converge finitamente, e portanto o valor da função objetivo decresce estritamente em cada passo fazendo com que sua aplicação em qualquer Problema de Programação Linear tenha um número finito de iterações.

A ciclagem pode ser evitada aplicando-se, por exemplo, uma regra chamada *Regra Lexicográfica* que pode ser encontrada em Bazaraa, Jarvis, Sherali [6]. Esta regra consiste em particionar a matriz A em $[B,L,S]$ em cada passo, onde as linhas da matriz B^{-1} correspondentes às variáveis básicas em suas capacidades inferiores são vetores lexicográficos positivos e as linhas da matriz B^{-1} correspondentes às variáveis básicas em suas capacidades superiores são vetores lexicográficos negativos. Entende-se por vetor *lexicográfico positivo* (*negativo*) todo vetor x não-nulo cuja primeira componente não-nula é positiva (negativa). Neste caso, dizemos também que a partição $[B,L,S]$ é uma *partição básica fortemente viável*.

Uma outra regra que previne ciclagem é sugerida por Bland [8], chamada *Regra de Bland*. Tal regra consiste primeiramente em ordenar todas as variáveis em alguma seqüência. Então, entrará na base a variável não-básica com o menor índice na seqüência ordenada dentre todas as variáveis não-básica candidatas a entrar na base. Similarmente, sairá da base a variável básica com o menor índice na seqüência ordenada dentre todas as variáveis básicas candidatas a sair na base.

Embora as regras mencionadas acima resolvem ciclagem, elas não evitam um outro fenômeno chamado "*stalling*". O fenômeno "stalling" é uma seqüência finita, mas muito longa, de pivôs degenerados consecutivos, e portanto, tal ocorrência pode fazer com que o algoritmo consuma uma quantidade de tempo muito grande, embora finita, em um vértice degenerado antes mesmo de obtermos a otimalidade ou movermos daquele vértice.

Cunningham [17] trata de problemas em redes não capacitadas e apresenta um exemplo no qual a regra de Bland não evita o fenômeno "stalling". Além disso, descreve três estratégias de selecionar a variável que entra na base para evitar o fenômeno "stalling". Uma delas, referida algumas vezes como a *Regra LRC* ("*Least Recently Considered*") para escolha da variável que entra na base, embora descrita no contexto de redes, é aplicável no caso geral do simplex primal para o problema de programação linear. Esta regra consiste em arranjar todas as variáveis numa ordem fixa qualquer e selecionar como variável a entrar na base a primeira variável deste arranjo que for candidata a entrar na base. Por exemplo, suponha que no início do algoritmo tivéssemos o seguinte arranjo $x_1, x_2, \ldots, x_m$. Então, num passo pivô qualquer, supondo que x_k foi a variável que entrou na base no passo anterior, será escolhida para entrar na base a primeira variável candidata da lista $x_{k+1}, x_{k+2}, \ldots, x_m$, $x_1, \ldots, x_k$.

As outras estratégias para selecionar a variável que entra na base no sentido de evitar o fenômeno "stalling" apresentadas por Cunningham são específicas para redes e serão discutidas no próximo capítulo.

É também conhecido que, aplicando-se a regra lexicográfica em conjunto com a regra de Dantzig a problemas estruturados em rede evita o fenômeno "stalling". Porém para problemas de programação linear geral permanece uma questão em aberto.

CAPÍTULO 2

PROBLEMA DE TRANSPORTE CAPACITADO

2.1 INTRODUÇÃO

Este capítulo tem por objetivo fornecer um breve estudo sobre Problemas de Transporte Capacitados, aplicando-se o Método Simplex com Variáveis Limitadas apresentado no Capítulo 1. Os detalhes podem ser obtidos nas referências Ahuja, Magnanti, Orlin [1], Bazaraa, Jarvis, Sherali [6], Murty [38], e Resende [41].

O Problema de Transporte Capacitado consiste em distribuir uma quantidade de um produto a partir de m pontos de origens para n pontos de destinos, onde cada origem i tem uma oferta de p_i unidades do produto, cada destino j tem uma demanda de q_j unidades do produto, e o custo de transportar uma unidade do produto da origem i para o destino j é de c_{ij} unidades monetárias, com a quantidade y_{ij} do produto, chamada *fluxo*, a ser enviada da origem i para o destino j limitada inferiormente por α_{ij} e superiormente por β_{ij}. O objetivo de um Problema de Transporte Capacitado é determinar como a distribuição do produto deve ser feita de modo a minimizar os custos.

Logo, o Problema de Transporte Capacitado tem a seguinte formulação:

$$\begin{aligned} \text{Minimizar: } & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}y_{ij} \\ \text{sujeito a: } & \sum_{j=1}^{n} y_{ij} = p_i, \qquad i = 1,2,\cdots,m \\ & \sum_{i=1}^{m} y_{ij} = q_j, \qquad j = 1,2,\cdots,n \\ & \alpha_{ij} \le y_{ij} \le \beta_{ij}, \quad i = 1,2,\cdots,m \text{ e } j = 1,2,\cdots,n \end{aligned} \tag{2.1.1}$$

Pretende-se fazer uma associação do Problema (2.1.1) a uma rede como ilustra a figura 2.1. Porém, antes de fazermos esta associação precisamos de algumas definições.

Um *grafo não-orientado* é uma estrutura $G = (V(G), A(G))$ na qual $V(G)$ é um conjunto finito, não-vazio, de pontos chamados *nós* ou *vértices*, e $A(G)$ é um subconjunto do

conjunto de pares não-ordenados de elementos de $V(G)$, chamados de *arestas*. Se i e j são elementos de $V(G)$ e constituem uma aresta de G, denota-la-emos por $[i,j]$. Neste caso, dizemos que os nós i e j são *adjacentes*, ou ainda, que a tal aresta *incide* sobre os nós i e j.

O *grau* de um nó em G é dado pelo número de nós adjacentes a ele.

Um *grafo orientado* é uma estrutura $G=(V(G),A(G))$ onde $V(G)$ é um conjunto finito, não-vazio, de pontos chamados *nós* ou *vértices*, e $A(G)$ é um subconjunto do conjunto de pares ordenados de elementos de $V(G)$, chamados de *arcos*. Assim um arco com extremidades i e j que pode ser percorrido somente no sentido de i para j será denotado pelo par ordenado (i,j). Dizemos também que os nós i e j representam a *cauda* e a *cabeça* do arco (i,j), repectivamente.

Uma *rede orientada* (*não-orientada*) é um grafo orientado (não-orientado) onde os nós e os arcos (as arestas) têm associados valores numéricos (geralmente custos, capacidades, e/ou ofertas e demandas).

Um *caminho* na rede (orientada) G do nó η_1 ao nó η_k, é uma seqüência de nós e arestas (arcos) alternados $\eta_1,e_1,\eta_2,e_2,\ldots,e_{k-1},\eta_k$, onde e_r é a aresta $[\eta_r,\eta_{r+1}]$ (o arco (η_r,η_{r+1}), ou o arco (η_{r+1},η_r)), para cada r de 1 a $k-1$, e e_r e η_r não se repetem ao longo desta seqüência. Neste caminho, os pontos η_1 e η_k são chamados de *nós iniciais* e *finais*, respectivamente. Note que ao percorrer um caminho numa rede orientada, os arcos não precisam ter necessariamente o mesmo sentido deste caminho.

Um *ciclo* na rede G é um caminho no qual os nós inicial e final coincidem. Uma rede que contiver um caminho entre cada par de nós é chamada *rede conexa*. Uma *subrede* da rede G é uma rede H na qual $V(H)\subseteq V(G)$ e $A(H)\subseteq A(G)$.

Uma *árvore* na rede G é uma subrede conexa sem ciclos. Uma árvore T é chamada *árvore geradora* da rede G, se $V(T)\equiv V(G)$. Uma *árvore geradora enraizada* de uma rede orientada G é uma árvore geradora de G com um de seus nós designado como *nó raiz* sobre o qual incide um arco designado como *arco raiz*.

Estamos apenas interessados em redes orientadas, assim usaremos a palavra rede para designar uma rede orientada no que se segue.

Considere a seguinte rede $G=(V(G),A(G))$ onde o conjunto de vértices *V(G)* é a união de dois conjuntos disjuntos: um de *m* origems $O=\{o_1,o_2,...,o_m\}$ e o outro de *n* destinos $D=\{d_1,d_2,...,d_n\}$; e o conjunto de arcos *A(G)* é o conjunto de ligações possíveis a partir da origem *i* ao destino *j*, ou seja (i,j) pertence a *A(G)* se existir um caminho que liga a origem *i* ao destino *j*. Associa-se a cada vértice de origem *i* a oferta p_i e a cada vértice de destino *j* a demanda q_j. Associa-se também a cada arco (i,j) em *A(G)* as capacidades inferior e superior e o custo c_{ij}. Sendo y_{ij} a quantidade do produto a ser enviada da origem *i* ao destino *j* pelo arco (i,j), o Problema de Transporte (2.1.1) fica associado a rede *G*. Ver figura 2.1 abaixo.

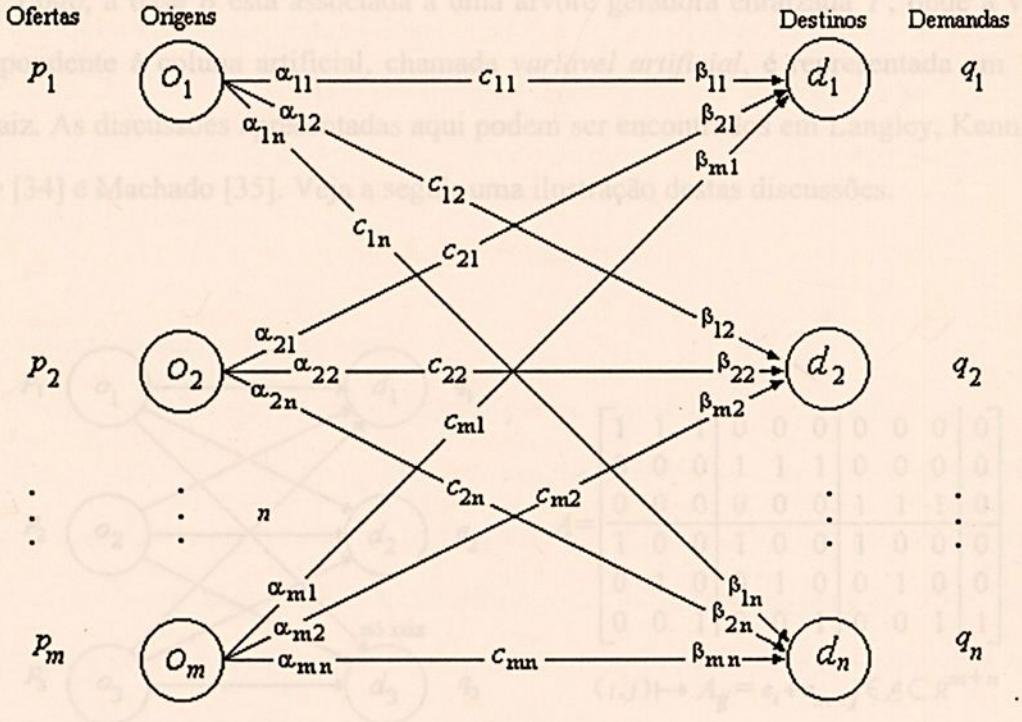


figura 2.1

2.2 OPERAÇÕES SIMPLEX

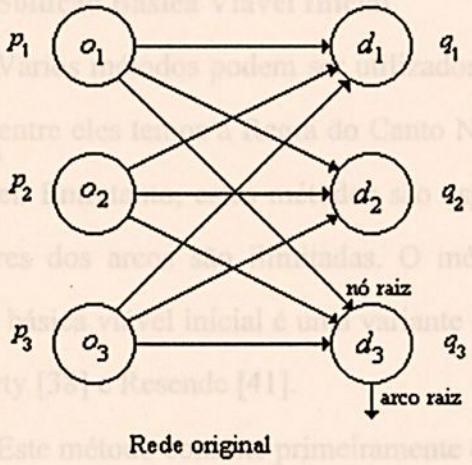
Nesta seção veremos como cada passo do Algoritmo Simplex pode ser aplicado para Problemas de Transporte Capacitados.

Solução Básica Viável

Denotemos por $\overset{o}{A}$ a matriz dos coeficientes do sistema dado em (2.1.1).

A matriz $\overset{o}{A}$ tem posto igual a $m+n-1$. Assim como no Capítulo 1, gostaríamos que a matriz $\overset{o}{A}$ tivesse posto completo igual a $m+n$, para que o Método Simplex possa ser aplicado com sucesso. Então, para que tenhamos a matriz dos coeficientes com posto completo $m+n$, introduzimos uma coluna em $\overset{o}{A}$, chamada *coluna artificial*, correspondente ao nó destino n (poderia ter sido escolhido qualquer outro nó), a qual deverá estar presente em toda base B com posto $m+n$. A cada arco (i,j) da rede está associada uma coluna $A_{ij} = e_i + e_{m+j} \in \mathrm{R}^{m+n}$ da nova matriz que passaremos a denotar por A, onde e_i representa um vetor com "1" na i-ésima posição e "0" nas demais posições, e e_{m+j} representa um vetor com "1" na $(m+j)$-ésima posição e "0" nas demais posições.

Logo, a base B está associada a uma árvore geradora enraizada T, onde a variável correspondente à coluna artificial, chamada *variável artificial*, é representada em T pelo arco raiz. As discussões apresentadas aqui podem ser encontrados em Langley, Kennington, Shetty [34] e Machado [35]. Veja a seguir uma ilustração destas discussões.

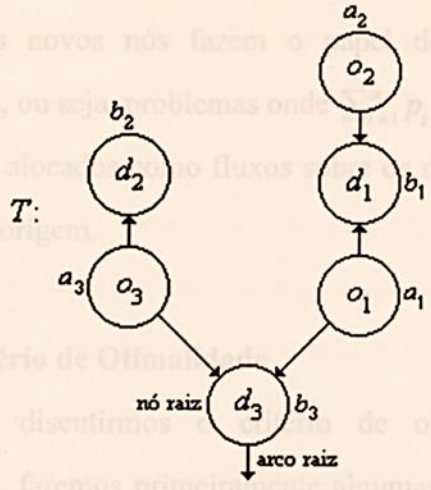


Rede original

$$A=\left[\begin{array}{ccc|ccc|ccc|c} 1&1&1&0&0&0&0&0&0&0\\ 0&0&0&1&1&1&0&0&0&0\\ 0&0&0&0&0&0&1&1&1&0\\ \hline 1&0&0&1&0&0&1&0&0&0\\ 0&1&0&0&1&0&0&1&0&0\\ 0&0&1&0&0&1&0&0&1&1 \end{array}\right]$$

$$(i,j) \mapsto A_{ij} = e_i + e_{m+j} \in A \subset R^{m+n}$$

Matriz dos coeficientes



Árvore Geradora Enraizada

$$B=\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Base de Trabalho

De agora em diante, denotaremos por (T, L, S) a estrutura da árvore geradora viável do Problema de Transporte Capacitado, onde T representa a árvore associada à base de trabalho, e L e S representam o conjunto dos arcos com fluxo em sua capacitade inferior e superior, respectivamente.

Solução Básica Viável Inicial

Vários métodos podem ser utilizados para a obtenção de uma solução básica viável inicial, entre eles temos a Regra do Canto Noroeste, Método do Custo Mínimo e o Método de Vogel. Entretanto, estes métodos são específicos para os casos em que as capacidades superiores dos arcos são ilimitadas. O método que utilizaremos para obtenção de uma solução básica viável inicial é uma variante do Método do M-Grande e pode ser encontrado em Murty [38] e Resende [41].

Este método consiste primeiramente em criar uma nova origem e um novo destino, e em seguida, conectá-los a todos os destinos e todas as origens na rede, respectivamente, conectando inclusive a nova origem ao novo destino. Desta maneira, obtemos a seguinte solução básica viável inicial:

$$x_{ij} = \begin{cases} p_i & \text{para } x_{i,n+1},\ i = 1,2,\cdots,m \\ q_j & \text{para } x_{m+1,j},\ j = 1,2,\cdots,n \\ 0 & \text{para } x_{m+1,n+1} \end{cases} \qquad (2.2.1)$$

Estes novos nós fazem o papel de *nós fictícios* no caso de problemas não balanceados, ou seja, problemas onde $\sum_{i=1}^{m} p_i \neq \sum_{j=1}^{n} q_j$. Neste caso, o excesso ou déficit de oferta serão alocados como fluxos sobre os novos arcos incidentes com o novo destino ou com a nova origem.

Critério de Otimalidade

Para discutirmos o critério de otimalidade para Problemas de Transporte Capacitados, faremos primeiramente algumas alterações no problema dado em (2.1.1), de forma a reduzir as capacidades inferiores a zero.

Assim, introduzindo-se a variável $x = y - \alpha$, o problema (2.1.1) toma a seguinte forma:

$$\begin{aligned} \text{Minimizar: } & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{sujeito a: } & \sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1,2,\cdots,m \\ & \sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1,2,\cdots,n \\ & 0 \le x_{ij} \le \varepsilon_{ij}, \quad i = 1,2,\cdots,m \quad j = 1,2,\cdots,n \end{aligned} \tag{2.2.2}$$

onde $cx = cy + c\alpha$ (função objetivo atualizada), $a_i = p_i - \sum_{j=1}^{n} \alpha_{ij}$ com $\le i \le m$ (ofertas atualizadas), $b_j = q_j - \sum_{i=1}^{m} \alpha_{ij}$ com $\le j \le n$ (demandas atualizadas), $\varepsilon_{ij} = \beta_{ij} - \alpha_{ij}$ com $\le i \le m$ e $\le j \le n$ (capacidades superiores atualizadas).

Note que, após a remoção das capacidades inferiores, devemos ter $a_i > 0$ e $b_j > 0$, pois do contrário, todos os arcos incidentes com o nó cuja oferta ou demanda atualizada seja nula, poderiam ser eliminados do equacionamento do problema uma vez que a quantidade do produto a ser transportada sobre tais arcos já estariam definidas pelas suas próprias capacidades inferiores antes da nova reformulação. Note ainda que a remoção das capacidades inferiores em nada alteraram a matriz dos coeficientes A, o que significa que manteremos a designação dada a matriz A anteriormente.

Dada uma solução básica viável $\bar{x}$, a próxima tarefa é verificar se esta é uma solução ótima, ou se a mesma pode ser melhorada de modo a decrescer ainda mais o valor da função objetivo.

Denotemos a matriz base de trabalho por $B = (A_{r_1,s_1}, \cdots, A_{r_{m+n-1},s_{m+n-1}}, e_{m+n})$, onde A_{r_i,s_i}, $i = 1,2,\cdots,m+n-1$, denotam as colunas da matriz A associadas com as variáveis básicas, e e_{m+n} denota a coluna artificial introduzida para obtenção do posto completo da matriz dos coeficientes.

Uma vez que as colunas da matriz A estão representadas por $A_{ij} = e_i + e_{m+j} \in \mathrm{R}^{m+n}$, o resultado obtido em (1.3.1) fornece a seguinte relação:

$$\bar{x}_B = B^{-1}b - \sum_{(i,j)\in I_L} B^{-1}\bar{x}_{ij}a_{ij} - \sum_{(i,j)\in I_S} B^{-1}\bar{x}_{ij}a_{ij} \qquad (2.2.3)$$

onde I_L e I_S representam o conjunto de índices das variáveis não-básicas em suas capacidades inferior e superior, respectivamente.

Substituindo a relação (2.2.3) na função objetivo do problema (2.2.2), após algumas operações algébricas, obtemos:

$$z = c_B B^{-1}b + \sum_{(i,j)\in I_L} \pi_{ij}\bar{x}_{ij} + \sum_{(i,j)\in I_S} \pi_{ij}\bar{x}_{ij}$$

onde $\pi_{ij} = c_{ij} - z_{ij}$, $z_{ij} = c_B B^{-1}A_{ij} = \psi A_{ij}$.

Assim como no Capítulo 1, também referimos a π_{ij} como *coeficiente de custo reduzido* da variável x_{ij}, enquanto que os *multiplicadores simplex* ψ_k, com $1 \le k \le m+n$, associados com a base de trabalho B, são obtidos da relação:

$$\psi = c_B B^{-1} \quad \text{ou} \quad \psi B = c_B \qquad (2.2.4)$$

De maneira similar àquela vista no Capítulo 1, os critérios de otimalidade são dados por:

$$\pi_{ij} \ge 0, \text{ para } (i,j) \in I_L$$

$$\pi_{ij} \leq 0, \text{ para } (i,j) \in I_S$$

A característica especial da matriz A faz com que os multiplicadores simplex sejam determinados de maneira prática e eficiente como podemos ver a seguir.

Denotando os multiplicadores simplex ψ_i para $1 \leq i \leq m$ por u_i, e ψ_j para $1 \leq j \leq n$ por v_j, o vetor ψ das variáveis duais é dado por:

$$\psi = (u_1, \cdots, u_m, v_1, \cdots, v_n)$$

Multiplique por "-1", as n últimas equações do problema (2.2.2). Agora, a coluna A_{ij} tem um "1" na i-ésima posição, e um "-1" na $(m+j)$-ésima posição, isto é, $A_{ij} = e_i - e_{m+j} \in \mathrm{R}^{m+n}$. Logo, fazendo as devidas substituições, obtemos:

$$\pi_{ij} = c_{ij} - z_{ij} = c_{ij} - \psi A_{ij} = c_{ij} - u_i + v_j .$$

Considerando c_a o custo associado com a *variável artificial* x_a (introduzida no vetor x devido a presença da coluna e_{m+n}), o vetor de custos básicos passa a ser dado por $c_B = (c_{r_1,s_1}, \cdots, c_{r_{m+n-1},s_{m+n-1}}, c_a)$.

Atribuindo um valor arbitrário a c_a, como por exemplo $c_a = 0$, e fazendo as devidas substituições na relação (2.2.4), obtemos o seguinte sistema:

$$(u_1, \cdots, u_m, v_1, \cdots, v_n)(A_{r_1,s_1}, \cdots, A_{r_m,s_m}, A_{r_{m+1},s_{m+1}}, \cdots, e_{m+n}) = (c_{r_1,s_1}, \cdots, c_{r_m,s_m}, c_{r_{m+1},s_{m+1}}, \cdots, c_a)$$

$$\begin{aligned}
(u_1, \cdots, u_m, v_1, \cdots, v_n) A_{r_1,s_1} &= c_{r_1,s_1} \\
&\vdots \\
(u_1, \cdots, u_m, v_1, \cdots, v_n) a_{r_m s_m} &= c_{r_m s_m} \\
&\vdots \\
(u_1, \cdots, u_m, v_1, \cdots, v_n) a_{r_{m+1},s_{m+1}} &= c_{r_{m+1},s_{m+1}} \\
&\vdots \\
(u_1, u_2, \cdots, u_m, v_1, v_2, \cdots, v_n) e_{m+n} &= 0
\end{aligned}$$

Ou seja:

$$
\begin{aligned}
u_{r_1} - v_{s_1} &= c_{r_1,s_1} \\
&\vdots \\
u_{r_m} - v_{s_m} &= c_{r_m,s_m} \\
u_{r_{m+1}} - v_{s_{m+1}} &= c_{r_{m+1},s_{m+1}} \\
&\vdots \\
v_n &= 0
\end{aligned}
\qquad (2.2.5)
$$

Portanto, aplicando o método da substituição a este novo sistema a partir de $v_n = 0$, obtemos os valores dos demais multiplicadores simplex.

Note que, a introdução da variável artificial na $(m+n)$-ésima equação do problema (2.2.2) através do vetor e_{m+n}, faz com que v_n seja o primeiro multiplicador simplex a ser obtido no sistema (2.2.5). Veja que o valor de v_n é idêntico ao valor arbitrário atribuído a c_a. Isto quer dizer que, atribuir valores arbitrários a c_a é equivalente a atribuir os mesmos valores ao seu correspondente multiplicador simplex v_n.

De um modo geral, a escolha do primeiro multiplicador simplex, u_i ou v_j, ao qual será atribuído um valor arbitrário, corresponde à introdução da coluna e_i ou e_{m+j} na matriz base de trabalho B, ou ainda, corresponde à introdução da variável artificial na i-ésima ou $(m+j)$-ésima equação. Podemos observar ainda que cada nó da rede está associado a um multiplicador simplex, e como o nó destino n foi designado como nó raiz, então o multiplicador simplex associado este nó é o primeiro multiplicador simplex a ser obtido no sistema (2.2.5).

Critério para Entrada do Arco

De acordo com o estudo do Método Simplex com Variáveis Limitadas dado no Capítulo 1, um arco candidato a entrar na árvore básica é qualquer arco não-básico $(i,j) \in L$ tal que $\pi_{ij} < 0$, ou $(i,j) \in S$ tal que $\pi_{ij} > 0$. No Capítulo 3 descreveremos um critério que melhor determina o arco que entra, para fins de implementação computacional.

Critério para Saída do Arco

Suponha que (k,l) seja o arco selecionado para entrar na base.

Adicionando-se o arco (k,l) a T, definimos um único ciclo C com a mesma orientação do arco (k,l) se $(k,l) \in L$, e orientação oposta a de (k,l) se $(k,l) \in S$.

Denotemos por $\overline{C}$ o conjunto de arcos em C cuja orientação é a mesma de C e denotemos por $\underline{C}$ o conjunto de arcos em C cuja orientação é contrária a de C.

Enviando fluxo adicional ao redor do ciclo na direção de sua orientação, o custo decresce estritamente na razão $\delta|\pi_{kl}|$ por unidade. Em seguida, aumentamos o fluxo tanto quanto possível até que um arco no ciclo C atinja sua capacidade inferior ou superior. O fluxo aumentado ao longo do ciclo C aumenta o fluxo sobre os arcos com orientação de C e diminue o fluxo sobre os arcos com orientação oposta a de C. Conseqüentemente, a mudança máxima de fluxo sobre um arco $(i,j) \in C$, denotada por δ_{ij}, satisfaz as seguintes restrições:

$$\delta_{ij} = \begin{cases} \varepsilon_{ij} - x_{ij}, & \text{se } (i,j) \in \overline{C} \\ x_{ij}, & \text{se } (i,j) \in \underline{C} \end{cases} \qquad (2.2.6)$$

Para manter a viabilidade, devemos aumentar $\delta = \min\{\delta_{ij} : (i,j) \in C\}$ unidades de fluxo ao longo do ciclo C. Um arco $(p,q) \in C$ tal que $\delta_{pq} = \delta$, é selecionado para sair da árvore T; existindo mais de um candidato a sair de T, seleciona-se um deles arbitrariamente.

2.3 ALGORITMO

O algoritmo para o Problema de Transporte Capacitado consiste nos seguintes passos:

Passo 1: Encontre uma solução básica viável inicial;

Passo 2: Calcule os multiplicadores simplex u_i's e v_j's em $u_i - v_j = c_{ij}$, para $(i,j) \in T$, com $v_n = 0$;

Passo 3: Calcule $\pi_{ij} = c_{ij} - u_i + v_j$ para cada arco não-básico (i,j).

Se $(i,j) \in L$ e $\pi_{ij} < 0$, ou $(i,j) \in S$ e $\pi_{ij} > 0$, então (i,j) é um arco candidato a entrar em T. Dentre estes candidatos, escolha aquele cujo π_{ij} é o de maior valor absoluto. Se tal candidato não existir então Pare, a solução atual é ótima se não existirem arcos artificiais na base com fluxo positivo, caso contrário, o problema não tem solução viável;

Passo 4: Determine o ciclo formado em T com a entrada do novo arco;

Passo 5: Determine o arco que sai de T;

Passo 6: Encontre a nova solução básica viável. Vá para o Passo 2.

2.4 CONVERGÊNCIA

Ciclagem

O algoritmo para o Problema de Transporte Capacitado não termina necessariamente em um número finito de iterações, a menos que a árvore T que representa a base de trabalho tenha uma estrutura especial chamada árvore geradora fortemente viável cuja definição é dada como segue. Uma árvore T é chamada *árvore geradora fortemente viável* se todo arco na árvore T com fluxo zero tiver mesmo sentido do arco raiz, e todo arco da árvore com fluxo igual a sua capacidade superior tiver sentido oposto ao do arco raiz. Em outras palavras, T é uma árvore geradora fortemente viável se pudermos enviar uma quantidade positiva de fluxo de qualquer nó ao nó raiz ao longo da árvore sem que qualquer uma das capacidades de fluxo sejam violadas.

Note que uma árvore geradora não-degenerada é sempre fortemente viável, enquanto que, uma árvore geradora degenerada pode ou não ser fortemente viável. Veremos no capítulo 3 que os valores atribuídos às capacidades superiores dos arcos farão com que a solução básica viável inicial dada em (2.2.1) seja fortemente viável.

O processo a seguir, proposto por Cunningham [16, 17], Barr, Glover e Klingman [4] descreve como podemos manter, em cada iteração, a árvore geradora fortemente viável quando a árvore inicial é uma árvore geradora fortemente viável.

Suponha que temos uma árvore geradora fortemente viável. Suponhamos também que durante uma operação pivô, (k,l) seja o arco selecionado para entrar na árvore.

Sejam ∇_k e ∇_l os caminhos que conectam k e l ao nó raiz. Adicionando-se o arco (k,l) a T, definimos um único ciclo C com a mesma orientacão do arco (k,l) se o fluxo estiver na sua capacidade inferior, ou sentido contrário a do arco (k,l) se o fluxo estiver na sua capacidade superior. Seja a o *ápice*, isto é, o primeiro nó comum aos caminhos ∇_k e ∇_l.

Após aumentarmos δ unidades de fluxo ao longo do ciclo C, o algoritmo identifica o arco candidato a sair da árvore T, isto é, um arco $(i,j) \in C$ tal que $\delta_{ij} = \delta$, onde δ_{ij} é dado pela relação (2.2.6). Se existir um único arco (i,j) candidato a sair de T, então este arco é escolhido para sair de T. Todavia, se existir mais de um arco candidato a sair de T, então a próxima árvore geradora será degenerada, e portanto, podendo não ser fortemente viável. Neste caso, o arco selecionado para sair de T deve ser o último arco candidato à sair de T quando percorremos o ciclo C no sentido da sua orientação a partir do ápice a. Desta maneira, este processo garante que na próxima árvore geradora cada nó sobre o ciclo C poderá enviar uma quantidade positiva de fluxo até o nó raiz, e portanto, tal árvore geradora será fortemente viável na próxima iteração.

Portanto, para obtermos árvores geradoras fortemente viáveis em cada iteração do Algoritmo de Transporte dado na seção anterior, devemos alterar o Passo-1 para "*Encontre uma solução básica inicial fortemente viável*", e em seguida, procedermos a escolha da variável que sai de T da maneira como foi mencionada acima.

"Stalling"

Cunningham [16] mostra que o fenômeno "stalling", explicado no capítulo anterior, pode ocorrer mesmo usando o método de Bland ou das árvores fortemente viáveis. Um dos métodos utilizados para evitar "stalling", que pode ser utilizado em qualquer contexto, foi apresentado no Capítulo 1. Agora descrevemos rapidamente outro método de escolha do arco candidato a entrar na base devido a Cunningham, o qual é específico para problemas de

fluxo em redes. Limitamos esta discussão ao problema de fluxo não capacitado. Cunningham [16] discute este caso e diz que seus resultados podem ser estendidos para problemas capacitados.

Seja $\eta_1, \eta_2, \ldots, \eta_{n+m}$ uma ordenação dos vértices da rede G, fixa mas arbitrária.

Sejam T uma árvore geradora básica fortemente viável e x^o uma solução básica viável correspondente. Suponha que a seja o arco que se tornou básico quando T passou a ser a árvore geradora básica em questão. Suponha também que o vértice t seja a cabeça do arco a. Seja $(k,l) \notin T$ e denote por $C(T,(k,l))$ o único ciclo formado com a inclusão do arco (k,l) na árvore T cuja orientação é no sentido de (k,l).

O custo $C_{(k,l)}(T)$ é soma dos custos dos arcos com sentido do ciclo menos a soma dos custos dos arcos com sentido contrário ao ciclo. Escolha o arco que entra na base da seguinte forma:

$\bar{e}$ entra na base se a cabeça de $\bar{e}$ for o primeiro elemento da lista $\eta_{t+1}, \ldots, \eta_{n+m}$, $\eta_1, \ldots, \eta_t$ tal que $C_{\bar{e}}(T) = Min\{C_e(T)$: cabeça de e = cabeça de $\bar{e}$ e $C_e(T) < 0\}$.

Com esta estratégia de escolha do arco que comporá a nova base, combinada com o critério de escolha do arco que deixa a base mantendo a nova árvore básica fortemente viável, o número de pivôs degenerados na execução do algoritmo é limitado polinomialmente com relação às dimensões do problema.

CAPÍTULO 3

IMPLEMENTAÇÃO DO ALGORITMO SIMPLEX PARA PROBLEMAS DE TRANSPORTE CAPACITADOS

3.1 INTRODUÇÃO

Este capítulo tem o propósito de descrever uma implementação do Algoritmo Simplex para Problemas de Transporte Capacitados baseando-se principalmente nos trabalhos de Bradley, Brown, Graves [11], Glover, Klingman [27], Klingman, Napier, Stutz [32], Langley, Kennington, Shetty [34] e Resende [41]. Na implementação, as árvores geradoras serão representadas convenientemente fazendo com que as operações básicas do algoritmo sejam efetuadas de maneira eficiente. Tal implementação originou *TRANSMAX*, um programa escrito em linguagem Pascal que resolve Problemas de Transporte esparsos ou densos, com ou sem arcos capacitados, e gera automaticamente nós e arcos artificiais no caso de problemas não-balanceados. *TRANSMAX* foi submetido a vários testes gerados por *GERAPTC*, um gerador aleatório de Problemas de Transporte Capacitados também implementado neste trabalho para tal propósito.

Reformulação do Problema (2.1.2)

Os principais avanços computacionais nos códigos para Problemas de Fluxo em Redes, baseados no Algoritmo Simplex Primal, se deve a melhoramentos na representação e manipulação dos dados do modelo. Assim, para melhorar a eficiência de um algoritmo é necessário entender bem as relações entre o algoritmo e as estruturas de dados empregados na sua implementação.

Para facilitar a armazenagem dos arcos do Problema de Transporte em questão e posterior manipulação dos dados, faremos uma reformulação do problema (2.1.2) e este passa a ser escrito da seguinte forma:

$$\begin{aligned} &\text{Minimizar} && \sum_{t\in\Gamma} c_t x_t \\ &\text{sujeito a:} && \sum_{t\in\Gamma_i^+} x_t = a_i\,, \quad i\in N_1 \\ & && \sum_{t\in\Gamma_j^-} x_t = b_j\,, \quad j\in N_2 \\ & && 0\le x_t\le \varepsilon_t,\ t\in\Gamma \end{aligned} \tag{3.1.1}$$

onde $\Gamma=\Gamma_i^+\cup\Gamma_j^-$ representa o conjunto de todos os arcos no problema, Γ_i^+ representa o conjunto dos arcos em Γ com cauda i, Γ_j^- representa o conjunto dos arcos em Γ com cabeça j, N_1 representa o conjunto dos nós origens e N_2 representa o conjunto dos nós destinos.

A notação utilizada em (3.1.1) nos permite armazenar somente os arcos que estão presentes na rede, isto é, a rede original não precisa ser necessariamente uma rede na qual todos os nós origens têm grau igual a n e todos os nós destinos têm grau igual a m.

Rotulação dos Nós

Os nós da rede do problema (3.1.1) são rotulados da maneira como segue.

A i-ésima linha da Matriz A de Transporte está associada com a i-ésima origem, e é representada na árvore geradora enraizada pelo nó rotulado pelo número serial i. A j-ésima coluna da Matriz A de Transporte está associada com o j-ésimo destino, e é representada na árvore geradora enraizada pelo nó rotulado pelo número serial $j+m$.

Portanto, os nós origens são rotulados por 1,2,...,m, e os nós destinos são rotulados por $m+1, m+3, \ldots, m+n$.

3.2 ESTRUTURA DE DADOS

De agora em diante, denote por T a árvore que representa a base de trabalho do problema (3.1.1), e por L e S os conjuntos das capacidades inferiores e superiores, respectivamente.

Armazenagem da Rede

Uma vez que matrizes esparsas são muito comuns em Problemas de Transporte de Grande Porte, não é eficiente armazenar a matriz de transporte, assim como as matrizes de capacidade superior e custos.

De acordo com Bradley, Brown, Graves [11], a rede de um Problema de Transporte será armazenada nos seguintes vetores:

KUDA(*): Armazena a cauda de cada arco;

PKBSA(*): Armazena a cabeça de cada arco de tal modo que todos os arcos com a mesma cabeça são armazenados em espaços contíguos na memória, pois deste modo, a lista de cabeças i é substituída por PKBSA(j)=i, onde i é o primeiro arco com a cabeça j;

KUSTO(*): Armazena o custo (por unidade do produto) a ser pago para transportar sobre cada arco.

KSUP(*): Armazena a capacidade (superior) de cada arco. Arcos não-básicos em suas capacidades superiores são identificados no vetor KSUP(*) pelo elemento -KSUP(*).

Armazenagem da Base

Seguindo Bradley, Brown, Graves [11], Glover, Klingman [27], Klingman, Napier, Stutz [32], Langley, Kennington, Shetty [34] e Resende [41], as estruturas de dados que representam a árvore T são dadas pelos seguintes vetores:

NIVEL(*): Armazena o *nível* de cada nó, isto é, NIVEL(i) armazena o número de arcos no caminho conectando o nó i ao nó raiz;

PREDSOR(*): Armazena o *antecessor* de cada nó, isto é, PREDSOR(i) armazena o nó de nível NIVEL(i)-1 no caminho conectando o nó i ao nó raiz. Um elemento dado por $-$PREDSOR(i) indica que um arco em T tem orientação de i para PREDSOR(i); caso contrário, o arco em T tem orientação de PREDSOR(i) para i;

PROXIMO(*): Armazena uma seqüência de nós que começa no nó raiz da árvore T e passa por todos os nós na mesma seqüência em que os correspondentes multiplicadores simplex são determinados no sistema (2.2.5);

ULTSUC(*): Armazena o *último sucessor* de cada nó, isto é, ULTSUC(i) armazena o nó cujo caminho deste ao nó raiz contém o nó i de tal modo que NIVEL(PROXIMO(ULTSUC(i)))≤NIVEL(i).

Os vetores que representam a árvore T são inicializados da seguinte maneira:

$$\text{NIVEL}(i) = \begin{cases} 2, & \text{para } m+2 \le i \le m+n+1 \\ 1, & \text{para } 1 \le i \le m+1 \\ 0, & \text{para } i = m+n+2 \end{cases}$$

$$\text{PREDSOR}(i) = \begin{cases} -m-n-2, & \text{para } 1 \le i \le m+1 \\ m+1, & \text{para } m+2 \le i \le m+n+1 \\ 0, & \text{para } i = m+n+2 \end{cases}$$

$$\text{PROXIMO}(i) = \begin{cases} 0, & \text{para } i = m+n+2 \\ i+1, & \text{para } 1 \le i \le m+n+1 \end{cases}$$

$$\text{ULTSUC}(i) = \begin{cases} i, & \text{para } \{1 \le i \le m\} \cup \{m+2 \le i \le m+n+1\} \\ m+n+1, & \text{para } \{i = m+1\} \cup \{i = m+n+2\} \end{cases}$$

Operações Simplex

As operações simplex são efetuadas através das seguintes estruturas de dados:

LISTANB(*): Armazena todos os arcos não-básicos;

LISTATIVA(*): Armazena até λ arcos não-básicos com o melhor coeficiente de custo reduzido;

MSIMPLEX(*): Armazena os multiplicadores simplex;

FLUXO(*): Armazena o fluxo sobre o arco (i,PREDSOR(i)) para cada nó i em T,exceto para o nó raiz.

3.3 IMPLEMENTANDO AS OPERAÇÕES SIMPLEX

As operações simplex mencionadas aqui estão baseadas nas referências Bazaraa, Jarvis, Sherali [6], Bradley, Brown, Graves [11], Langley, Kennington, Shetty [34] e Resende [41].

Solução Básica Viável Inicial

Para evitar ciclagem em problemas de redes, Cunningham [15] sugeriu um refinamento do Método Simplex que consiste em conservar uma estrutura especial conhecida como árvore geradora enraizada fortemente viável, em cada iteração. Lembremos que, uma *árvore geradora enraizada fortemente viável* para o problema (3.1.1), incluindo o arco raiz, é uma árvore geradora enraizada na qual todo arco degenerado (i,j) com fluxo na capacidade inferior é tal que j encontra-se no caminho do nó i ao nó raiz e todo arco degenerado (i,j) com fluxo na capacidade superior é tal que i encontra-se no caminho do nó j ao nó raiz.

Assim, é necessário, para inicializar o algoritmo, encontrar uma árvore geradora enraizada fortemente viável para o problema (3.1.1).

Considerando a solução básica viável inicial dada em (2.2.1), vemos que o vetor FLUXO(*) é inicializado da seguinte maneira:

$$\text{FLUXO}(i) = \begin{cases} a_i & , \text{ se } 1 \le i \le m+1 \\ b_{i-m-1}, & \text{se } m+2 \le i \le m+n+1 \end{cases}$$

onde a demanda do novo destino vale $b_{n+1} = \sum_{i=1}^{m} a_i$, e a oferta da nova origem vale $a_{m+1} = \sum_{j=1}^{n} b_j$. Um arco (i, j) com capacidade superior ilimitada tem seu valor no vetor KPSUP(*) dado por $(a_i + b_j)$, $i = 1,...,m+1$ e $j = m+2,...,m+n+2$.

Então, a solução básica viável inicial obtida anteriormente constitui uma árvore geradora enraizada fortemente viável. A árvore inicial conserva a estrutura fortemente viável quando utilizamos a regra dada na subseção que determina o arco que sai da árvore.

Determinação do arco que entra

Vimos no Capítulo 1 que, em cada iteração, o Método Simplex requer o cálculo dos coeficientes de custo reduzido de todos os arcos não-básicos para determinar o arco que entra na árvore *T*, acarretando com isto um consumo de tempo muito grande. O esquema de custo que utilizaremos é uma variante do método de Dantzig, o qual considera até λ arcos não-básicos com o melhor coeficiente de custo reduzido e somente estes arcos são considerados a tornarem-se básicos durante as iterações do algoritmo. Nesta operação utilizamos as listas ligadas LISTANB(*) e LISTATIVA(*). O arco não-básico que entrar na árvore *T* é substituído na lista LISTATIVA(*) pelo arco que sai de *T*. Se dentre os λ arcos de LISTATIVA(*) não existir pelo menos um a se tornar básico, o esquema reconstrói LISTATIVA(*) com até λ arcos com o melhor coeficiente de custo reduzido a partir de LISTANB(*), e determina se existir, um candidato a entrar na base. Este procedimento é repetido até que não existam mais candidatos a entrar na árvore *T*. Neste momento o algoritmo determina se a solução é ótima ou inviável.

Determinação do arco que sai

Seja *w* o arco com extremidades *k* e *l* selecionado para entrar na árvore *T*. Sejam ∇_k e ∇_l os caminhos que conectam os nós *k* e *l* ao nó raiz. Adicionando-se o arco *w* a *T*, definimos um único ciclo *C* com a mesma orientacão do arco *w* se $w \in L$, e orientação oposta a de *w* se $w \in S$.

Os caminhos ∇_k e ∇_l são gerados a partir dos vetores PREDSOR(*) e NIVEL(*).

Com a informação do vetor NIVEL(*) é possível iterar sincronizadamente os caminhos ∇_k e ∇_l para identificar o ápice *a*. O vetor NIVEL(*) indica qual a profundidade do ápice *a* em *T* e qual caminho será iterado primeiro. Quando ambos os caminhos tiverem os mesmos níveis, os nós são comparados por igualdade. Uma igualdade indica o nó ápice *a* e uma desigualdade indica que ambos os caminhos deverão ser iterados para uma outra comparação.

De acordo com a restrição (2.2.6), para que a viabilidade seja mantida, devemos aumentar $\delta = \min\{\delta_t : t \in C\}$ unidades de fluxo ao longo de *C*, onde:

$$\delta_t = \begin{cases} \varepsilon_t - x_t, & \text{se } t \in \overline{C} \\ x_t, & \text{se } t \in \underline{C} \end{cases}$$

Então, um arco w' tal que $\delta_{w'} = \delta$, será um arco candidato a sair da árvore *T*.

Percorrendo o ciclo *C* ao longo da sua orientação, à partir do ápice *a*, selecionamos o último destes candidatos para sair da árvore *T*, pois desta maneira fazemos com que *T* conserve a propriedade de ser fortemente viável.

O ciclo *C* é percorrido duas vezes. Na primeira vez determinamos o ápice e o maior fluxo possível que pode ser aumentado ao longo de *C*. Na segunda vez ajustamos os fluxos.

3.4 ATUALIZAÇÃO DA ÁRVORE

As referências Bradley, Brown, Graves [11], Langley, Kennington e Shetty [34] e Resende [41] podem ser consultadas no sentido de obter maiores detalhes sobre o estudo de atualização da árvore que constitui a base de trabalho.

Sejam *k* e *l* as extremidades do arco que entra, e *p* e *q* as extremidades do arco que sai, de maneira que ∇_l contenha o arco que sai de *T*.

Suponha, para efeito de ilustração, que NIVEL(*q*)>NIVEL(*p*) (os casos em que NIVEL(*q*)=NIVEL(*p*) ou NIVEL(*q*)<NIVEL(*p*) podem ser analisados de maneira similar).

Chamamos de *ramo principal* o caminho conectando o nó l ao nó q, e de *ramo secundário* o caminho conectando o nó l ao nó p.

Considere a árvore T da figura 3.4.1, uma árvore obtida de uma iteração qualquer na qual a orientação dos arcos é ignorada. Suponha, neste caso, que temos $k = 4$, $l = 1$, $p = 7$, $q = 8$, e designemos o nó raiz por $r = 16$.

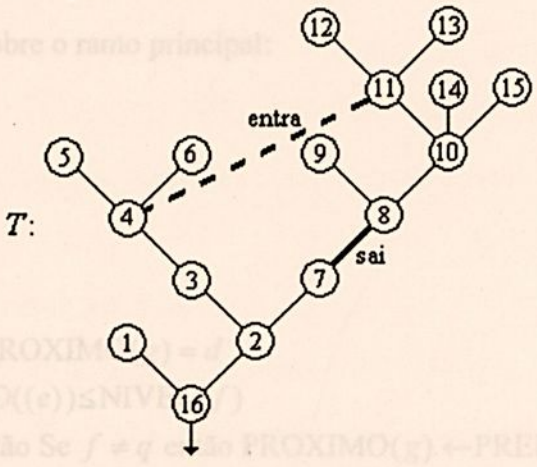


Fig. 3.4.1

De agora em diante denote por T_j a subárvore de T enraizada em j, e por T^* a árvore T atualizada.

Atualizando o vetor PROXIMO(*)

A figura 3.4.2 mostra a seqüência dos nós no vetor PROXIMO(*), onde a flecha pontilhada entre dois nós i e j representa o espaço contíguo ocupado pelos nós i e j no vetor PROXIMO(*), isto é, PROXIMO(1) = 2, PROXIMO(2) = 3,..., PROXIMO(15) = 16.

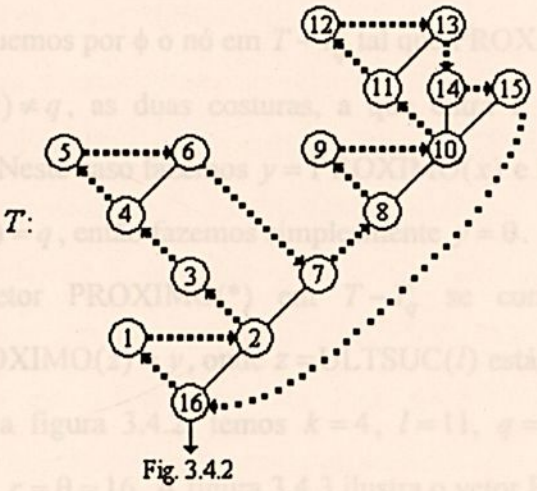


Fig. 3.4.2

Usaremos o termo "costura" para representar o par (i,PROXIMO(i)). Desta maneira, quando PROXIMO(i) = j diremos que existe uma costura saindo do nó i e chegando em j.

Inicialmente, mostremos como atualizar o vetor PROXIMO(*) em T_q, ou ainda, como enraizar a subárvore T_q em l.

O algoritmo a seguir nos dá uma idéia geral de como atualizar o vetor PROXIMO(*) em T_q, onde i é um nó sobre o ramo principal:

$d \leftarrow i$
$e \leftarrow$ULTSUC(d)
$f \leftarrow$PREDSOR(d)
PROXIMO(e) $\leftarrow f$
Determine g tal que PROXIMO(g) = d
Se NIVEL(PROXIMO((e))$\leq$NIVEL(f)
então Se $g \neq f$ então Se $f \neq q$ então PROXIMO(g) $\leftarrow$PREDSOR((f)
senão $z \leftarrow g$
senão PROXIMO(g) $\leftarrow$PROXIMO(e)
Se $f = q$ então $z \leftarrow$ULTSUC(f)

Agora, mostremos como atualizar o vetor PROXIMO(*) em $T - T_q$.

Observando a figura 3.4.2, vemos que existe uma costura entrando em T_q e outra saindo para reentrar em $T - T_q$; à saber, a costura que une os nós 4 e 11, e a costura que une os nós ULTSUC(q) = 15 e r = 16, respectivamente.

Indiquemos os nós ULTSUC(k) e PROXIMO(ULTSUC(q)) por x e θ, respectivamente, e indiquemos por ϕ o nó em $T - T_q$ tal que PROXIMO(ϕ) = q.

Se PROXIMO(x) $\neq q$, as duas costuras, a que entra e a que sai, precisam ser cortadas e reconectadas. Neste caso fazemos y = PROXIMO(x) e PROXIMO(ϕ) = θ.

Se PROXIMO(x) = q, então fazemos simplesmente $y = \theta$.

O ajuste do vetor PROXIMO(*) em $T - T_q$ se completa quando fazemos PROXIMO(x) = l e PROXIMO(z) = y, onde z =ULTSUC(l) está em T^*.

De acordo com a figura 3.4.2, temos $k = 4$, $l = 11$, $q = 8$, $a = 2$ (ápice), $x = 6$, $p = y = \phi = 7$, $z = w = 9$, $r = \theta = 16$. A figura 3.4.3 ilustra o vetor PROXIMO(*) atualizado.

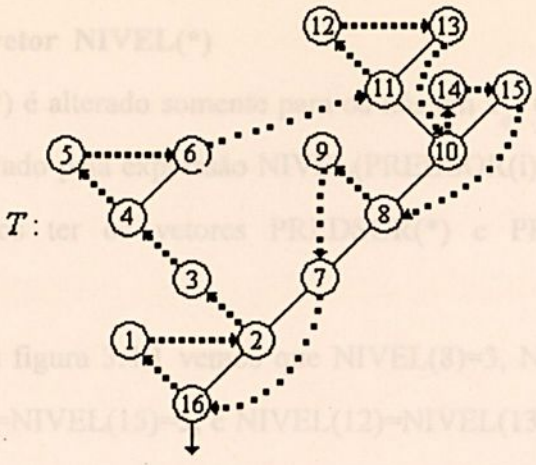


Fig. 3.4.3

Atualizando o vetor ULTSUC(*)

Seja z o nó obtido na atualização do vetor PROXIMO(*). Todos os nós i's em $T - T_q$ tais que ULTSUC(i)=ULTSUC(k) terão seus valores em ULTSUC(*) ajustados para ULTSUC(i) = z, enquanto que todos os nós j's em $T - T_q$ tais que ULTSUC(j) =ULTSUC(q) terão seus valores em ULTSUC(*) ajustados para ULTSUC(j) = ϕ.

De acordo com a figura 3.4.1, os nós 3 e 4 têm o nó 6 como último sucessor, enquanto que os nós 2 e 7 têm o nó 7 como último sucessor. Isto quer dizer que devemos ter em T^* o nó 9 como último sucessor dos nós 3 e 4, enquanto que o nó 7 será o último sucessor dos nós 2 e 7.

Atualizando o vetor PREDSOR(*)

O antecessor será alterado somente para os nós sobre o ramo principal. Neste momento fazemos PREDSOR(l) = k, em seguida, fazemos PREDSOR(v) = $\pm u$ para todos os nós u e v sobre o ramo principal tais que PREDSOR(u) = $\pm v$.

De acordo com a árvore T da figura 3.4.1, os elementos do vetor PREDSOR(*) que devem ter seus valores atualizados são: PREDSOR(11)=10, PREDSOR(10)=8 e PREDSOR(8)=7. Então em T^* devemos ter PREDSOR(8)=10, PREDSOR(10)=11 e PREDSOR(11)=4. Note que aqui não estamos considerando a orientação dos arcos.

Atualizando o vetor NIVEL(*)

O vetor NIVEL(*) é alterado somente para os nós em T_q. Cada nó i em T_q terá seu valor em NIVEL(*) alterado pela expressão NIVEL(PREDSOR(i))+1. Para obtermos êxito nesta operação, devemos ter os vetores PREDSOR(*) e PROXIMO(*) atualizados anteriormente.

De acordo com a figura 3.4.1 vemos que NIVEL(8)=3, NIVEL(9)=NIVEL(10)=4, NIVEL(11)=NIVEL(14)=NIVEL(15)=5, e NIVEL(12)=NIVEL(13)=6. Então, na árvore T^* teremos, NIVEL(11)=4, NIVEL(12)=NIVEL(13)=NIVEL(10)=5, NIVEL(14)=NIVEL(15)= NIVEL(8)=6 e NIVEL(9)=7.

Atualizando o vetor MSIMPLEX(*)

Todos os nós em T_q terão seus valores no vetor MSIMPLEX(*) aumentados por $-\pi_{kl}$ se o arco que entrar em T for (k,l), e terão seus valores aumentados por π_{lk} se o arco que entrar em T for (l,k).

Atualizando o vetor FLUXO(*)

Seja w' o arco selecionado para sair de T. Utilizando o vetor (atualizado) PREDSOR(*) a partir do nó $i=q$, fazemos FLUXO(i)=FLUXO(PREDSOR(i)) até atingirmos i=PREDSOR(l). Finalmente, para $i=l$ fazemos FLUXO(i)=$\delta_{w'}$ se o arco (i,PREDSOR(i)) estiver em sua capacidade superior, e fazemos FLUXO(i)=FLUXO(i)-$\delta_{w'}$ se o arco (i,PREDSOR(i) estiver em sua capacidade inferior.

A figura 3.4.4 ilustra a atualização completa da árvore dada na figura 3.4.1.

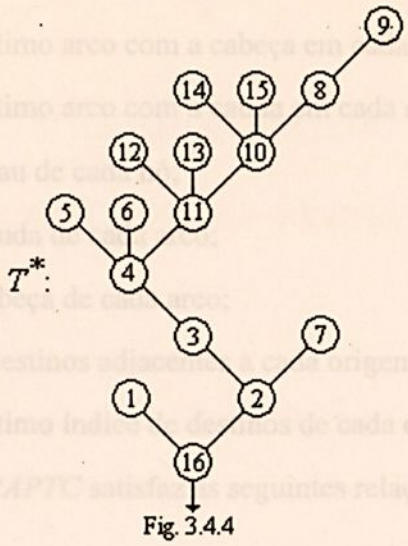


Fig. 3.4.4

3.5 GERADOR

As discussões deste capítulo originou o programa, que denominamos *TRANSMAX* cujo desempenho é avaliado através de problemas testes de várias dimensões gerados por um gerador aleatório (*GERAPTC*) *de* Problemas de Transporte Capacitados ou não, o qual também foi implementado neste trabalho.

Nesta seção fazemos um breve estudo sobre a construção do gerador aleatório de Problemas de Transporte Capacitados.

O usuário tem pleno domínio sobre o gerador pois o mesmo pode controlar os parâmetros referente ao número de origens, número de destinos, quantidade de arcos (capacitados ou não) que a rede deve conter. Cada um destes parâmetros tem a opção de ser gerado aleatoriamente ou ser digitado pelo próprio usuário.

A construção de *GERAPTC* é dividida em duas partes.

A primeira parte consiste em gerar o que chamamos de *esqueleto da rede*, ou seja, a rede que representará o Problema de Transporte Capacitado. Neste momento também são introduzidas as quantidades de origens, de destinos e de arcos definindo assim o tamanho do problema a ser gerado.

A estrutura de dados utilizada neste passo é a seguinte:

UKBSA[*] :Armazena o último arco com a cabeça em cada destino;

UKUDA[*] :Armazena o último arco com a cauda em cada origem;

GRAU[*] :Armazena o grau de cada nó;

KUDA[*] :Armazena a cauda de cada arco;

KBSA[*] :Armazena a cabeça de cada arco;

DESTINOS[*]:Armazena os destinos adjacentes a cada origem;

IUD[*] :Armazena o último índice de destinos de cada origem.

A rede gerada por *GERAPTC* satisfaz as seguintes relações:

$$\sum_{i=1}^{m-1} GRAU[i] \geq n \qquad GRAU[i] \geq 2,\ i = 1,\ldots,m+n \qquad \sum_{i=1}^{m} GRAU[i] = \text{(Total de arcos)}$$

A segunda parte destina-se a gerar os demais dados tais como as ofertas, demandas, custos e capacidades dos arcos.

CAPÍTULO 4

RESULTADOS OBTIDOS E CONCLUSÕES FINAIS

4.1 INTRODUÇÃO

Neste capítulo apresentamos os resultados obtidos com o *TRANSMAX* e também discutimos os caminhos para trabalhos futuros.

4.2 RESULTADOS OBTIDOS

Assim como na prática, os problemas testados têm número de destinos muito maior do que o número de origens (lembramos que o programa desenvolvido neste trabalho também resolve os demais casos).

GERAPTC gerou redes para Problemas de Transporte Capacitados onde as capacidades inferiores eram iguais a zero, enquanto que a escolha de um arco ser capacitado superiormente ou não foi feita de maneira aleatória. Os custos tiveram valores reais compreendidos entre 1 e 100. As ofertas e as demandas tiveram valores inteiros variando entre 9 e 606, de modo a obtermos problemas balanceados e não-balanceados. O experimento foi realizado em um microcomputador 486 DX2 de 66 Mhz com 4Mb de memória RAM.

Foram gerados 10 problemas de ordem 5x10 que denominamos problemas do tipo-I, 10 problemas de ordem 10x15 que denominamos problemas do tipo-II, e 10 problemas de ordem 15x50 que denominamos problemas do tipo-III. Para cada valor de λ foram registradas as médias do número de iterações e do tempo de CPU dos 10 problemas de cada tipo. As tabelas 3.4.5, 3.4.6 e 3.4.7 ilustram os resultados obtidos para os problemas de tipos I, II e III, respectivamente.

Tipo-I

m=5 n=10 Total de arcos: 50 Total de arcos capacitados: 42			
λ	Total de iterações (Média)	Tempo de CPU em segundos (Médio)	Variância
1	28	00 : 09	0.0008
10	30	00 : 01	0.0005
20	29	00 : 04	0.0006
30	29	00 : 03	0.0006
40	29	00 : 03	0.0007
50	28	00 : 04	0.0006

Tabela 3.4.5

Tipo-II

m=10 n=15 Total de arcos: 40 Total de arcos capacitados: 33			
λ	Total de iterações (Média)	Tempo de CPU em segundos (Médio)	Variância
1	37	00 : 07	0.0008
8	38	00 : 03	0.0008
16	37	00 : 04	0.0003
24	37	00 : 03	0.0007
32	38	00 : 05	0.0000
40	36	00 : 05	0.0000

Tabela 3.4.6

Tipo-III

m=15 n=50 Total de arcos: 200 Total de arcos capacitados: 163			
λ	Total de iterações (Média)	Tempo de CPU em segundos (Médio)	Variância
1	143	03 : 31	0.0410
40	148	00 : 85	0.0038
80	150	00 : 95	0.0113
120	156	01 : 13	0.0143
160	149	01 : 30	0.0146
200	142	01 : 54	0.0336

Tabela 3.4.7

4.3 CONCLUSÕES FINAIS

Analisando as tabelas percebemos que λ igual a 1 corresponde ao pior tempo de CPU dos problemas, enquanto que o tempo de execução é mais lento para os problemas cujo valor de λ supera 1/5 do total de arcos. E observa-se também que λ igual a 1/5 do total de arcos nos dará o melhor tempo de CPU, porém essa certeza só poderá ser dada através de testes estatísticos adequados, os quais deixamos para um trabalho futuro.

Pretendemos também, num trabalho futuro, adaptar o código *TRANSMAX* para resolver problemas mais gerais de fluxo a custo mínimo e testá-lo em problemas grandes. Depois, comparar sua performance com códigos de domínio público e pacotes comerciais para solução de problemas de grande porte em redes. Além disso, pretendemos também implementar uma subrotina que permite evitar o fenômeno "stalling" e estender os resultados de Cunningham [16] para redes capacitadas.

Uma outra questão que merece ser investigada é a busca e refinamento de métodos para evitar ciclagem e/ou "stalling" em algoritmos simplex duais para problemas em redes.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] AHUJA, R. K., MAGNANTI, T. L., ORLIN, J. B. Network flows,Theory Algorithms, and Applications. Printice Hall, New Jersey, 1993.

[2] AKGÜL, M. A Note On Lexicographic Linear Programming. *INFOR* v.22, n.4, 1984.

[3] BALAS, E., and HAMMER, P. L. On The Transportation Problem-Part I, *Cahiers du Centre d'Etudes de Recherche Operationalle*, v.4, n.2 , pp. 98, 1962.

[4] BARR, R. S., GLOVER, F., KLINGMAN, D. A. An Improved Version of the Out-kilter Method and a Comparative Study of Computer Codes, *Mathematical Programming*, v.7, n.1, p. 60, August 1974.

[5] BARR, R. S., GLOVER,F., KLINGMAN, D. A New Optimization Method for Large Scale Fixed Charge Transportation Problems. *Operations Research* v.29, n.3, 1981.

[6] BAZARAA, M. S., JARVIS, J. J., SHERALI, H. D. Linear Programming and Network Flows. 2nd ed., Wiley, New York, 1990.

[7] BERTSEKAS. D. P., CASTANON, D. A. A Generic Algorithm for the Minimum Cost Network Flow Problem, Journal Computacional Optimization and Application, n.3, pp.229-259, 1993.

[8] BLAND, R. G. New Finite Pivoting Rules For The Simplex Method, *Mathematics of Operations Research*, 2, pp. 103-107, 1977.

[9] BOAVENTURA, P. O. Teoria dos grafos. *I Colóquio de Pós-Graduação e Pesquisa em Matemática Aplicada e Computacional*. UNESP-São José do Rio Preto, 1995.

[10] BRADLEY, G. H. Survey of Deterministic Network. *AIIE Transactions* v.7, n.3, p.222, September 1975.

[11] BRADLEY, G. H., BROWN, G. G., GRAVES, G. W. Design and Implementation of Large Scale Primal Transshipment Algoritms. *Management Science* v.24, n.1, 1977-Excepcional Paper.

[12] BUSACKER, R. G. and SAATY, T. Finite Graphs and Networks. McGraw-Hill, New York, 1965.

[13] CHARNES, A., and COOPER, W. W. Management Models and Industrial Applications of Linear Programming, v.I, II, John Wiley and Sons, New York, 1961.

[14] CLASEN, R., J. The Numerical Solution of Network Problems Using the Out-of-Kilter Algorithm, *RAND Memorandum* RM-5456, Santa Monica, California, March 1968.

[15] CUNNINGHAM, W.H. A Network Simplex Method, *MP*, n.11, pp. 105-116, 1976.

[16] CUNNINGHAM, W.H. Theorical Properties of the Network Simplex Method, *MOR*, n.4, pp. 196-208, 1979.

[17] CUNNINGHAM, W.H., KLINCEWI CZ, J. G. On Cycling in the Network Simplex Method, *MP*, n.26, pp. 182-189, 1983.

[18] DANTZIG, G. B. Application of the Simplex Method to a Transportation Problem in Activity Analysis of Production and Allocation, T. C. Koopmans, ed., John Wiley and Sons, New York, 1951.

[19] DANTZIG, G. B. Notes On Linear Programming:PartsVIII, IX, X-Upper Bounds, Secondary Constraints, and Block Triangularity in Linear Programming, Research

Memorandum RM-1367, The Rand Corporation, Santa Monica, California, October 1954, also published in Econometria, 23 (2), pp. 174-183, April 1955.

[20] DANTZIG, G. B. Linear Programming and Extensions. *Princeton University Press*, Princeton, New Jersey, 1963.

[21] EDMONDS, J. and KARP, R. M. Theorical Improviments in Algorithmic Efficiency for Network Flow Problems, *Journal of the Association for Computing Machinery*, v.19, n.2, pp 248, April 1972.

[22] ELMAGHRABY, S. Some Network Models in Operations Research. *Springer-Verlag*, New York, 1970.

[23] FORD, L. R. and FULKERSON, D. R. Flow in Networks. *Princeton University Press*, Princeton, New Jersey, 1962.

[24] FULKERSON, D. R., Flow Networks and Combinatorial Operations Research. *American Mathematical Monthly* v.73, n. 2, p.115, February 1966.

[25] GILL, P. E., W. MURRAY, M. A. SAUNDERS, and M. H. WRIGHT Pratical Anti-Cicling Procedure For Linear And Nonlinear Programming, *Technical Report SOL 88-4*, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA 94305, 1988.

[26] GLICKSMAN, S., JOHNSON, L., ESELSON, L. Coding the Transportation Problem, *Naval Research Logistics Quarterly* v.7, n.2, p. 169, June 1960.

[27] GLOVER, F., KARNEY, D., KLINGMAN, D. Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems, *Networks* v.4, n.3, p. 191, 1974.

[28] GLOVER, F.,KLINGMAN, D. Recent Developments in Computer Implementation Technology for Network Flow Algoritms. *INFOR* v.20, n.4, 1982.

[29] HITCHCOCK, F. L. The Distribution of a Product from Several Sources to Numerous Localities. *Journal of Mathematics and Physics* v.20, n.2, p.224, April 1941.

[30] JOHNSON, E. L. Networks and Basic Solutions, *Operations Research* v.14, n.4, p. 619, August 1966.

[31] KLEIN, M. A Primal Method for Minimal Cost Flows with Application to the Assignment and Transportation Problems, *Manegement Science*, v.14, n.3, pp 205, November 1967.

[32] KLINGMAN, D., NAPIER, A., STUTZ, J. Netgen - A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. *Management Science* v.20, n.5, 1974.

[33] KOOPMANS, T. C. Optimum Utilization of the Transportation System. *Proceedings of the International Statistical Conferences*, Washington, D. C., 1947, published in v.5, 1949, pp. 136.(Also in Scientific Papers of Tjalling C. Koopmans, Springer-Verlag, New York, pp. 184, 1970).

[34] LANGLEY, R. W., KENNINGTON, J., SHETTY, C. M. Efficient Computational Devices for the Capacitated Transportation Problem. *Naval Research Logistics Quarterly* v.4, n.21, 1974.

[35] MACHADO, H. V. Programação Linear. *10º Colóquio Brasileiro de Matemática*. IMPA-Poços de Caldas, 1975.

[36] MARSHAL, K. T., and J. W. SUURBALLE A Note On Cicling In The Simplex Method, *Naval Research Logistics Quarterly*, 16, pp 121-137, 1969.

[37] MOALA, M. M. Como Implementar Problemas de Transporte Capacitados de Grande Porte. *I Colóquio de Pós-Graduação e Pesquisa em Matemática Aplicada e Computacional*. UNESP-São José do Rio Preto, 1995.

[38] MURTY, K. G. Linear and Combinatorial Programming, John Wiley and Sons, 1976.

[39] MURTY, K. G. Linear Programming, Wiley, New York, 1983.

[40] ORLIN, J. B. On The Simplex Algorithm For Network and Generalized Networks, *Mathematical Programming*, 24, pp. 166-178, 1985.

[41] RESENDE, L. I. P. Design and Implementation of a Computer Program for Large Scale Transportation Problems. *Pesquisa Operacional* v.4, n.2, 1984.