Instituto de Matemática, Estatística e
Computação Científica
Universidade Estadual de Campinas

Método Primal-Dual de Pontos Interiores Aplicado ao Problema de Multifluxo

Tese de Doutorado

Valéria de Podestá Gomes

Orientador: Clovis Perin Filho

DMA - IMECC - UNICAMP

Campinas, 30 de julho de 1999.

MÉTODO PRIMAL-DUAL DE PONTOS INTERIORES APLICADO AO PROBLEMA DE MULTIFLUXO

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por
VALÉRIA DE PODESTÁ GOMES
e aprovada pela comissão julgadora.

Campinas, 30 de julho de 1999

Prof. Dr. Clovis Perin Filho
Orientador

Banca Examinadora:

- Prof. Dr. Clovis Perin Filho
- Prof. Dr. Marcos Nereu Arenales
- Prof. Dr. Aurélio Ribeiro Leite de Oliveira
- Prof. Dr. José Mário Martínez Pérez
- Prof. Dr. Antônio Carlos Moretti

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para obtenção do Título de DOUTORA em Matemática Aplicada

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Gomes, Valéria de Podestá

P751m Método primal-dual de pontos interiores aplicado ao problema de multifluxo / Valéria de Podestá Gomes -- Campinas, [S.P. :s.n.], 1999.

Orientador : Clovis Perin Filho

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Sistemas lineares. 2. Programação (Matemática). 3. Programação linear. I. Perin Filho, Clovis. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Tese de Doutorado defendida em 30 de julho de 1999 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.:

Prof (a). Dr (a). CLOVIS PERIN FILHO

Prof (a). Dr (a). MARCOS NEREU ARENALES

Prof (a). Dr (a). AURÉLIO RIBEIRO LEITE DE OLIVEIRA

Prof (a). Dr (a). JOSÉ MÁRIO MARTÍNEZ PÉREZ

Prof (a). Dr (a). ANTONIO CARLOS MORETTI

"Neste velho mundo, a única coisa nova somos nós mesmos..."

Agradecimentos

Ao professor Clovis Perin pelas idéias, contribuições, sugestões oportunas, estímulo, disponibilidade e paciência constantes, sem os quais não seria possível a realização deste trabalho.

Aos amigos, pelo apoio recebido durante este longo tempo, em especial à Cristina, Anamaria e Sílvio.

Ao Lúcio, por tantas diferentes sugestões.

Aos colegas do Departamento de Matemática Aplicada.

À Fátima e à Secretaria de Pós-Graduação.

Ao cinema e à música, meu descanso.

Ao pai, perdido.

Para Alexandre e Elisa

Resumo

O problema de *Multifluxo* (Fluxo de Multiproduto) em uma rede é um modelo de Programação Matemática com muitas aplicações práticas. Neste trabalho, apresentamos um estudo computacional do MétodoPrimal-Dual de Pontos Interiores aplicado ao problema de *Multifluxo*. Destacamos a resolução do sistema linear das direções, onde é utilizado o método dos Gradientes Conjugados com uma combinação dos precondicionadores *Diagonal* e *Floresta Geradora Máxima*. Vários experimentos computacionais foram realizados, incluindo duas regras de atualização do parâmetro de centragem, três pontos iniciais e critérios de parada no Gradiente Conjugado, entre outros. Apresentamos ainda a caracterização da base de Multifluxo, uma heurística para a obtenção de uma base ótima a partir de uma solução interior "quase-ótima" fornecida pelo Método Primal-Dual e um estudo sobre a degenerescência.

Abstract

The *Multicommodity Flow Problem* is a model of Mathematical Programming defined on a network that has many important applications. In this work, we perform a computational study of a Primal-Dual Interior Point Method applied to this problem. We solve the linear system of iterate displacements using the Conjugate Gradient Method with a combination of the preconditioners *Diagonal* and *Maximum Spanning Forest*. Several computational experiments were performed, considering different starting points, different rules of the centering parameter update and different stopping criterion for the Conjugate Gradient. We present a characterization of the Multiflow basis, a heuristic for constructing an optimal basis from an interior "quasi-optimal" solution given by the Primal-Dual Method as well as a study about degeneracy.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 O problema

O *problema de multifluxo (multicommodity flow problem)* é um modelo matemático de programação linear que considera o compartilhamento de diferentes produtos através de uma mesma rede de transporte. Problemas de multifluxo formam uma classe especial de problemas devido à sua estrutura *bloco angular* e ainda à sua característica de *rede*. Existem aplicações em sistemas de comunicações, sistemas de tráfego urbano, sistemas de produção/distribuição de vários produtos, sistemas de rodovias, logísticas militares e várias outras.

Considere que diferentes produtos devam ser transportados de determinados centros de produção (fontes ou origens) para determinados centros de consumo (destinos), através de uma rede de transporte. Existem restrições de conservação de fluxo para cada produto, além de outras restrições relativas às capacidades das vias de transporte, pois, diferentes produtos compartilhando uma mesma via geram competição e congestionamento. Na prática, o fluxo pode ser de bens, veículos, indivíduos, mensagens, documentos e outros, em redes de transporte, distribuição, comunicações, etc. A função objetivo mais comumente encontrada é a que minimiza o custo total (linear) de transporte.

A literatura apresenta alguns modelos de programação linear deste problema (Assad [5], Kennington [35], Jones, Lustig e outros [30]). Em comum, estas diferentes formulações consideram uma rede com um conjunto finito $\mathcal{N}$

de m nós $\{1, \cdots, m\}$ e um conjunto finito $\mathcal{A}$ de n arcos $\{e_j : j = 1, \cdots, n\}$. Cada arco e_j tem capacidade d_j, $j = 1, \cdots, n$. Supõe-se ainda p produtos, sendo x_j^k, c_j^k e u_j^k o fluxo, o custo unitário e o limitante superior de fluxo, respectivamente, do produto k sobre o arco e_j, $k = 1, \cdots, p$, $j = 1, \cdots, n$ e b_i^k a oferta/ demanda do produto k no nó i, $k = 1, \cdots, p$ e $i = 1, \cdots, m$.

Associaremos o índice i a um nó, com $i = 1, \cdots, m$; o índice j a um arco, com $j = 1, \cdots, n$ e o índice k a um produto, com $k = 1, \cdots, p$.

Assim, definimos:
m : número de nós da rede;
n : número de arcos da rede;
p : número de produtos;
A : $m \times n$ matriz de incidência nó-arco da rede;
x^k : vetor de fluxos do produto k;
c^k : vetor de custos unitários do produto k;
u^k : vetor de limitantes superiores de fluxo do produto k;
b^k : vetor de ofertas/demandas do produto k;
d : vetor de capacidade dos arcos,
onde:
$x^k,\ c^k,\ u^k \in \mathbb{R}^n;\ \ b^k \in \mathbb{R}^m,\ k = 1, \cdots, p\ ;\ \ d \in \mathbb{R}^n;\ \ A \in \mathbb{R}^{m \times n}$.

O modelo mais empregado é o do *problema de multifluxo de custo mínimo*, onde se quer determinar o fluxo ótimo x^k de cada produto e que tem por objetivo minimizar o custo total de transporte, satisfazendo as restrições de (i) conservação de fluxo nos nós para cada produto; (ii) capacidade dos arcos da rede (restrições de acoplamento) e (iii) limitantes inferior e superior de fluxo em cada produto. É natural considerar nulo o limitante inferior de fluxo. De qualquer forma, mesmo quando este limitante não é nulo, é possível utilizar uma mudança de variáveis que o torna nulo.

Assim, o problema de multifluxo pode ser formulado por:

$$\min \ \sum_{k=1}^{p} (c^k)' x^k$$

$$s.a. \ \begin{cases} Ax^k = b^k, \ \ k = 1, \ldots, p \\ \sum_{k=1}^{p} x^k \leq d \\ 0 \leq x^k \leq u^k, \ \ k = 1, \ldots, p \end{cases} \tag{1.1}$$

Neste modelo, o primeiro conjunto de restrições, $Ax^k = b^k$, nos fornece as equações de *conservação de fluxo* em cada nó da rede, para cada produto. Como a mesma rede é compartilhada por todos os produtos, a matriz de coeficientes A se repete em cada um deles. Esta é a matriz de incidência da rede. Associamos, à cada linha de A, um nó da rede e, à cada coluna de A, um arco. A matriz A é uma matriz esparsa, uma vez que cada coluna possui apenas dois elementos não nulos: $+1$ na linha correspondente ao nó origem do arco e -1 na linha correspondente ao nó destino do arco.

O segundo conjunto de restrições é chamado de *restrições de acoplamento*. Sem estas restrições, o problema se reduz a p problemas de *fluxo de custo mínimo* independentes (desacoplados), que podem ser resolvidos por algoritmos específicos. De fato, estas restrições é que tornam o problema difícil: como produtos individuais compartilham arcos capacitados comuns, devemos encontrar uma maneira ótima de repartir estes arcos, de modo que as outras restrições também sejam satisfeitas.

O último grupo de restrições se refere às canalizações dos fluxos, próprias para cada produto. Algumas delas (ou todas) poderão não existir ($u_j^k = \infty$) .

Este modelo é um programa linear, com a particularidade de incluir um modelo de redes. Entretanto, é freqüente o uso de outras funções objetivos que conduzem a problemas não lineares que são mais difíceis de resolver.

Outro modelo muito comum é denominado *problema de multifluxo de fluxo máximo*, onde se quer encontrar fluxos não negativos que maximizem o fluxo total dos produtos. Se f^k é o fluxo do produto k, então a função objetivo do problema será:

$$\max \sum_{k=1}^{p} f^k$$

Neste caso não existem os limitantes superiores u^k e cada produto possui uma única origem, digamos h^k e um único destino t^k, de modo que o lado

direito b^k, para todo i e k, fica definido por:

$$b_i^k = \begin{cases} f^k & , \text{ se } i = h^k ; \\ -f^k & , \text{ se } i = t^k ; \\ 0 & , \text{ caso contrário } . \end{cases}$$

Outros modelos também são utilizados. A rede pode ser apresentada na forma *nó-arco* como em (1.1), porque utiliza uma matriz de incidência nó-arco, ou *arco-caminho (arc-path)* (Kennington [35]). Os recursos podem ser definidos na forma (Jones, Lustig e outros [30]) (i) *origem-destino*: cada produto vai de uma origem específica a um destino específico; (ii) *destino específico*: cada produto vai de uma origem específica para vários destinos ou de um destino específico para várias origens; (iii) *produto específico* : cada produto pode sair de várias origens indo para vários destinos. Esta última é a formulação mais usual, como em (1.1). Nestes três tipos, o que varia de modo significativo é o número de elementos diferentes de zero em cada b^k : o tipo (i), por exemplo, possui vetores b^k muito mais esparsos do que os outros dois tipos (apenas dois elementos não nulos por produto). Jones, Lustig e outros [30] mostram a influência destes três tipos de formulação em algoritmos de decomposição.

Além disto, Kennington [35] e Ahuja [3] observam que podem ser usados *pesos* nos fluxos de cada produto. Por exemplo, os fluxos podem estar sendo transformados em volume, e existem restrições sobre o volume total transportado. Chamando de q_j^k a quantidade da capacidade d_j consumida por uma unidade de fluxo do produto k sobre o arco e_j, as restrições de acoplamento em (1.1) são substituídas por

$$\sum_{k=1}^{p} q_j^k x_j^k \leq d_j, \quad j = 1, \cdots, n \quad .$$

Em nosso estudo, chamaremos o problema (1.1) apenas de *problema de multifluxo*, ao invés de *problema de multifluxo de custo mínimo*. Este problema possui uma grande variedade de aplicações, como os exemplos descritos a seguir:

• **Roteamento de múltiplos produtos**: nesta aplicação, distinguimos os produtos porque eles são bens físicos diferentes (por exemplo, diferentes produtos manufaturados) ou porque eles possuem diferentes pontos de origem e

destino, definidos por pares de nós distintos na rede (por exemplo, mensagens numa rede, sistemas de comunicação ou sistemas de transporte/distribuição).

• **Redes de comunicação**: aqui, os nós representam as estações de origem e destino das mensagens e os arcos representam as linhas de transmissão. Mensagens entre diferentes pares de nós definem produtos diferentes; a oferta e a demanda de cada produto é o número de mensagens a serem enviadas entre os nós origem e destino deste produto. Cada linha de transmissão tem uma capacidade fixa (em algumas aplicações, a capacidade de cada arco é fixa; em outras, podemos aumentar a capacidade com um certo custo/unidade).

• **Redes de computadores**: numa rede de computadores, os nós representam terminais, computadores, servidores ou unidades de armazenamento temporário e as demandas correspondem à razão de transmissão de dados entre os nós. As capacidades das linhas de transmissão definem as restrições de acoplamento.

• **Redes de transporte ferroviário**: nestas aplicações, os nós representam depósitos ou estações de trens e os arcos representam os trilhos entre eles. A demanda pode ser medida pelo número de vagões (ou alguma medida de peso) que compõem um trem. Como o sistema possui diferentes custos para diferentes bens, podemos dividir a demanda do tráfego em diferentes classes. Cada produto nesta rede corresponde a uma classe particular de demanda entre um certo par origem-destino. A capacidade de cada arco do acoplamento é o número de vagões que podemos colocar nos trens, os quais são programados para serem despachados sobre um trilho (arco), em algum período de tempo. O objetivo é encontrar a demanda de vagões que minimiza o custo da operação.

• **Redes de distribuição**: em sistemas de distribuição e planejamento, desejamos distribuir vários produtos de determinadas fábricas para certos centros de consumo, usando, por exemplo, uma frota de caminhões e vários armazéns. As restrições ficam definidas pelas capacidades das fábricas, dos armazéns, dos caminhões e das estradas. Neste caso, tanto os nós (fábricas e armazéns) como as vias de transporte dividem capacidades mútuas.

• **Redes de importação/exportação de grãos alimentícios**: novamente os nós da rede correspondem às diversas localizações geográficas em diferentes países e os arcos correspondem às vias marítimas, ou estradas. Entre estas

localizações, os produtos são os vários tipos de grãos. As capacidades dos portos podem definir as restrições de acoplamento.

1.2 Técnicas especiais de solução

Alguns resultados e propriedades do problema (1.1) são apresentados por Kennington [35]. Diversos resultados especiais para problemas com dois ou três produtos não podem ser generalizados para um número qualquer de produtos, assim como problemas em redes especiais com, no máximo, duas origens e dois destinos (os outros nós são de transbordo, isto é, $b_i^k = 0$) também não podem ser generalizados para redes quaisquer. Em particular, se os dados do problema são inteiros, não se garante a existência de uma solução ótima inteira, uma vez que a matriz de restrições não é totalmente unimodular, em geral. Também não há garantia de que o valor do fluxo máximo seja igual ao valor do corte de capacidade mínima (resultados válidos para problemas de fluxo de custo mínimo com um único produto).

Problemas de multifluxo são problemas de programação linear e como tais podem ser resolvidos pelo método Simplex. No entanto, problemas reais de multifluxo são frequentemente de tamanho tão grande que a aplicação do método Simplex se torna proibitiva. As restrições de acoplamento tornam este problema difícil. Algumas técnicas especiais para este tipo de problema já foram bastante estudadas, e nelas se tira proveito da estrutura angular (bloco diagonal) da matriz de restrições.

Estas técnicas se dividem em dois grandes grupos: *técnicas de decomposição* e *técnicas de partição*, que são brevemente discutidas a seguir. Grande parte delas foi desenvolvida na década de 60, e a maioria dos trabalhos que citamos aqui, para efeito de realizar um estudo completo, foi desenvolvida nas décadas de 60 e 70.

1.2.1 Técnicas de decomposição

Dentre as técnicas mais conhecidas de decomposição, específicas para o problema de multifluxo, podemos citar:

(1) **decomposição direcionada para preços** [3, 5, 7, 14, 35, 37, 41], também conhecida por *decomposição de Dantzig-Wolfe*, onde a coordenação entre o problema mestre e os subproblemas é feita atualizando a função objetivo (preços) dos subproblemas até que os preços ótimos (variáveis duais)

sejam encontrados. Em cada iteração, são resolvidos p subproblemas de fluxo de custo mínimo, um para cada produto, e o problema mestre tem "poucas" restrições $(n + p)$, mas muitas colunas. Ford e Fulkerson [15] foram os primeiros a empregar esta técnica ao problema de multifluxo. Em seu artigo original, é apresentada a técnica de *geração de colunas* para um problema de multifluxo de fluxo máximo. Este trabalho inspirou a tão conhecida *técnica de decomposição de Dantzig-Wolfe* [14]. Tomlim [69] estendeu a técnica de geração de colunas de Ford e Fulkerson para o problema de multifluxo de custo mínimo e desenvolveu o primeiro programa [70] para este problema. Supondo $u_j^k = \infty$, $k = 1 \cdots, p$, $j = 1, \cdots, n$, nos subproblemas, ele mostrou que os subproblemas podem ser resolvidos por algoritmos de *caminho mínimo*. Swoveland, em sua tese de doutorado [67], apresenta uma discussão rigorosa e detalhada do *princípio da decomposição*. Outros estudos sobre esta técnica e/ou sua programação também foram apresentados por Jarvis [27], Cremeans, Smith and Tyndall [13], Swoveland [67, 68], Weigel e Cremeans [72], Wollmer [73], Jarvis e Keith [28], Chen e Dewald [12], Jarvis e Martinez [29] e Helgason [25].

(2) **decomposição direcionada para recursos** [3, 17, 35, 37, 41], ou *decomposição por alocação do lado direito (RHS)*, onde a idéia é distribuir a capacidade dos arcos entre os produtos, de tal forma que as soluções dos p *subproblemas* produzam uma solução para o problema acoplado. Em cada iteração, é feita uma alocação dos recursos disponíveis e são resolvidos p subproblemas; esta alocação é tal que o fluxo combinado dos subproblemas resulta num fluxo factível para o problema original. Robacker [64] foi o primeiro a sugerir este procedimento para problemas de multifluxo. É comum o uso de penalização do tipo relaxação lagrangeana na utilização desta técnica. As diferentes técnicas da decomposição direcionada para recursos aparecem dependendo da maneira como é resolvido o problema mestre, que é um problema convexo com restrições lineares. Dentre elas, as mais utilizadas são:

• **método da aproximação tangencial**, onde o problema mestre possui um número muito grande de restrições (que são geradas quando necessário) e os subproblemas são problemas de fluxo de custo mínimo de um único produto que têm apenas os limitantes superiores das variáveis alterados em cada iteração, o que permite a utilização do método *dual Simplex para redes*. Geoffrion discute este procedimento em [17] e experimentos computacionais podem ser encontrados em Swoveland [68] e Geoffrion e Graves [18].

• **método das direções factíveis**, técnica que também pode ser usada, já que o problema mestre é um problema convexo com restições lineares; está bem discutida em Lasdon [41]. É construído um problema para encontrar a melhor direção na qual se pode melhorar uma alocação, até se chegar à alocação ótima.

• **otimização com subgradiente**, uma alternativa ao método das direções factíveis, onde, em vez de procurar a *melhor* direção como anteriormente, o que torna o procedimento computacionalmente caro, utiliza-se algum subgradiente para definir a direção de movimento. Se não houver preocupação com o tamanho do passo, a alocação pode ser infactível, tornando-se necessário fazer uma projeção para se retornar à região factível. Held, Wolfe e Crowder [24] propuseram esta técnica para o problema de multifluxo de fluxo máximo. Kennington e Shalaby [38] estenderam este trabalho para o problema de multifluxo de custo mínimo, considerando $u_j^k = \infty$, para todo j e k, apresentando experimentos computacionais.

1.2.2 Técnicas de partição

As *técnicas de partição* são técnicas associadas ao método Simplex, no sentido de que a base corrente é particionada e sua estrutura especial é explorada. Nos problemas de multifluxo, a motivação é explorar a estrutura de rede existente no problema. Dentre as *técnicas de partição*, podemos citar:

(1) **método de Rosen** [41, 65], um método do tipo dual, que particiona as variáveis dos blocos em *dependentes* e *independentes* e, posteriormente, estende esta partição às restrições de acoplamento, sendo portanto um método de *partição*. Isto assegura um problema mestre pequeno, chamado de *problema reduzido*, o qual será formado pelas variáveis independentes, sendo então esquecidas as canalizações sobre as variáveis dependentes — por isso, este é também um método de *relaxação*. Este método gera uma sequência de soluções básicas infactíveis para o problema original, correspondendo a uma sequência de soluções factíveis para o seu dual. A factibilidade só é alcançada quando o ótimo é encontrado. Em cada iteração são resolvidos p subproblemas de fluxo de custo mínimo de um único produto, que têm apenas sua função objetivo modificada. O problema reduzido é construído de tal forma que permite a aplicação do método *dual Simplex* com variáveis canalizadas. Lasdon [41] propõe a resolução do problema reduzido pelo método *dual Sim-*

plex, pois isto permite que a base inicial deste problema seja obtida da base final da iteração anterior, através de atualizações.

(2) **partição primal** [3, 5, 35, 37, 41], que é uma especialização do método Simplex primal para o problema do multifluxo. Esta especialização envolve uma partição da *base* do problema, de modo a explorar sua estrutura. Para isso, é necessário fazer a caracterização da base do problema do multifluxo — ver Seção 2 do Capítulo 2 — e então, este método segue os mesmos passos que o método Simplex (cálculo dos custos relativos, teste da razão, atualização da base e das variáveis) mas, todos os cálculos que envolvem a matriz básica são feitos sobre uma matriz de trabalho, chamada *base de trabalho* ou *matriz de ciclo*, quadrada, "pequena", que pode ser gerada a partir das bases dos blocos. Kennington [37] apresenta detalhadamente uma maneira eficiente de se implementar este algoritmo, incluindo a atualização da inversa da matriz de trabalho.

Finalmente cabe dizer que, sendo a *partição primal* uma especialização do método Simplex Revisado feita para problemas de multifluxo, ela se encontra dentro de técnicas um pouco mais gerais, desenvolvidas para problemas com estrutura bloco angular. Técnicas deste tipo, como por exemplo *"compact inverse methods"* ou *"Generalized Upper Bounding"* (GUB) estão descritas em Lasdon [41]. As idéias básicas da *partição primal* se encontram em Bennett [8], Kaul [34], Saigal [66] e Hartman e Lasdon [22, 23]. Estas idéias foram mais tarde desenvolvidas e adaptadas para problemas de multifluxo por Maier [45], Kennington [36], Helgason e Kennington [26], Ali, Helgason, Kennington e Lall [4] e McCallum [48]. Grigoriadis e White [21] desenvolveram a técnica de partição dual.

Concluindo, experimentos computacionais (Kennington [35], Assad [5]) mostraram que técnicas de decomposição têm convergência lenta, sendo que a decomposição por recursos é ainda mais lenta do que a decomposição por preços. As técnicas de partição tiveram um desempenho melhor. Além disso, como a partição primal e a decomposição por preços são baseadas no algoritmo Simplex, estas duas técnicas podem ser chamadas de exatas. Isto não acontece com a decomposição por recursos, que não possui um critério de otimalidade efetivo e, consequentemente, pode ser vista como uma heurística. Ali e outros [4] também mostraram que a decomposição por recursos é mais lenta do que a decomposição por preços e a partição primal.

Segundo Castro [10], nenhuma implementação da partição primal foi

ainda capaz de resolver problemas grandes num tempo significativamente melhor do que os códigos Simplex para problemas gerais de programação linear. Por exemplo, o pacote PPRN [11], em média, se equiparou com o MINOS 5.3, ou foi pouco melhor. Estudos e implementações mais recentes da decomposição por preços, como por exemplo a que foi desenvolvida por Frangioni [16], e que usam centros analíticos e função de penalidade mostraram que, apesar destas novas variantes, para algumas classes de problemas — com redes grandes e um número não muito grande de produtos — esta técnica apresentou um desempenho pior do que a implementação de Castro.

Atualmente, a crescente utilização de métodos de pontos interiores na resolução de problemas gerais de programação linear fez com que estes métodos também fossem utilizados em problemas de fluxo em redes, particularmente em problemas de multifluxo (Castro [10] e Li e Lustig [42]).

De fato, a melhor complexidade conhecida para problemas de multifluxo, foi obtida com algoritmos de pontos interiores por Kapoor e Vaidya [32]: $\mathcal{O}(p^{3.5}m^{2.5}nL)$ operações, onde cada operação é efetuada com precisão de $\mathcal{O}(L)$ bits. Resende e Pardalos [59] citam também Kamath e Palmon [31], que, também com algoritmos de pontos interiores, encontram a solução exata, numa primeira variante (onde é usado o método dos gradientes conjugados), em $\mathcal{O}(p^{2.5}mn^{1.5}\bar{l}\log(n\bar{d}\bar{u}))$, onde $\bar{l} = pm + n$, $\bar{d}$ é a maior demanda e $\bar{u}$ é a maior capacidade. Uma segunda variante tem complexidade de $\mathcal{O}(p^{0.5}n^3 + pm^{1.5}n^{1.5})\bar{l}\log(m\bar{d}\bar{u}))$ e pode ser melhorada para $\mathcal{O}(p^{0.5}n^{2.7} + pm^{1.2}n^{1.5} + pn^{2.5})\bar{l}\log(m\bar{d}\bar{u}))$ usando "multiplicação rápida de matrizes". Esta variante usa inversa de matrizes e atualizações de posto um.

Neste trabalho, temos três **objetivos** básicos:

- implementar e testar um algoritmo primal-dual de pontos interiores aplicado ao problema de multifluxo, onde são realizados vários experimentos computacionais, explorando sugestões da literatura e também apresentando outros refinamentos;

- apresentar uma heurística para a obtenção de uma base ótima através de uma solução "quase-ótima" fornecida pelo método primal-dual de pontos interiores;

- fazer um estudo sobre a degenerescência e falta de pontos interiores no problema de multifluxo.

No Capítulo 2, apresentamos a caracterização da base do problema. No Capítulo 3, aplicamos o método primal-dual de pontos interiores, explorando a estrutura particular deste problema. Como o maior esforço computacional deste método está na resolução do sistema linear das direções, destacamos este item no Capítulo 4. No Capítulo 5, apresentamos uma heurística para a obtenção de uma base ótima para o problema, a partir de uma solução do método primal-dual. Os experimentos computacionais estão descritos no Capítulo 6. O Capítulo 7 apresenta um estudo sobre a degenerescência. Finalmente, o Capítulo 8 apresenta as conclusões do trabalho.

Capítulo 2

Caracterização da Base de Multifluxo

Apresentamos aqui a caracterização da *base* do problema (1.1), ainda em sua forma original, levando em consideração a estrutura bloco angular de sua matriz dos coeficientes, que pode ser observada abaixo e também a sua estrutura de rede para cada produto:

$$\begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \\ I & I & \cdots & I \end{bmatrix}$$

Antes de caracterizarmos a base de multifluxo propriamente, colocaremos algumas definições e resultados relativos à base de fluxo de um único produto.

2.1 Base de fluxo

Vamos supor que a rede $G = (\mathcal{N}, \mathcal{A})$ é conectada e está representada pela matriz de incidência nó-arco A. É bem conhecido que a matriz A é *totalmente unimodular*, isto é, toda submatriz de A tem determinante $+1$,

-1 ou zero. Cada linha de A corresponde a um nó da rede, assim como cada coluna de A corresponde a um arco.

Usaremos, indistintamente, a seguinte notação: o t-ésimo arco de G, que possui sua origem no nó i e seu destino no nó j, será denotado por e_t ou por (i,j). A coluna correspondente na matriz A será a coluna a^{ij}, que possui exatamente dois elementos não nulos: $+1$ na linha i e -1 na linha j.

Uma *cadeia* em G é uma seqüência alternada de nós e arcos adjacentes dois a dois, sem repetição de nós. Um *caminho* é uma cadeia cujos arcos têm a mesma orientação. Um *circuito* em G é um caminho fechado, isto é, os nós inicial e final são idênticos; um *ciclo* é uma cadeia fechada. Uma rede é *conectada* se existe uma cadeia ligando quaisquer pares de seus nós. Caso contrário, ela é desconectada.

O *grau* de um nó é o número de arcos que incidem nele. Um nó é denominado *folha* quando o seu grau é 1.

Uma *árvore geradora* B de G é uma árvore, isto é, um grafo conectado e sem ciclos, e que inclui todos os nós de G. Podemos afirmar que a árvore geradora de um grafo com m nós possui $m-1$ arcos. Pode-se provar que cada *base* B da rede G é uma árvore geradora de G. Como posto$(A) = m-1$, então a dimensão de qualquer base de G é $m-1$. Vamos retirar uma linha qualquer de A, e vamos denominar de *raiz* da árvore geradora o nó correspondente à linha que foi retirada de A. Assim, estaremos trabalhando com uma matriz de posto linha completo. Usaremos, indistintamente, o símbolo B para denotar uma matriz básica, ou uma árvore geradora ou ainda para denotar índices de colunas básicas.

O subconjunto de colunas definido por um ciclo é linearmente dependente.

Uma *floresta* é um grafo que não contém ciclos (geralmente desconectado). Cada componente conectada de uma floresta é uma árvore. Se uma floresta possui f componentes, a matriz de incidência associada à ela tem posto $m - f$. Matrizes de incidência com posto menor que $m - 1$ correspondem à redes desconectadas. Uma *floresta geradora* de G é uma floresta que contém o mesmo número de nós e de componentes que a rede original G.

Um resultado bastante conhecido é que qualquer base B pode ser colocada na forma *triangular (inferior)*. Para isso, basta reordenar linhas (nós) e/ou colunas (arcos) de B, partindo de cada nó folha e tomando o arco incidente nele, até chegar ao nó raiz. Este "procedimento" significa *percorrer a árvore no sentido folhas → raiz*. Utilizamos três vetores para realizar este

procedimento: o *predecessor* de um nó, o *fio* e o *fio reverso* (ver Seção 2.1.1).

Como exemplo, considere a rede G com $m = 4$ nós e $n = 6$ arcos dada pela Figura 2.1. Os arcos são dados por $\{e_1, \cdots, e_6\}$. Retirando a linha 4 de A e supondo a raiz no nó 4, podemos tomar, como exemplo de base B de G, a matriz formada pelas colunas correspondentes aos arcos e_1, e_4, e_3 , conforme a Figura 2.1. Usaremos a notação $B = (e_1, e_4, e_3)$. Além disso, nas figuras onde aparecem árvores geradoras, o índice do nó raiz sempre aparecerá retirado.

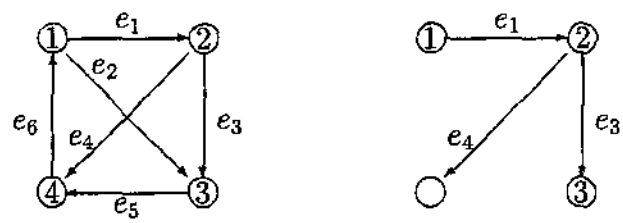


Figura 2.1: *rede exemplo 1 e uma base*

Neste caso, a base original é dada pela matriz

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_4 & e_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \end{array}$$

onde a numeração vertical se refere à ordem dos nós e os arcos são dados na ordem em que aparecem na base.

Podemos organizar esta base numa forma triangular inferior, através de permutações de arcos e nós, como pode ser visto a seguir:

$$\begin{array}{c} \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} e_3 & e_1 & e_4 \end{array} \\ \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix} \end{array}$$

Assim, se x_B é o vetor das variáveis básicas e b é o lado direito, o sistema linear básico, $Bx_B = b$, pode ser resolvido com operações de soma e subtração apenas, ao percorrer a árvore B no sentido *das folhas para a raiz.*

Por exemplo, se $b = (\ 3,\ \ 2,\ \ -1\ )'$, o sistema linear $Bx_B = b$ é:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_4 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -1 \end{bmatrix}$$

o qual, reorganizado de maneira conveniente, corresponde ao sistema triangular inferior equivalente:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_3 \\ x_1 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ 2 \end{bmatrix}$$

cuja solução é

$$\begin{bmatrix} x_1 \\ x_4 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}$$

Seja c_B o vetor de custos associado à base B. Se y é o vetor das variáveis duais associado às equações de conservação de fluxo $Ax = b$, podemos obter y a partir de

$$y' = c'_B B^{-1} \Leftrightarrow y'B = c'_B \Leftrightarrow B'y = c_B$$

onde c_B e y são vetores coluna do $\mathbb{R}^m$.

O sistema linear $B'y = c_B$ também pode ser resolvido com operações de soma e subtração apenas, ao percorrer a árvore básica B no sentido *da raiz para as folhas.*

Por exemplo, se os custos das variáveis básicas são $c_B = (\ 1,\ \ 5,\ \ 3\ )'$, o sistema linear $B'y = c_B$ é:

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 3 \end{bmatrix}$$

Reorganizando convenientemente o sistema anterior, temos o sistema linear equivalente triangular superior:

$$\begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_3 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix}$$

cuja solução é

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \\ 2 \end{bmatrix}$$

Uma coluna não básica de A pode ser expressa em função das colunas básicas sob a forma de um *cadeia.* Se a^{ij} é uma coluna original de A correspondendo ao arco não básico e_t, então a^{ij} pode ser representada através de uma combinação linear de colunas de B. O vetor coluna $\hat{a}^{ij}$ conterá os elementos desta combinação linear, isto é,

$$B\hat{a}^{ij} = a^{ij} \Rightarrow \hat{a}^{ij} = B^{-1}a^{ij}$$

Também é bastante conhecido que $\hat{a}^{ij} \in \{-1,\ 0,\ 1\}$ e $\hat{a}^{ij}$ pode ser obtido através de uma cadeia em B que liga o nó i ao nó j. Esta cadeia, mais o arco e_t, formam um único ciclo em B. Usaremos a seguinte convenção:
a orientação do ciclo é dada pelo arco e_t. Se um arco do ciclo concorda com a orientação, a componente correspondente em $\hat{a}^{ij}$ é -1; caso contrário é $+1$. Se um arco de B não está no ciclo, a componente correspondente em $\hat{a}^{ij}$ é zero.

Tomando o mesmo exemplo de G e B, considere o arco não básico $e_5 = (3, 4)$. Lembrando que $B = (e_1,\ e_4,\ e_3)$, nesta ordem, obtemos $\hat{a}^{34} = (\ 0,\ \ 1,\ \ -1\ )'$, como podemos observar na Figura 2.2.

Por último, chamamos de *custos relativos* aos coeficientes

$$\bar{c}_{ij} = c_{ij} - c'_B B^{-1} a^{ij}$$

que podem ser calculados por $\bar{c}_{ij} = c_{ij} - y' a^{ij}$ ou por $\bar{c}_{ij} = c_{ij} - c'_B \hat{a}^{ij}$.

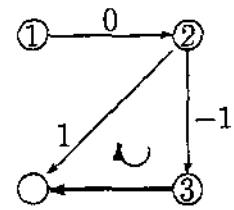


Figura 2.2: *exemplo de ciclo*

2.1.1 Estrutura de dados para árvore geradora

A estrutura de dados que utilizamos é a sugerida por Kennington e Helgason [37], que utiliza apenas vetores unidimensionais com m ou n elementos. Os arcos da rede estão numerados de 1 até n e os nós, de 1 até m.

Dados da rede: A, b, c, u

A: matriz de incidência nó-arco, armazenada em dois vetores inteiros:
$orig[1..n]$: origem de cada arco;
$dest[1..n]$: destino de cada arco.

$b[1..m]$: ofertas/demandas nos nós; vetor real.

$c[1..n]$: custos unitários dos arcos; vetor real.

$u[1..n]$: limitantes superiores de fluxos; vetor real.

Solução: x, y

$x[1..n]$: fluxos nos arcos; vetor real.

$y[1..m]$: preços nos nós; vetor real.

Árvore geradora:

A árvore geradora com raiz (base) é armazenada através de três vetores inteiros:

pred[1..*m*] : predecessor do nó; indica qual arco precede este nó quando percorremos a árvore no sentido da *folhas* para a *raiz*. O predecessor do nó *raiz* é marcado com zero.

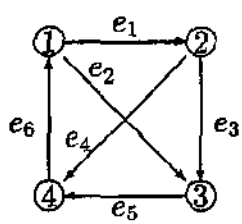
fio[1..*m*] : o *fio direto* indica a seqüência de nós utilizada para percorrer a árvore no sentido da *raiz* para as *folhas* (busca em profundidade).

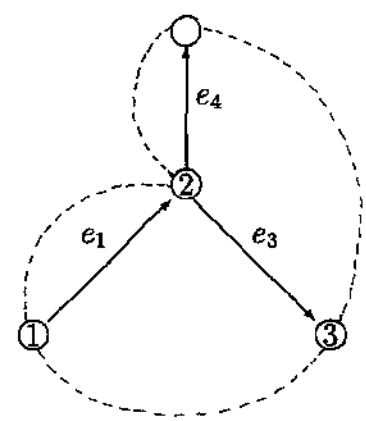
rev[1..*m*] : o *fio reverso* indica a seqüência de nós utilizada para percorrer a árvore no sentido das *folhas* para a *raiz*, e é definido na ordem inversa do fio.

Com o vetor *pred*, temos a informação de quais são os arcos básicos e em que ordem eles aparecem. Combinando *pred* e *rev*, conseguimos organizar uma árvore geradora com raiz de modo que a matriz B correspondente se transforma numa matriz triangular inferior. Similarmente, combinando *pred* e *fio*, conseguimos uma matriz triangular superior.

Exemplo:

Considerando a rede G e a base B da Figura 2.1, temos, respectivamente, a seguinte representação para a matriz A e para a árvore geradora com raiz:





arco	1	2	3	4	5	6
orig	1	1	2	2	3	4
dest	2	3	3	4	4	1

nó	1	2	3	4
pred	1	4	3	0
fio	3	1	4	2
rev	2	4	1	3

Assim, usando o predecessor e o fio reverso, temos a base na forma triangular inferior. Similarmente, com o predecessor e o fio, temos a base na forma triangular superior:

$$\begin{array}{c} \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} e_3 & e_1 & e_4 \end{array} \\ \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ e_3 \\ e_1 \\ e_4 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 1 & 2 \end{array} \\ \begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Na resolução de sistemas lineares que envolvem a matriz B, percorremos a árvore no sentido *folhas* $\rightarrow$ *raiz*: para isto, usamos *pred* e *rev*. Em sistemas lineares que envolvem a matriz B', percorremos a árvore no sentido *raiz* $\rightarrow$ *folhas*, usando *pred* e *fio*.

A seguir, apresentamos dois algoritmos básicos para sistemas lineares com B e B' , que utilizam da ordem de $\mathcal{O}(m)$ somas/subtrações apenas. Nestes algoritmos, *raiz* representa a raiz da árvore geradora; z é o vetor das variáveis e r é um lado direito conhecido. Ao final de cada um dos algoritmos, z armazena a solução do sistema linear.

Algoritmo 2.1 (*resolve* $Bz = r$)

```
for  i ← 1..m  do  z[i] ← r[i] ;
   no ← rev[raiz] ;
   while no ≠ raiz do
   begin
       arco ← pred[no] ;
       if no = orig[arco]
         then z[dest[arco]] ← z[dest[arco]] + z[no]
         else begin
                z[orig[arco]] ← z[orig[arco]] + z[no] ;
                z[no] ← −z[no] ;
              endelse;
       no ← rev[no] ;
   endwhile
```

Algoritmo 2.2 (*resolve* $B^t z = r$)

```
   for  i ← 1..m  do  z[i] ← r[i] ;
   z[raiz] ← 0 ;
   no ← fio[raiz] ;
   while no ≠ raiz do
   begin
       arco ← pred[no] ;
       if no = orig[arco]
         then z[no] ← z[no] + z[dest[arco]]
         else begin
                z[no] ← −z[no] ;
                z[no] ← z[no] + z[orig[arco]] ;
              endelse;
       no ← fio[no] ;
   endwhile
```

Note que deve ser respeitada a ordem dos arcos da base, dada por *pred*. Por exemplo, se queremos resolver $B^t z = c'_B$, onde o lado direito é formado por componentes básicas que são retiradas de um vetor do $\mathbb{R}^n$ $(m < n)$, a segunda linha do Algoritmo 2.2 deverá ser trocada por:

```
for  i ← 1..m  do
if pred[i] = 0 then z[i] ← 0
               else z[i] ← c[pred[i]] ;
```

2.2 Base de multifluxo

Vamos, agora, caracterizar a base de multifluxo. Como posto$(A) = m-1$ (supondo a rede conectada), o sistema linear de cada produto k, $Ax^k = b^k$, tem uma equação redundante, a qual será retirada. Além disso, adicionamos as variáveis de folga x^{p+1} às restrições de acoplamento, de maneira que a matriz dos coeficientes relativa ao problema (1.1) na sua forma padrão será:

$$\begin{bmatrix} A & 0 & \cdots & 0 & 0 \\ 0 & A & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \cdots & A & 0 \\ I & I & \cdots & I & I \end{bmatrix}$$

onde A, com um abuso de linguagem, agora é a matriz de incidência nó-arco com uma linha retirada (cada b^k terá então uma componente retirada: a componente correspondente à linha que foi retirada em A) e I é a matriz identidade de ordem n.

Qualquer base tem dimensão $\widehat{m} = (m-1)p + n$ e tem a forma:

$$\widehat{B} = \begin{bmatrix} [B^1|R^1] & 0 & \cdots & 0 & 0 \\ 0 & [B^2|R^2] & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \cdots & [B^p|R^p] & 0 \\ [E^1|F^1] & [E^2|F^2] & \cdots & [E^p|F^p] & E^{p+1} \end{bmatrix}$$

onde:
• B^k, $k = 1, ..., p$, são bases (árvores geradoras) de A, de dimensão $(m-1) \times (m-1)$. Qualquer uma das matrizes R^k, $k = 1, ..., p$, pode ser vazia.

• E^k e F^k seguem a mesma partição de $[B^k|R^k]$. E^k, F^k e E^{p+1} são matrizes formadas por colunas da matriz identidade de ordem n. A linha onde aparece o número 1 em E^k identifica qual arco foi usado em B^k, $k = 1, ..., p$; a linha onde aparece o número 1 em F^k identifica qual arco foi usado em R^k, $k = 1, ..., p$. As colunas de E^{p+1} identificam as variáveis de folga das restrições de acoplamento que estão na base.

Com estas partições, reescrevemos $\hat{B}$:

$$\hat{B} = \left[\begin{array}{cccc|cccc|c} B^1 & 0 & \cdots & 0 & R^1 & 0 & \cdots & 0 & 0 \\ 0 & B^2 & \cdots & 0 & 0 & R^2 & \cdots & 0 & 0 \\ & & \ddots & & & & \ddots & & \\ 0 & 0 & \cdots & B^p & 0 & 0 & \cdots & R^p & 0 \\ \hline E^1 & E^2 & \cdots & E^p & F^1 & F^2 & \cdots & F^p & E^{p+1} \end{array}\right]$$

Ou seja, $\hat{B}$ tem a estrutura geral:

$$\hat{B} = \left[\begin{array}{ccc} B & R & 0 \\ E & F & E^{p+1} \end{array}\right]$$

Vamos utilizar, para as soluções básicas primal e dual, a transformação (2.1), que nos permite trabalhar com os produtos separadamente, resolvendo sistemas triangulares. x_B e x_R são particionados por produtos: $x_B = (x_B^1 \ \cdots \ x_B^p)'$, $x_R = (x_R^1 \ \cdots \ x_R^p)'$. Cada x_B^k possui $m-1$ componentes; alguns (ou todos) x_R^k podem não existir e a soma do número de componentes de x_R e de x_B^{p+1} é n.

$$\left[\begin{array}{c} x_B \\ x_R \\ x_B^{p+1} \end{array}\right] = \left[\begin{array}{ccc} I & -B^{-1}R & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{array}\right] \left[\begin{array}{c} \bar{x}_B \\ \bar{x}_R \\ \bar{x}_B^{p+1} \end{array}\right] \qquad (2.1)$$

A conclusão que se segue pode ser encontrada em Bazaraa e outros [7] e em Kennington e Helgason [37]: *cada base para o problema de multifluxo é um conjunto de colunas linearmente independentes formado por três tipos de arcos: arcos que formam uma árvore geradora (com raiz) para cada produto,*

arcos que formam ciclos com suas respectivas árvores geradoras e arcos correspondentes às variáveis de folga das restrições de acoplamento. Os arcos das árvores geradoras totalizam $(m-1)p$ e os arcos dos dois últimos conjuntos totalizam n .

Observe que, definido um conjunto de colunas básicas, esta caracterização pode não ser única. Isto pode ser visto no exemplo a seguir.

Suponha um problema com $p = 2$ produtos, numa rede com $m = 4$ nós e $n = 4$ arcos, dada pela Figura 2.3. Os arcos são dados por $\{e_1,\ e_2,\ e_3,\ e_4\}$. A dimensão da base será $\widehat{m} = p(m-1) + n = 10$.

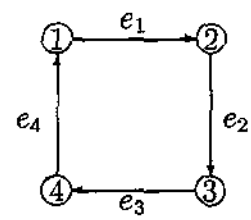


Figura 2.3: *rede exemplo 2*

Uma caracterização possível para a base é dada na Figura 2.4. Aqui, temos:

$B^1 = (e_1,\ e_2,\ e_4)$, $R^1 = (e_3)$, $B^2 = (e_2,\ e_3,\ e_4)$, $R^2 = (e_1)$ e $E^3 = (e_2,\ e_4)$ representa as folgas das restrições de acoplamento, ou seja, a base é formada por três conjuntos: < (1, 2, 4); (2, 3, 4); (2, 3, 1, 3) >, que são os índices, respectivamente, dos arcos na árvore do produto 1, do produto 2 e o terceiro vetor indica qual produto está associado a que arco (3 indica arco de folga).

Neste exemplo, há uma segunda caracterização para a base, dada pela Figura 2.5:

$B^1 = (e_2,\ e_3,\ e_4)$, $R^1 = (e_1)$, $B^2 = (e_1,\ e_2,\ e_4)$, $R^2 = (e_3)$ e $E^3 = (e_2,\ e_4)$ representa as folgas das restrições de acoplamento. Neste caso, a base é formada por < (2, 3, 4); (1, 2, 4); (1, 3, 2, 3) > .

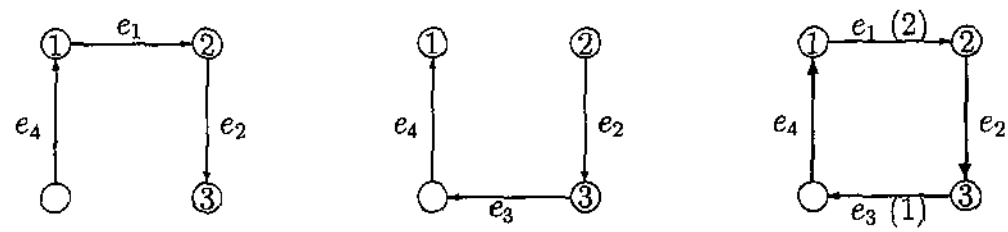


Figura 2.4: *primeira representação: produto 1, produto 2, acoplamento*

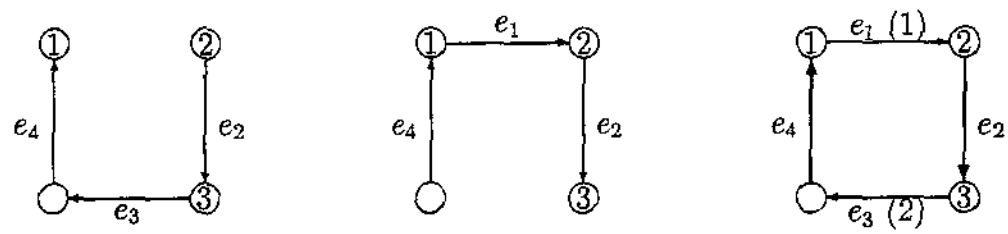


Figura 2.5: *segunda representação: produto 1, produto 2, acoplamento*

2.2.1 Solução básica primal

A solução básica primal é obtida resolvendo o sistema

$$\widehat{B}\widehat{x}_B = \widehat{b}$$

onde $\widehat{b} = (b \mid d)'$ é o lado direito do problema (1.1), com $b = (b^1 \ \cdots \ b^p)'$ (cada b^k possui $m-1$ componentes). Ou, equivalentemente,

$$\begin{bmatrix} B & R & 0 \\ E & F & E^{p+1} \end{bmatrix} \begin{bmatrix} x_B \\ x_R \\ x_B^{p+1} \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix} \tag{2.2}$$

Usando a transformação (2.1), obtemos :

$$\begin{bmatrix} B & R & 0 \\ E & F & E^{p+1} \end{bmatrix} \begin{bmatrix} I & -B^{-1}R & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \bar{x}_B \\ \bar{x}_R \\ \bar{x}_B^{p+1} \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$

ou:

$$\begin{bmatrix} B & 0 & 0 \\ E & F - EB^{-1}R & E^{p+1} \end{bmatrix} \begin{bmatrix} \bar{x}_B \\ \bar{x}_R \\ \bar{x}_B^{p+1} \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$

O primeiro conjunto de equações, $B\bar{x}_B = b$, é formado por p sistemas desacoplados $B^k x_B^k = b^k$ com matrizes triangulares de árvores geradoras.

O segundo conjunto de equações é:

$$\begin{bmatrix} \bar{F} & E^{p+1} \end{bmatrix} \begin{bmatrix} \bar{x}_R \\ \bar{x}_B^{p+1} \end{bmatrix} = [d - E\bar{x}_B] \tag{2.3}$$

onde $\bar{F} = F - EB^{-1}R$.

Lembramos que E^{p+1} é uma matriz cujas colunas correspondem às variáveis de folga na base. Uma coluna da matriz $\bar{F}$, correspondente ao produto k, será do tipo $e^{ij} - E^k(B^k)^{-1}a^{ij}$, onde e^{ij} é um vetor unitário com n componentes cuja posição do número 1 indica o número do arco que está sendo usado e a^{ij} é a coluna correspondente da matriz A.

Seja $\hat{a}^{ij} = (B^k)^{-1}a^{ij}$. Então $\hat{a}^{ij}$ corresponde a uma cadeia do nó i ao nó j através dos arcos da árvore geradora com raiz do produto k. Como E^k é uma matriz $n \times (m-1)$ cujas colunas indicam quais são os arcos básicos de B^k, então $E^k(B^k)^{-1}a^{ij}$ simplesmente expande o vetor $(B^k)^{-1}a^{ij}$ de m componentes num vetor de n componentes, assinalando zero aos coeficientes que correspondem aos arcos não básicos do produto k. Em outras palavras, $E^k(B^k)^{-1}a^{ij}$ é um vetor de n componentes correspondendo à cadeia do nó i ao nó j na árvore básica do produto k. Finalmente, $e^{ij} - E^k(B^k)^{-1}a^{ij}$ corresponde ao único ciclo formado quando o arco (i,j) é adicionado à árvore básica, mas com o sinal trocado. Assim, conhecendo B, é fácil formar a matriz $\bar{F} \in \mathbb{R}^{n \times n}$.

Resumindo, o sistema linear das variáveis básicas primais pode ser resolvido como segue:

(i) Resolver $B\bar{x}_B = b \Rightarrow \begin{cases} B^1\bar{x}_B^1 = b^1 \\ \vdots \\ B^p\bar{x}_B^p = b^p \end{cases}$

(ii) Montar a matriz $(\bar{F} \quad E^{p+1})$, isto é, para $k = 1, \cdots p$, calcular: $(B^k)^{-1}R^k$; $E^k\,[(B^k)^{-1}R^k]$; $\bar{F}^k = F^k - [E^k(B^k)^{-1}R^k]$.

(iii) Resolver o sistema linear $n \times n$ (2.3). Dadas as particularidades das matrizes $\bar{F}$ e E^{p+1}, este sistema pode ser reorganizado: se o número de variáveis de folga na base for n_f, então n_f variáveis são encontradas por substituição e resolvemos um sistema linear $(n - n_f) \times (n - n_f)$ para encontrar as variáveis restantes. De fato, quanto maior for n_f, mais fácil será a resolução deste sistema. $n_f = n$ significa que as restrições de acoplamento não estão, de fato, acoplando.

(iv) Correção das variáveis:
Usando a transformação dada por (2.1) temos que:

$$x_R = \bar{x}_R \Rightarrow \begin{cases} x_R^1 = \bar{x}_R^1 \\ \vdots \\ x_R^p = \bar{x}_R^p \end{cases}$$

$$x_B^{p+1} = \bar{x}_B^{p+1}$$

$x_B = \bar{x}_B - B^{-1}R\bar{x}_R = \bar{x}_B - B^{-1}Rx_R$,
ou:

$$\begin{cases} x_B^1 = \bar{x}_B^1 - (B^1)^{-1}R^1x_R^1 \\ \vdots \\ x_B^p = \bar{x}_B^p - (B^p)^{-1}R^px_R^p \end{cases}$$

e, para $k = 1, \cdots p$, x_B^k pode ser obtido por: $\begin{cases} r^k = R^kx_R^k \\ t^k = (B^k)^{-1}r^k \Leftrightarrow B^kt^k = r^k \\ x_B^k = \bar{x}_B^k - t^k \end{cases}$

2.2.2 Solução básica dual

Seja $\hat{y}$ o vetor das variáveis duais, que é decomposto em

$\widehat{y} = (y \mid y^{p+1})' = (y^1, \ y^2, \ \cdots, y^p \mid y^{p+1})'$
onde:

y^k : vetor das variáveis duais correspondentes às restrições de conservação de fluxo do produto k, $Ax^k = b^k$, $k = 1, \ \cdots, \ p$;

y^{p+1} : vetor das variáveis duais correspondentes às restrições de acoplamento.

y^k são vetores coluna de dimensão $m-1$, $k = 1, \ \cdots, \ p$ e y^{p+1} é um vetor coluna de dimensão n.

O vetor de custos segue a mesma partição feita anteriormente em B, R, E e F:

$$\widehat{c} = (c^1, \ \cdots, \ c^p \mid \ 0)' = (c_B \mid c_R \mid 0)' = (c_{B^1}, \ \cdots, \ c_{B^p} \mid c_{R^1}, \ \cdots, \ c_{R^p} \mid 0)'$$

onde cada c_B^k possui $m-1$ componentes e a soma do número de componentes de c_R e do vetor nulo é n.

Com estas partições, o sistema básico dual é do tipo:

$$\widehat{y}'\widehat{B} = \widehat{c}'$$

ou,

$$\begin{bmatrix} y' & (y^{p+1})' \end{bmatrix} \begin{bmatrix} B & R & 0 \\ E & F & E^{p+1} \end{bmatrix} = \begin{bmatrix} c_B' & c_R' & 0 \end{bmatrix}$$

Usando a mesma transformação dada em (2.1), temos:

$$\begin{bmatrix} y' & (y^{p+1})' \end{bmatrix} \begin{bmatrix} B & R & 0 \\ E & F & E^{p+1} \end{bmatrix} \begin{bmatrix} I & -B^{-1}R & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} = \begin{bmatrix} c_B' & c_R' & 0 \end{bmatrix} \begin{bmatrix} I & -B^{-1}R & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}$$

que produz:

$$\begin{bmatrix} y' & (y^{p+1})' \end{bmatrix} \begin{bmatrix} B & 0 & 0 \\ E & \bar{F} & E^{p+1} \end{bmatrix} = \begin{bmatrix} c_B' & c_R' - c_B'B^{-1}R & 0 \end{bmatrix}$$

Deste último sistema, temos três conjuntos de equações lineares:

- $(y^{p+1})'\ E^{p+1} = 0$;
- $(y^{p+1})'\ \bar{F} = c'_R - c'_B B^{-1} R$;
- $y'B + (y^{p+1})'E = c'_B$.

O primeiro deles é um sistema de n equações cujo número de incógnitas é o número de variáveis de folga primais das restrições de acoplamento que estão na base. Lembrando que cada coluna de E^{p+1} é uma coluna da matriz identidade de ordem n, este conjunto de equações apenas nos diz quais componentes de y^{p+1} são necessariamente nulas.

No segundo conjunto de equações temos a matriz dos coeficientes $\bar{F}$, que é particionada por produtos, isto é, $\bar{F}^k = F^k - E^k(B^k)^{-1}R^k$, $k = 1, \cdots, p$, e cuja construção já apresentamos na solução básica primal. O lado direito também é particionado por produtos:

$$\bar{c}' = c'_R - c'_B B^{-1} R = (c'_{R^1} - c'_{B^1}(B^1)^{-1}R^1 \mid \cdots \mid c'_{R^p} - c'_{B^p}(B^p)^{-1}R^p)$$

e, para $k = 1, \cdots, p$, cada componente $(\bar{c}^k)' = c'_{R^k} - c'_{B^k}(B^k)^{-1}R^k$ pode ser obtida separadamente, envolvendo a resolução de um sistema linear triangular superior.

Desta maneira, este segundo conjunto de equações,

$$(y^{p+1})'\ [\ \bar{F}^1,\ \cdots,\ \bar{F}^p\ ] = [(\bar{c}^1)',\ \cdots,\ (\bar{c}^p)']$$

nos fornecerá as componentes restantes de y^{p+1}, já que algumas componentes nulas foram apontadas no primeiro conjunto de equações. De fato, descontando estas componentes nulas, o sistema linear restante será quadrado e sua dimensão é a soma do número de colunas de cada R^k, $k = 1, \cdots, p$. Além disto, os coeficientes da matriz deste sistema linear são 1 ou -1 ou zero.

Encontrado y^{p+1}, o último conjunto de equações é $y'B = c'_B - (y^{p+1})'E$, que pode ser expresso por p sistemas triangulares desacoplados, cada um do tipo

$$(y^k)'B^k = c'_{B^k} - (y^{p+1})'E^k,\quad k = 1, \cdots, p\ ,$$

onde, em cada um deles, percorremos a árvore básica no sentido raiz $\rightarrow$ folhas.

Resumindo, o sistema linear das variáveis duais pode ser resolvido como segue:

(i) $(y^{p+1})'\ E^{p+1} = 0$: zerar as componentes de y^{p+1} que correspondem às variáveis de folga das restrições de acoplamento que estão na base.

(ii) Resolver $(y^{p+1})'\ \bar{F} = \bar{c}'$, isto é, para $k = 1, \cdots, p$, calcular:

• $(B^k)^{-1}$; $E^k[(B^k)^{-1}R^k]$ e $\bar{F}^k = F^k - [E^k(B^k)^{-1}R^k]$, como na solução primal.

• Cada lado direito $(\bar{c}^k)' = c'_{R^k} - c'_{B^k}(B^k)^{-1}R^k$ pode ser obtido por:

$$\begin{cases} (\bar{t}^k)' = c'_{B^k}(B^k)^{-1} \Leftrightarrow (\bar{t}^k)'B^k = c'_{B^k} \Leftrightarrow (B^k)'\bar{t}^k = c^k_B \\ (\bar{r}^k)' = (\bar{t}^k)'R^k \\ (\bar{c}^k)' = c'_{R^k} - (\bar{r}^k)' \end{cases}$$

• Eliminar as componentes de y^{p+1} que já foram zeradas em (i), de modo que o sistema linear resultante é quadrado e com dimensão igual à soma do número de colunas de R^k, $k = 1, \cdots, p$.

Com (i) e (ii), calculamos y^{p+1}.

(iii) Calcular $y = (y^1, \cdots, y^p)'$, isto é, resolver p sistemas lineares triangulares do tipo:

$$(y^k)'B^k = c'_{B^k} - (y^{p+1})'E^k, \quad k = 1, \cdots, p\ .$$

2.2.3 Exemplo

Vamos exemplificar, considerando um problema com $p = 2$ produtos, na mesma rede da Figura 2.1.

Considere os demais dados:

$$\text{produto 1:} \begin{cases} b^1 = (\ 1,\ \ 2,\ \ -4,\ \ 1\ )' \\ u^1 = (\ 7,\ \ 7,\ \ 7,\ \ 7,\ \ 7,\ \ 7\ )' \\ c^1 = (\ 2,\ \ 4,\ \ 1,\ \ 6,\ \ 4,\ \ 1\ )' \end{cases}$$

$$\text{produto 2:} \begin{cases} b^2 = (\ 5,\ \ 1,\ \ -1,\ \ -5\ )' \\ u^2 = (\ 6,\ \ 6,\ \ 6,\ \ 6,\ \ 6,\ \ 6\ )' \\ c^2 = (\ 7,\ \ 4,\ \ 5,\ \ 1,\ \ 3,\ \ 7\ )' \end{cases}$$

acoplamento: $d = (\ 9,\ \ 6,\ \ 9,\ \ 3,\ \ 7,\ \ 7)'$

A dimensão da base será $\widehat{m} = p(m-1) + n = 12$. De cada produto, será eliminada a restrição relativa ao nó 4.

Suponha que $B^1 = (e_6,\ e_1,\ e_3)$, $R^1 = (e_5,\ e_2)$, $B^2 = (e_2,\ e_4,\ e_5)$, $R^2 = (e_1)$ e que as variáveis de folga das restrições de acoplamento que estão na base são x_1^3, x_3^3, x_6^3.

Temos, então, a representação dada pela Figura 2.6, supondo a raiz de cada árvore no nó 4.

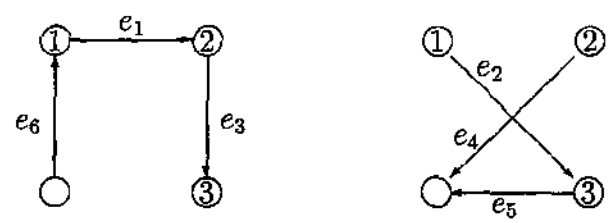


Figura 2.6: *árvores geradoras dos produtos 1 e 2*

A matriz básica $\widehat{B}$ correspondente é:

$$\widehat{B} = \begin{array}{c} \overbrace{}^{B^1} \overbrace{}^{B^2} \overbrace{}^{R^1} \overbrace{}^{R^2} \\ \left[\begin{array}{ccc|ccc|cc|ccc} -1 & 1 & & & & & & 1 & & & \\ & -1 & 1 & & & & & & & & \\ & & -1 & & & & 1 & -1 & & & \\ \hline & & & 1 & & & & & 1 & & \\ & & & & 1 & & & & -1 & & \\ & & & -1 & & 1 & & & & & \\ \hline & 1 & & & & & & & 1 & 1 & \\ & & & 1 & & & & 1 & & & \\ & & 1 & & & & & & & 1 & \\ & & & & 1 & & & & & & \\ & & & & & 1 & 1 & & & & \\ 1 & & & & & & & & & & 1 \end{array}\right] \\ \underbrace{}_{E^1} \underbrace{}_{E^2} \underbrace{}_{F^1} \underbrace{}_{F^2} \underbrace{}_{E^3} \end{array}$$

solução básica primal:

O sistema linear básico $\widehat{B}\bar{x}_B = \widehat{b}$, depois da mudança de variáveis, pode ser resolvido usando o sistema equivalente:

$$\left[\begin{array}{cc|ccc} B_1 & 0 & 0 & 0 & 0 \\ 0 & B_2 & 0 & 0 & 0 \\ \hline E^1 & E^2 & F^1 & F^2 & E^3 \end{array}\right] \left[\begin{array}{c} \bar{x}_B^1 \\ \bar{x}_B^2 \\ \hline \bar{x}_R \\ \bar{x}_B^3 \end{array}\right] = \left[\begin{array}{c} b^1 \\ b^2 \\ \hline d \end{array}\right]$$

Vamos supor todas as variáveis não básicas no limite inferior. Temos os seguinte passos:

(i) Resolver $B\bar{x}_B = b$:

produto 1:
$B^1\bar{x}_B^1 = b^1$; $\bar{x}_B^1 = (\bar{x}_6^1 \quad \bar{x}_1^1 \quad \bar{x}_3^1)'$

A matriz básica B^1 pode ser colocada na forma triangular. Para isto, basta percorrer a árvore geradora do produto 1 no sentido folhas $\rightarrow$ raiz, de onde se obtém:

$$\begin{array}{c} \\ 3 \\ 2 \\ 1 \end{array}\begin{array}{c} \begin{array}{ccc} e_3 & e_1 & e_6 \end{array} \\ \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \end{array}\begin{bmatrix} \bar{x}_3^1 \\ \bar{x}_1^1 \\ \bar{x}_6^1 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \\ 1 \end{bmatrix} \Rightarrow \bar{x}_B^1 = \begin{bmatrix} \bar{x}_3^1 \\ \bar{x}_1^1 \\ \bar{x}_6^1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$$

produto 2:

$B^2\bar{x}_B^2 = b^2$; $\bar{x}_B^2 = (\bar{x}_2^2 \quad \bar{x}_4^2 \quad \bar{x}_5^2)'$

Da mesma maneira, transformamos B^2:

$$\begin{array}{c} \\ 1 \\ 3 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} e_2 & e_5 & e_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}\begin{bmatrix} \bar{x}_2^2 \\ \bar{x}_5^2 \\ \bar{x}_4^2 \end{bmatrix} = \begin{bmatrix} 5 \\ -1 \\ 1 \end{bmatrix} \Rightarrow \bar{x}_B^2 = \begin{bmatrix} \bar{x}_2^2 \\ \bar{x}_5^2 \\ \bar{x}_4^2 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 1 \end{bmatrix}$$

(ii) Montar $\bar{F} = F - EB^{-1}R$:

produto 1: cálculo de $\bar{F}^1 = F^1 - E^1(B^1)^{-1}R^1$:

Para calcularmos $(B^1)^{-1}R^1$, devemos encontrar a cadeia do nó 3 ao nó 4, através dos arcos da árvore geradora com raiz do produto 1, e, em seguida, encontrar a cadeia do nó 1 ao nó 3, através dos arcos da árvore geradora com raiz do produto 1, como podemos observar na Figura 2.7.

Obtemos, então, as colunas de $(B^1)^{-1}R^1$: $(-1, \ -1, \ -1)'$ e $(0, \ 1, \ 1)'$.

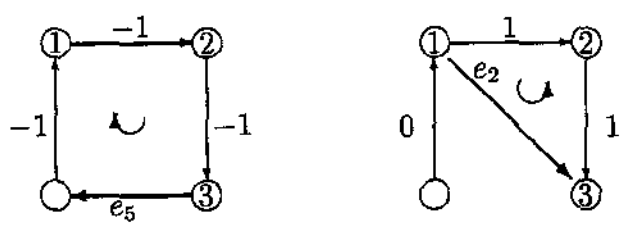


Figura 2.7: *ciclos do produto 1*

Assim,

$$E^1[(B^1)^{-1}R^1] = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & 0 \\ -1 & 1 \\ 0 & 0 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}$$

$$\bar{F}^1 = F^1 - [(E^1(B^1)^{-1}R^1)] = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} -1 & 1 \\ 0 & 0 \\ -1 & 1 \\ 0 & 0 \\ 0 & 0 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & -1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

produto 2: cálculo de $\bar{F}^2 = F^2 - E^2(B^2)^{-1}R^2$:
Novamente, para calcularmos $(B^2)^{-1}R^2$, devemos encontrar a cadeia do nó 1 ao nó 2, através dos arcos da árvore geradora com raiz do produto 2, como podemos observar na Figura 2.8.

Obtemos, então, $(B^2)^{-1}R^2 = (1, \ -1, \ 1)'$.

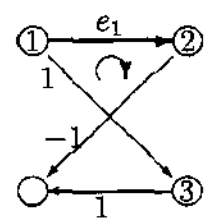


Figura 2.8: *ciclo do produto 2*

Assim,

$$E^2[(B^2)^{-1}R^2] = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}$$

$$\bar{F}^2 = F^2 - [(E^2(B^2)^{-1}R^2)] = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \\ -1 \\ 0 \end{bmatrix}$$

(iii) Resolução do sistema linear (2.3):

$$\begin{bmatrix} \bar{F} & E^3 \end{bmatrix} \begin{bmatrix} \bar{x}_R \\ \bar{x}_B^3 \end{bmatrix} = [d - E\bar{x}_B]$$

$$\left[\begin{array}{cc|c|ccc} 1 & -1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} \bar{x}_5^1 \\ \hline \bar{x}_2^1 \\ \bar{x}_1^2 \\ \hline \bar{x}_1^3 \\ \bar{x}_3^3 \\ \bar{x}_6^3 \end{array}\right] = \left[\begin{array}{c} 7 \\ 1 \\ \hline 5 \\ 2 \\ 3 \\ 6 \end{array}\right] \Rightarrow \left[\begin{array}{c} \bar{x}_5^1 \\ \hline \bar{x}_2^1 \\ \bar{x}_1^2 \\ \hline \bar{x}_1^3 \\ \bar{x}_3^3 \\ \bar{x}_6^3 \end{array}\right] = \left[\begin{array}{c} 5 \\ \hline 3 \\ 2 \\ \hline 3 \\ 3 \\ 1 \end{array}\right]$$

(iv) Correção das variáveis:
$x_R = \bar{x}_R \Rightarrow (x_R^1 \quad x_R^2)' = (\bar{x}_R^1 \quad \bar{x}_R^2)'$
$x_B^3 = \bar{x}_B^3$
$x_B = \bar{x}_B - B^{-1}Rx_R = \bar{x}_B - B^{-1}r = \bar{x} - t$

produto 1:

$$x_R^1 = \begin{bmatrix} x_5^1 \\ x_2^1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

$$r = R^1 x_R^1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix}$$

$$B^1 t = r \Leftrightarrow \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix} \Leftrightarrow \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} t_3 \\ t_2 \\ t_1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}$$

com solução: $t = (\ -5,\ \ -2,\ \ -2\ )'$; ou seja,

$$x_B^1 = \bar{x}_B^1 - t = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} - \begin{bmatrix} -5 \\ -2 \\ -2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \\ 6 \end{bmatrix}$$

produto 2:

$$x_R^2 = \begin{bmatrix} x_1^2 \end{bmatrix} = \begin{bmatrix} 2 \end{bmatrix}$$

$$r = R^2 x_R^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \begin{bmatrix} 2 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix}$$

$$B^2 t = r \Leftrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_3 \\ t_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -2 \end{bmatrix}$$

com solução: $t = (\ 2,\ \ -2,\ \ 2\ )'$; ou seja,

$$x_B^2 = \bar{x}_B^2 - t = \begin{bmatrix} 5 \\ 1 \\ 4 \end{bmatrix} - \begin{bmatrix} 2 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 2 \end{bmatrix}$$

Finalmente, juntando as componentes das árvores e os arcos em ciclo dos blocos, mais as variáveis de folga do acoplamento, temos a solução do sistema linear básico primal:

$$\widehat{x}_B = (x_B \mid x_R \mid x_B^3)' = (x_B^1 \quad x_B^2 \mid x_R^1 \quad x_R^2 \mid x_B^3)' =$$
$$= (\ 6 \quad 4 \quad 6 \mid 3 \quad 2 \quad 2 \mid 5 \quad 3 \mid 2 \mid 3 \quad 3 \quad 1\ )'$$

que conduz à solução completa abaixo, onde as variáveis não básicas são nulas:

$x^1 = (\ 4 \quad 3 \quad 6 \quad 0 \quad 5 \quad 6\ )'$
$x^2 = (\ 2 \quad 3 \quad 0 \quad 3 \quad 2 \quad 0\ )'$
$x^3 = (\ 3 \quad 0 \quad 3 \quad 0 \quad 0 \quad 1\ )'$

solução básica dual:

Seguindo os passos anteriores, temos:

(i) $(y^3)'\ E^3 = 0 \Rightarrow \begin{bmatrix} y_1^3 & y_3^3 & y_6^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$

$\Rightarrow y_1^3 = y_3^3 = y_6^3 = 0$

(ii) Resolver $(y^3)'\ \bar{F} = \bar{c}'$:

Cálculo do lado direito:

produto 1:

$(\bar{c}^1)' = c'_{R^1} - c'_{B^1}(B^1)^{-1}R^1 = c'_{R^1} - \bar{t}'R^1 = c'_{R^1} - \bar{r}'$

$$(B^1)'\bar{t} = c_{B^1} \Leftrightarrow \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \bar{t}_1 \\ \bar{t}_2 \\ \bar{t}_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \bar{t}_3 \\ \bar{t}_2 \\ \bar{t}_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

com solução: $\bar{t} = (\ -1, \quad -3, \quad -4\ )'$.

$$\bar{r}' = \bar{t}'R^1 = \begin{bmatrix} -1 & -3 & -4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} -4 & 3 \end{bmatrix}$$

$$(\bar{c}^1)' = c'_{R^1} - \bar{r}' = [\ 4 \quad 4\ ] - [\ -4 \quad 3\ ] = [\ 8 \quad 1\ ]$$

produto 2:

$$(\bar{c}^2)' = c'_{R^2} - c'_{B^2}(B^2)^{-1}R^2 = c'_{R^2} - \bar{t}'R^2 = c'_{R^2} - \bar{r}'$$

$$(B^2)'\bar{t} = c_{B^2} \Leftrightarrow \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{t}_1 \\ \bar{t}_2 \\ \bar{t}_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{t}_1 \\ \bar{t}_3 \\ \bar{t}_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix}$$

com solução: $\bar{t} = (\ 7, \quad 1, \quad 3\ )'$.

$$\bar{r}' = \bar{t}'R^2 = \begin{bmatrix} 7 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} = 6$$

$$(\bar{c}^2)' = c'_{R^2} - \bar{r}' = 7 - 6 = 1$$

Como a matriz $\bar{F}$ já foi calculada na solução primal, temos que a equação

$$(y^3)'\ \bar{F} = \bar{c}'$$

é

$$\begin{bmatrix} y_1^3 & y_2^3 & y_3^3 & y_4^3 & y_5^3 & y_6^3 \end{bmatrix} \left[\begin{array}{cc|c} 1 & -1 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{array}\right] = \left[\begin{array}{cc|c} 8 & 1 & 6 \end{array}\right]$$

ou ainda,

$$\left[\begin{array}{ccc|ccc} y_1^3 & y_3^3 & y_6^3 & y_2^3 & y_4^3 & y_5^3 \end{array}\right] \left[\begin{array}{cc|c} 1 & -1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & 0 \\ \hline 0 & 1 & -1 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \end{array}\right] = \left[\begin{array}{cc|c} 8 & 1 & 6 \end{array}\right]$$

Lembrando que no item (i) obtivemos $y_1^3 = y_3^3 = y_6^3 = 0$, este sistema se reduz a um sistema 3×3 com as últimas equações e variáveis, cuja solução é:

$(y_2^3,\ y_4^3,\ y_5^3)' = (\ 1,\ 15,\ 8\ )'$.

Logo, $y^3 = (\ 0,\ 1,\ 0,\ 15,\ 8,\ 0\ )'$.

(iii) Resolver $y'B = c_B' - (y^3)'\ E$:

produto 1:

$(y^1)'B^1 = c_{B^1}' - (y^3)'\ E^1$

Lado direito: $c_{B^1}' - (y^3)'\ E^1 =$

$$[\ 1 \quad 2 \quad 1\ ] - [\ 0 \quad 1 \quad 0 \quad 15 \quad 8 \quad 0\ ] \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = [\ 1 \quad 2 \quad 1\ ]$$

Assim,

$$(y^1)'B^1 = [y_1^1 \quad y_2^1 \quad y_3^1] \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} = [\ 1 \quad 2 \quad 1\ ] \Leftrightarrow$$

$$[y_3^1 \quad y_2^1 \quad y_1^1] \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} = [\ 1 \quad 2 \quad 1\ ] \Rightarrow y^1 = \begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \end{bmatrix} = \begin{bmatrix} -1 \\ -3 \\ -4 \end{bmatrix}$$

produto 2:

$(y^2)'B^2 = c_{B^2}' - (y^3)'\ E^2$

Lado direito: $c_{B^2}' - (y^3)'\ E^2 =$

$$[\,4\quad 1\quad 3\,]-[\,0\quad 1\quad 0\quad 15\quad 8\quad 0\,]\begin{bmatrix}0&0&0\\1&0&0\\0&0&0\\0&1&0\\0&0&1\\0&0&0\end{bmatrix}=[\,3\quad -14\quad -5\,]$$

Assim,

$$(y^2)'B^2=[y_1^2\quad y_2^2\quad y_3^2]\begin{bmatrix}1&0&0\\0&1&0\\-1&0&1\end{bmatrix}=[\,3\quad -14\quad -5\,]\Leftrightarrow$$

$$[y_1^2\quad y_3^2\quad y_2^2]\begin{bmatrix}1&0&0\\-1&1&0\\0&0&1\end{bmatrix}=[\,3\quad -5\quad -14\,]\Rightarrow$$

$$y^2=\begin{bmatrix}y_1^2\\y_2^2\\y_3^2\end{bmatrix}=\begin{bmatrix}-2\\-14\\-5\end{bmatrix}$$

Portanto, a solução dual completa é:

$(y^1 \mid y^2 \mid y^3)' = (\,-1,\ -3,\ -4 \mid -2,\ -14,\ -5 \mid 0,\ 1,\ 0,\ 15,\ 8,\ 0\,)'.$

2.3 Floresta geradora máxima

Vimos que uma base para o problema de multifluxo contém uma árvore geradora com raiz para cada produto (além dos outros n arcos escolhidos entre os arcos dos produtos e as variáveis de folga). Esta coleção de árvores geradoras nos fornece uma *floresta geradora*. De fato, se a rede não for conectada, teremos uma *floresta geradora* já para cada produto.

Um dos precondicionadores que utilizaremos (ver Capítulo 4) é baseado na *floresta geradora máxima*: para cada produto, os arcos são escolhidos de acordo com um certo *peso*, isto é, a árvore (ou floresta) geradora de cada produto é formada quando selecionamos os arcos de maior *peso* para compô-la.

Optamos por implementar o algoritmo de **Kruskal**, por ser um algoritmo eficiente, onde a seleção do arco de maior peso é feita através da estrutura de *heap*, uma árvore binária completa, onde cada nó possui uma informação chamada *peso* do nó. A propriedade do *heap* é que o peso de cada nó interno é maior ou igual que o peso de seus filhos. Maiores detalhes sobre esta estrutura e um algoritmo para montá-la podem ser encontrados em [9]. Aqui, o *heap* é usado para colocar o elemento de maior peso no topo da árvore. Para sua implementação, utilizamos os vetores:

$lst[1..n]$: lista de arcos para o *heap*; vetor inteiro. Depois de ordenado (*heap* concluído), o primeiro elemento contém o nó de maior peso.

$peso[1..n]$: peso de cada arco da rede; vetor real.

Como exemplo da estrutura de *heap*, veja a Figura 2.9, que representa uma árvore binária depois do *heap* (nós reordenados), supondo que o vetor *peso* seja dado por (10.7 , 0.5 , 30.1 , 1.2 , 20.8 , 40.5) . Apresentamos os vetores *lst* e *peso* antes e depois do *heap*.

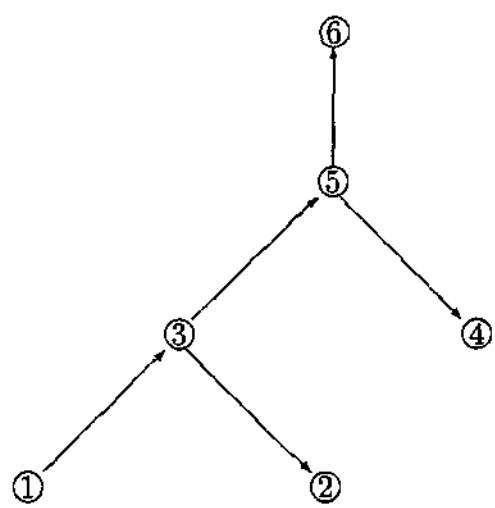


Figura 2.9: *exemplo de heap*

inicial

lst	1	2	3	4	5	6
peso	10.7	0.5	30.1	1.2	20.8	40.5

final

lst	6	3	5	4	1	2
peso	40.5	30.1	20.8	1.2	10.7	1.5

No algoritmo de *Kruskal*, a floresta geradora T, inicialmente, é vazia, e cada nó de G é uma componente distinta. No decorrer do algoritmo vão sendo incluídos arcos de modo que cada componente de T forme uma árvore geradora máxima com nós nesta componente. Ao final do algoritmo, se G é conectada, T é a árvore geradora máxima. Caso contrário, T é a floresta geradora máxima. Para redes conectadas, este algoritmo é executado em ordem de $\mathcal{O}(nlogm)$ operações (ver, por exemplo,[9]).

A seguir, apresentamos o algoritmo de *Kruskal* e um algoritmo para construir um *heap*. De fato, na implementação do Algoritmo 2.3, T é representada pelos vetores *pred*, *fio* e *rev* descritos anteriormente; na busca da raiz de um nó precisamos percorrer o vetor *pred* e a inclusão de um arco em T envolve a atualização de *pred*, *fio* e *rev*. Observe que, embora o primeiro *heap* seja efetuado em todo o vetor *lst*, os demais consideram uma posição a menos neste vetor (a dimensão do *heap* sempre diminui de uma unidade).

Algoritmo 2.3 (*monta a floresta geradora máxima*)

```
T ← ∅ ;
for  j ← 1..n  do  lst[j] ← j ;
for  j ← 1..n div 2 do  heap(j, n, lst, peso) ;
ult ← n          { no. de elementos do heap };
econt ← 0        { no. de arcos examinados };
tcont ← 0        { no. de arcos da floresta };
conet ← true     { se conectado, é árvore  };
while (tcont < m − 1) and (econt < n) do
   begin
      econt ← econt + 1 ;
      v1 ← orig[lst[1]] ;
      v2 ← dest[lst[1]] ;
      r1 ← raiz(v1)    { encontra a raiz do nó v1 } ;
      r2 ← raiz(v2)    { encontra a raiz do nó v2 } ;
      if r1 ≠ r2
        then           { inclui (v1, v2) na floresta};
          begin
            tcont ← tcont + 1 ;
            T ← T ∪ (v1, v2) ;
          endthen
      lst[1] ← lst[ult] ;
      ult ← ult − 1 ;
      heap(1, ult, lst, peso) ;
   endwhile;
if tcont ≠ m−1 then conet ← false     { floresta em vez de árvore }
```

Algoritmo 2.4 (*monta o heap*)

{ monta o heap no vetor *lst* de acordo com os valores dados no vetor *peso*, a partir do elemento *first* até o elemento *last* }

```
heap(first, last, lst, peso)
begin
    i ← first ;
    while i ≤ last div 2 do
     begin
       k ← 2 * i ;
       if (k < last) and (peso[lst[k]] < peso[lst[k+1]] then k ← k+1 ;
       if (peso[lst[k]] > peso[lst[i]]
         then begin    {troca os nós lst[k] e lst[i] }
                  lst[k] ↔ lst[i] ;
                  i ← k ;
              endthen
         else i ← last ;
     endwhile;
endheap.
```

Capítulo 3

Método Primal - Dual de Pontos Interiores

3.1 Introdução

No Capítulo 1, apontamos algumas técnicas específicas para resolver o *problema de multifluxo*. No entanto, sendo este um problema de grande porte e de grande complexidade, sua resolução ainda justifica o estudo e aplicação de outros algoritmos. Neste capítulo, descreveremos a aplicação de um método de pontos interiores a este problema, uma vez que a melhor complexidade conhecida para problemas de multifluxo foi obtida com estes algoritmos.

Os métodos de pontos interiores têm se mostrado competitivos para problemas "grandes". Estes métodos possuem a vantagem de convergir em um número de iterações relativamente pequeno, embora o trabalho de uma iteração seja muito maior do que o demandado no método Simplex, por exemplo. Desde que Karmarkar [33] apresentou seu método, muitas variantes dele surgiram e/ou reapareceram. Dentre elas, a que tem se mostrado mais eficiente é a versão primal- dual, que foi introduzida por Megiddo [50] e desenvolvida mais ou menos simultaneamente por vários outros autores, como, por exemplo, Kojima, Mizuno e Yoshise [40], Gonzaga [20], Monteiro e Adler [54], entre outros. Esta versão tem um esforço computacional por iteração da mesma ordem que outras classes de métodos de pontos interiores, mas apresenta melhores propriedades teóricas na análise de complexidade de pior caso e bom desempenho na prática, segundo Monteiro e Adler [54],

Wright [74] e Oliveira [56].

Outros pesquisadores também contribuíram com implementações computacionais, como, por exemplo, Lustig, Marsten e Shanno [43, 44], Mehrotra [51, 52], Resende e Veiga [61], McShane, Monma e Shanno [49], entre outros.

Para *problemas em redes* com um único produto, encontramos também outras contribuições e implementações, como, por exemplo, Mehrotra [53], Wallacher e Zimmermann [71], Masuzawa, Mizuno e Mori [47], Mizuno e Masuzawa [55], Resende e Pardalos [59], Resende e Veiga [60, 62], Portugal, Resende, Veiga e Júdice [58] e, para *multifluxo*, uma implementação de Li e Lustig [42] e outra e Castro [10]. Os resultados computacionais apresentados por estes autores foram bastante satisfatórios quando comparados a outros métodos e códigos.

Para facilitar a aplicação do método primal-dual ao problema (1.1), vamos escrever este problema de uma maneira conveniente.

3.2 Reformulação do problema

Vamos tomar nosso problema original (1.1). Chamando de x^{p+1} as folgas relativas às restrições de acoplamento e de s^k, $k = 1, \cdots, p$ as folgas relativas às canalizações dos produtos, escrevemos o problema (1.1) na forma padrão:

$$\min \quad \sum_{k=1}^{p} (c^k)' x^k$$

$$s.a. \quad \begin{cases} Ax^k = b^k, \quad k = 1, \ldots, p \\ \sum_{k=1}^{p} x^k + x^{p+1} = d \\ x^k + s^k = u^k, \quad k = 1, \ldots, p \\ x^k, \; s^k \geq 0, \; k = 1, \ldots, p, \; x^{p+1} \geq 0 \end{cases} \tag{3.1}$$

com x^{p+1} e s^k, $k = 1, \cdots, p \in \mathbb{R}^n$. Seu dual é:

$$\begin{aligned} \max \quad & \{ \sum_{k=1}^{p} [\ (b^k)'y^k - (u^k)'w^k\ ] + d'y^{p+1}\ \} \\ s.a. \quad & \begin{cases} A'y^k + y^{p+1} - w^k + z^k = c^k, \quad k = 1, \ldots, p \\ w^k,\ z^k\ \geq 0,\ k = 1, \ldots, p,\ y^{p+1} \leq 0 \end{cases} \end{aligned} \tag{3.2}$$

onde, para cada produto $k = 1, \cdots, p$, definimos:

y^k: vetor das variáveis duais correspondentes às restrições de conservação de fluxo;
w^k: vetor das variáveis duais correspondentes às canalizações;
z^k: vetor de folgas das restrições duais;
y^{p+1}: vetor das variáveis duais correspondentes às restrições de acoplamento.

$(y^k \in I\!R^m\ ,\ w^k,\ z^k \in I\!R^n,\ k = 1, \cdots, p,\ \ y^{p+1} \in I\!R^n)$.

Este par primal-dual pode agora ser reescrito num formato conveniente. Vamos formar a matriz $\hat{A}$ com as p matrizes A dos produtos (ou blocos), já com uma linha retirada (de modo que a nova matriz A tem posto completo) mais as matrizes identidade das restrições de acoplamento. O último bloco será denominado bloco $p+1$. Estendendo esta notação para as demais variáveis, temos o seguinte par primal-dual:

$$\begin{aligned} \min \quad & \hat{c}'\hat{x} \\ s.a. \quad & \begin{cases} \hat{A}\hat{x} = \hat{b} \\ \hat{x} + \hat{s} = \hat{u} \\ \hat{x},\ \hat{s} \geq 0 \end{cases} \end{aligned} \tag{3.3}$$

$$\begin{aligned} \max \quad & \{\ \hat{b}'\hat{y} - \hat{u}'\hat{w}\ \} \\ s.a. \quad & \begin{cases} \hat{A}'\hat{y} - \hat{w} + \hat{z} = \hat{c} \\ \hat{w},\ \hat{z} \geq 0 \end{cases} \end{aligned} \tag{3.4}$$

onde:

$$\hat{A} = \begin{bmatrix} A & & & & \\ & A & & & \\ & & \ddots & & \\ & & & A & \\ I & I & \cdots & I & I \end{bmatrix}$$

$$\hat{b} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^p \\ d \end{bmatrix}, \quad \hat{c} = \begin{bmatrix} c^1 \\ c^2 \\ \vdots \\ c^p \\ 0 \end{bmatrix}, \quad \hat{u} = \begin{bmatrix} u^1 \\ u^2 \\ \vdots \\ u^p \\ \infty \end{bmatrix}, \quad \hat{x} = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^p \\ x^{p+1} \end{bmatrix},$$

$$\hat{s} = \begin{bmatrix} s^1 \\ s^2 \\ \vdots \\ s^p \\ \infty \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^p \\ y^{p+1} \end{bmatrix}, \quad \hat{w} = \begin{bmatrix} w^1 \\ w^2 \\ \vdots \\ w^p \\ 0 \end{bmatrix}, \quad \hat{z} = \begin{bmatrix} z^1 \\ z^2 \\ \vdots \\ z^p \\ z^{p+1} \end{bmatrix}$$

Vamos definir

$$\hat{m} = p(m-1) + n$$

e

$$\hat{n} = (p+1)n$$

Assim, temos que $\hat{A} \in \mathbb{R}^{\hat{m} \times \hat{n}}$; $\hat{x}$, $\hat{c}$, $\hat{s}$, $\hat{u} \in \mathbb{R}^{\hat{n}}$; $\hat{b}$, $\hat{y} \in \mathbb{R}^{\hat{m}}$. O símbolo "$\infty$" em $\hat{s}$ e $\hat{u}$ representa um vetor com n componentes ∞; o símbolo "0" em $\hat{c}$ e $\hat{w}$ representa um vetor com n componentes nulas. Convém notar ainda que o sinal de y^{p+1} está implícito na restrição dual $\hat{A}'\hat{y} - \hat{w} + \hat{z} = \hat{c}$, uma vez que, para o bloco $p+1$, esta restrição se reduz a $y^{p+1} - w^{p+1} + z^{p+1} = 0 \Rightarrow y^{p+1} = -z^{p+1} \Rightarrow y^{p+1} \leq 0$.

3.3 Especialização para multifluxo

O método primal-dual de pontos interiores utiliza a função *barreira logarítmica* para criar a família de problemas

$$\begin{aligned} \min \quad & \{\, \widehat{c}'\widehat{x} - \mu\sum_{j=1}^{\widehat{n}} \ln \widehat{x}_j - \mu\sum_{j=1}^{\widehat{n}} \ln \widehat{s}_j \,\} \\ s.a. \quad & \begin{cases} \hat{A}\widehat{x} = \widehat{b} \\ \widehat{x} + \widehat{s} = \widehat{u} \\ \widehat{x}, \quad \widehat{s} > 0 \end{cases} \end{aligned} \tag{3.5}$$

onde $\mu > 0$ é chamado de *parâmetro de barreira ou de centragem.* É bem conhecido que, resolvendo o problema (3.5) para certas seqüencias de μ que convergem para zero, geramos uma seqüencia de soluções que convergem para a solução do problema original (3.3).

Chamaremos de *ponto primal-dual factível* um ponto $(\widehat{x}, \widehat{s}, \widehat{y}, \widehat{w}, \widehat{z})$ que satisfaz

$$\hat{A}\widehat{x} = \widehat{b},\ \widehat{x} + \widehat{s} = \widehat{u},\ \hat{A}'\widehat{y} - \widehat{w} + \widehat{z} = \widehat{c},\ \ (\widehat{x}, \widehat{s}, \widehat{w}, \widehat{z}) \geq 0\ .$$

Um ponto primal-dual factível $(\widehat{x}, \widehat{s}, \widehat{y}, \widehat{w}, \widehat{z})$ é *interior* se ele satisfaz $(\widehat{x}, \widehat{s}, \widehat{w}, \widehat{z}) > 0$.

Como a função objetivo do problema (3.5) é estritamente convexa, este problema possui, no máximo, um mínimo global e, se este mínimo existe ele é bem determinado pelas condições de Karush-Kuhn-Tucker, ou seja, para $(\widehat{x}, \widehat{s}, \widehat{w}, \widehat{z}) \geq 0$, temos:

$$\begin{cases} \hat{A}\widehat{x} - \widehat{b} = 0 \\ \widehat{x} + \widehat{s} - \widehat{u} = 0 \\ \hat{A}'\widehat{y} - \widehat{w} + \widehat{z} - \widehat{c} = 0 \\ \widehat{X}\widehat{Z}\mathbf{1} - \mu\mathbf{1} = 0 \\ \widehat{S}\widehat{W}1 - \mu\mathbf{1} = 0 \end{cases} \tag{3.6}$$

onde $\widehat{X} = \begin{bmatrix} X^1 & & & \\ & \ddots & & \\ & & X^p & \\ & & & X^{p+1} \end{bmatrix}$

com $X^k = diag(x_j^k)$, $k = 1, \cdots, p+1$; $j = 1, \cdots, n$; $\widehat{S}$, $\widehat{W}$ e $\widehat{Z}$ têm definições similares à matriz $\widehat{X}$, isto é, cada uma delas é uma matriz diagonal

por blocos com $\hat{n}$ elementos e $\mathbf{1}$ é o vetor coluna com $\hat{n}$ componentes iguais a 1. Convém notar que, pelas definições anteriores de $\hat{s}$ e $\hat{w}$, cada uma das n componentes de S^{p+1} é ∞, enquanto que as n componentes de W^{p+1} são nulas.

Notamos também que (3.6) é um sistema de equações não lineares da forma

$$H(\hat{x}, \hat{s}, \hat{y}, \hat{w}, \hat{z}) = 0$$

onde, no primeiro e no segundo conjuntos de equações contemplamos a factibilidade primal; no terceiro conjunto, a factibilidade dual e nos dois últimos, as condições de folgas complementares.

A cada iteração, a partir de um ponto interior $(\hat{x}, \hat{s}, \hat{y}, \hat{w}, \hat{z})$, isto é, $(\hat{x}, \hat{s}, \hat{w}, \hat{z}) > 0$, é obtida uma *direção de deslocamento* para se chegar ao próximo ponto, em direção à trajetória central. Esta direção é escolhida como sendo a *direção de Newton* associada ao sistema (3.6). Se $J_H(.)$ é a *matriz Jacobiana* de H no ponto $(\hat{x}, \hat{s}, \hat{y}, \hat{w}, \hat{z})$ e $H(.)$ é a função H aplicada no ponto $(\hat{x}, \hat{s}, \hat{y}, \hat{w}, \hat{z})$, a direção de deslocamento Δ é determinada pelo sistema linear

$$J_H(.)\,\Delta = -H(.)$$

ou ainda,

$$\begin{bmatrix} \hat{A} & 0 & 0 & 0 & 0 \\ I & I & 0 & 0 & 0 \\ 0 & 0 & \hat{A}' & -I & I \\ \widehat{Z} & 0 & 0 & 0 & \widehat{X} \\ 0 & \widehat{W} & 0 & \widehat{S} & 0 \end{bmatrix} \begin{bmatrix} \Delta\hat{x} \\ \Delta\hat{s} \\ \Delta\hat{y} \\ \Delta\hat{w} \\ \Delta\hat{z} \end{bmatrix} = \begin{bmatrix} \hat{b} - \hat{A}\hat{x} \\ \hat{u} - \hat{x} - \hat{s} \\ \hat{c} - \hat{A}'\hat{y} + \hat{w} - \hat{z} \\ \mu\mathbf{1} - \widehat{X}\widehat{Z}\mathbf{1} \\ \mu\mathbf{1} - \widehat{S}\widehat{W}\mathbf{1} \end{bmatrix} \tag{3.7}$$

onde:
I : matriz identidade de ordem $\hat{n}$;
$\Delta = (\ \Delta\hat{x},\ \Delta\hat{s},\ \Delta\hat{y},\ \Delta\hat{w},\ \Delta\hat{z}\ )'$, com:
$\Delta\hat{x} = (\ \Delta\hat{x}^1,\ \cdots,\ \Delta\hat{x}^p,\ \Delta\hat{x}^{p+1}\ )'$;
$\Delta\hat{s} = (\ \Delta\hat{s}^1,\ \cdots,\ \Delta\hat{s}^p,\ \Delta\hat{s}^{p+1}\ )'$;
$\Delta\hat{y} = (\ \Delta\hat{y}^1,\ \cdots,\ \Delta\hat{y}^p,\ \Delta\hat{y}^{p+1}\ )'$;

$\Delta\widehat{w} = (\Delta\widehat{w}^1, \cdots, \Delta\widehat{w}^p, \Delta\widehat{w}^{p+1})'$;
$\Delta\widehat{z} = (\Delta\widehat{z}^1, \cdots, \Delta\widehat{z}^p, \Delta\widehat{z}^{p+1})'$.

Note que o sistema linear (3.7) é muito grande, já que $\hat{A} \in I\!R^{\widehat{m}\times\widehat{n}}$ e $\widehat{X}$, $\hat{S}$, $\hat{Z}$, $\widehat{W} \in I\!R^{\widehat{n}\times\widehat{n}}$. As equações deste sistema são:

(i) $\hat{A}\Delta\hat{x} = \hat{b} - \hat{A}\hat{x}$

(ii) $\Delta\hat{x} + \Delta\hat{s} = \hat{u} - \hat{x} - \hat{s}$

(iii) $\hat{A}'\Delta\hat{y} - \Delta\hat{w} + \Delta\hat{z} = \hat{c} - \hat{A}'\hat{y} + \hat{w} - \hat{z}$

(iv) $\widehat{X}\Delta\hat{z} + \hat{Z}\Delta\hat{x} = \mu\mathbf{1} - \widehat{X}\hat{Z}\mathbf{1}$

(v) $\hat{S}\Delta\hat{w} + \widehat{W}\Delta\hat{s} = \mu\mathbf{1} - \hat{S}\widehat{W}\mathbf{1}$

Estas equações, por sua vez, podem ser desmembradas devido à existência de $p+1$ blocos. Além disso, as referentes ao último bloco são mais simples do que as referentes aos p primeiros blocos. Desta maneira, podemos reescrever estas equações:
Para $k = 1, \cdots, p$:

(i) $A\Delta x^k = b^k - Ax^k$

(ii) $\Delta x^k + \Delta s^k = u^k - x^k - s^k$

(iii) $A'\Delta y^k + \Delta y^{p+1} - \Delta w^k + \Delta z^k = c^k - A'y^k - y^{p+1} + w^k - z^k$

(iv) $X^k\Delta z^k + Z^k\Delta x^k = \mu\mathbf{1} - \widehat{X}^k\widehat{Z}^k\mathbf{1}$

(v) $S^k\Delta w^k + W^k\Delta s^k = \mu\mathbf{1} - \hat{S}^k\widehat{W}^k\mathbf{1}$

e, para o bloco $p+1$:

(i) $\sum_{k=1}^{p+1}\Delta x^k = d - \sum_{k=1}^{p+1} x^k$

(iii) $\Delta y^{p+1} + \Delta z^{p+1} = -y^{p+1} - z^{p+1}$

(iv) $X^{p+1}\Delta z^{p+1} + Z^{p+1}\Delta x^{p+1} = \mu\mathbf{1} - \widehat{X}^{p+1}\widehat{Z}^{p+1}\mathbf{1}$

As equações (ii) e (v) do bloco $p+1$ foram desconsideradas devido às particularidades deste bloco.

Manipulando as equações de (3.7), conseguimos obter as expressões para a direção Δ . A componente $\Delta\widehat{y}$ é obtida através da resolução de um sistema linear de dimensão bem menor do que o sistema (3.7), com $\widehat{m}$ equações e incógnitas, que será chamado de *sistema reduzido*, ou seja:

$$(\hat{A}\hat{\Theta}\hat{A}')\ \Delta\widehat{y} = \hat{A}\hat{\Theta}\ \{[\ \mu((\widehat{S})^{-1} - (\widehat{X})^{-1}) - \widehat{W} + \widehat{Z}\ ]\mathbf{1} + (\ \hat{c} - \hat{A}'\widehat{y} + \hat{w} - \hat{z}\ ) - (\widehat{S})^{-1}\widehat{W}(\ \hat{u} - \hat{x} - \hat{s}\ )\} + (\ \hat{b} - \hat{A}\hat{x}\ )$$

ou, de uma maneira mais conveniente,

$$\underbrace{(\hat{A}\hat{\Theta}\hat{A}')}_{\widehat{Q}}\ \Delta\widehat{y} = \underbrace{\hat{A}\hat{\Theta}\ [r_f + r_d - r_c] + r_p}_{\widehat{q}} \tag{3.8}$$

ou,

$$\widehat{Q}\ \Delta\widehat{y} = \widehat{q} \tag{3.9}$$

onde:

$r_f = [\ \mu((\widehat{S})^{-1} - (\widehat{X})^{-1}) - \widehat{W} + \widehat{Z}\ ]\mathbf{1}$
$r_d = \hat{c} - \hat{A}'\widehat{y} + \hat{w} - \hat{z}$
$r_c = (\widehat{S})^{-1}\widehat{W}(\ \hat{u} - \hat{x} - \hat{s}\ )$
$r_p = \hat{b} - \hat{A}\hat{x}$

Em muitas implementações, é comum considerar as canalizações satisfeitas, isto é, $\hat{s} = \hat{u} - \hat{x}$, o que implica em $r_c = 0$.

A matriz

$$\hat{\Theta} = [(\widehat{X})^{-1}\hat{Z} + (\widehat{S})^{-1}\widehat{W}]^{-1} \tag{3.10}$$

é denominada *matriz de scaling* do método primal-dual e é tal que

$$\hat{\Theta} = \begin{bmatrix} \Theta^1 & & & \\ & \ddots & & \\ & & \Theta^p & \\ & & & \Theta^{p+1} \end{bmatrix}$$

Cada Θ^k, matriz de "scaling" do produto k, é uma matriz diagonal de dimensão n, $k = 1, \cdots, p+1$, com

$$\Theta^k = [\ (X^k)^{-1}Z^k\ + (S^k)^{-1}W^k\ ]^{-1}\ ,$$

isto é, cada elemento de Θ^k tem a forma

$$(x_j^k s_j^k)\ /\ (s_j^k z_j^k + x_j^k w_j^k)\ ,\quad j = 1, \cdots, n\ . \tag{3.11}$$

A expressão para Θ^{p+1} é

$$\Theta^{p+1} = [\ (X^{p+1})^{-1}Z^{p+1}\ ]^{-1}$$

e cada elemento possui, agora, a forma

$$x_j^{p+1}\ /\ z_j^{p+1} = -x_j^{p+1}\ /\ y_j^{p+1}\ ,\quad j = 1, \cdots, n\ . \tag{3.12}$$

As outras componentes da direção são obtidas a partir de $\Delta\widehat{y}$ por:

$$\begin{cases} \Delta\widehat{x} = \widehat{\Theta}\ [\ \hat{A}'\Delta\widehat{y} - r_f - r_d + r_c] \\ \Delta\widehat{s} = \widehat{u} - \widehat{x} - \widehat{s} - \Delta\widehat{x} \\ \Delta\widehat{z} = (\widehat{X})^{-1}[\ \mu\mathbf{1} - \widehat{X}\widehat{Z}\mathbf{1} - \widehat{Z}\Delta\widehat{x}\ ] \\ \Delta\widehat{w} = (\widehat{S})^{-1}[\ \mu\mathbf{1} - \widehat{S}\widehat{W}\mathbf{1} - \widehat{W}\Delta\widehat{s}\ ] \end{cases} \tag{3.13}$$

Como cada componente da direção $\Delta = (\ \Delta\widehat{x},\ \Delta\widehat{s},\ \Delta\widehat{y},\ \Delta\widehat{w},\ \Delta\widehat{z}\ )$ é particionada em $p+1$ blocos, ou seja, $\Delta x^k,\ \Delta s^k,\ \Delta w^k,\ \Delta z^k \in I\!R^n,\ k = 1, \cdots, p+1,\ \Delta y^k \in I\!R^{m-1},\ k = 1, \cdots, p$ e $\Delta y^{p+1} \in I\!R^n$, obtemos:

$$\Delta x^k = \Theta^k \, [\, A'\Delta y^k + \Delta y^{p+1} - r_f^k - r_d^k + r_c^k \,], \; k = 1, \cdots, p$$
$$\Delta x^{p+1} = \Theta^{p+1} \, [\, \Delta y^{p+1} - r_f^{p+1} \,]$$

$$\Delta s^k = u^k - x^k - s^k - \Delta x^k$$
$$\Delta s^{p+1} = \infty$$

$$\Delta z^k = [\, \mu(X^k)^{-1} - Z^k \,]\mathbf{1} - (X^k)^{-1} Z^k \Delta x^k \;, \; k = 1, \cdots, p$$
$$\Delta z^{p+1} = -\Delta y^{p+1}$$

$$\Delta w^k = [\, \mu(S^k)^{-1} - W^k \,]\mathbf{1} - (S^k)^{-1} W^k (u^k - x^k - s^k - \Delta x^k) \;, \; k = 1, \cdots, p$$
$$\Delta w^{p+1} = 0$$

onde:

$$r_f^k = \{\, \mu[\, (S^k)^{-1} - (X^k)^{-1} \,] - W^k + Z^k \,\}\mathbf{1} \;, \; k = 1, \cdots, p$$
$$r_d^k = c^k - A'y^k - y^{p+1} + w^k - z^k \;, \; k = 1, \cdots, p$$
$$r_c^k = (S^k)^{-1} \, W^k \, (u^k - x^k - s^k) \;, \; k = 1, \cdots, p$$
$$r_f^{p+1} = [\, -\mu(X^{p+1})^{-1} + Z^{p+1} \,]\mathbf{1}$$
$$r_d^{p+1} = 0$$
$$r_c^{p+1} = 0 \;.$$

O lado direito de (3.9) é particionado em $p+1$ blocos do tipo:

$q^k = A\Theta^k \, [\, r_f^k + r_d^k - r_c^k \,] + r_p^k$, onde $r_p^k = b^k - Ax^k$, $k = 1, \cdots, p$ e

$q^{p+1} = r_a + \Theta^{p+1} \, r_f^{p+1} + \sum_{k=1}^{p} \Theta^k \, [r_f^k + r_d^k - r_c]$, onde $r_a = d - \sum_{k=1}^{p+1} x^k$.

Observe que $q^k \in I\!R^{m-1}$, $k = 1, \cdots, p$ e $q^{p+1} \in I\!R^n$.

Assim, podemos escrever o Algoritmo 3.1 para o método primal-dual. Neste algoritmo, ℓ está representando o número da iteração corrente.

Algoritmo 3.1 (*método primal-dual de pontos interiores*)

(Inicialização)
Fornecer $(\widehat{x}^0, \widehat{s}^0, \widehat{y}^0, \widehat{w}^0, \widehat{z}^0)$ tal que $\widehat{x}^0, \widehat{s}^0, \widehat{w}^0, \widehat{z}^0 > 0$;
$\ell \leftarrow 0$;

Repetir até que algum critério de parada seja satisfeito:

- *(parâmetro de centragem)*
  Escolher $\eta^\ell \in [0,\ 1)$;
  $\gamma^\ell \leftarrow (\widehat{x}^\ell)'\ \widehat{z}^\ell + (\widehat{w}^\ell)'\ \widehat{s}^\ell$;
  $\mu^\ell \leftarrow \eta^\ell \gamma^\ell\ /\ \widehat{n}$;

- *(sistema linear das direções)*
  Resolver o sistema linear (3.9) para $\Delta\widehat{y}$, isto é, $\widehat{Q}\ \Delta\widehat{y} = \widehat{q}$
  e obter $\Delta\widehat{x}$, $\Delta\widehat{s}$, $\Delta\widehat{w}$ e $\Delta\widehat{z}$ por (3.13) ;

- *(teste da razão)*
  Calcular:
  $\lambda_P^\ell = \min\limits_i \{\ (-\widehat{x}_i^\ell\ /\ \Delta\widehat{x}_i^\ell\ ,\ \Delta\widehat{x}_i^\ell < 0\ ) \cap (\ -\widehat{s}_i^\ell\ /\ \Delta\widehat{s}_i^\ell\ ,\ \Delta\widehat{s}_i^\ell < 0\ )\ \}$;
  $\lambda_D^\ell = \min\limits_i \{\ (-\widehat{z}_i^\ell\ /\ \Delta\widehat{z}_i^\ell\ ,\ \Delta\widehat{z}_i^\ell < 0\ ) \cap (\ -\widehat{w}_i^\ell\ /\ \Delta\widehat{w}_i^\ell\ ,\ \Delta\widehat{w}_i^\ell < 0\ )\ \}$;
  Escolher $\delta^\ell \in (0,\ 1)$ e calcular
  $\alpha_P^\ell = \min\{1,\ \delta^\ell \lambda_P^\ell\}$;
  $\alpha_D^\ell = \min\{1,\ \delta^\ell \lambda_D^\ell\}$;

- *(novo ponto)*
  $(\ \widehat{x}^{\ell+1},\ \widehat{s}^{\ell+1}\ ) \leftarrow (\ \widehat{x}^\ell,\ \widehat{s}^\ell\ ) + \alpha_P^\ell\ (\Delta\widehat{x}^\ell,\ \Delta\widehat{s}^\ell)$;
  $(\ \widehat{y}^{\ell+1},\ \widehat{w}^{\ell+1},\ \widehat{z}^{\ell+1}\ ) \leftarrow (\ \widehat{y}^\ell,\ \widehat{w}^\ell,\ \widehat{z}^\ell\ ) + \alpha_D^\ell\ (\Delta\widehat{y}^\ell,\ \Delta\widehat{w}^\ell,\ \Delta\widehat{z}^\ell)$;
  $\ell \leftarrow \ell + 1$;

Podemos colocar algumas observações a respeito deste algoritmo:

Na literatura, são apresentadas diferentes escolhas para os parâmetros η^ℓ e δ^ℓ, gerando variantes deste método (Monteiro e Adler [54], Kojima, Megiddo e Mizuno [39], Portugal, Resende, Veiga e Júdice [58], Mehrotra e Wang

[53], entre outros). Se $\eta^\ell = 0$, temos a versão afim do método. Para certas escolhas de η^ℓ e δ^ℓ e um ponto inicial factível, o método apresenta propriedades teóricas muito boas e complexidade polinomial (Gonzaga [20]). Nas implementações computacionais, costuma-se começar com um ponto interior infactível, uma vez que é caro, computacionalmente, encontrar um ponto inicial factível. No entanto, a teoria apresenta melhores resultados para pontos iniciais factíveis (Wright [74]). Nossa versão é do tipo *primal infactível, dual factível*, uma vez que é fácil obter uma solução inicial factível para o dual, mas não para o primal, isto é, as restrições duais $\widehat{A}'\widehat{y} - \widehat{w} + \widehat{z} = \widehat{c}$, $\widehat{w} > 0$, $\widehat{z} > 0$ são sempre satisfeitas e as canalizações também ($\widehat{x} + \widehat{s} = \widehat{u}$, $\widehat{x} > 0$, $\widehat{s} > 0$, o que implica em $r_c = 0$ e $r_d = 0$).

Se o passo completo de Newton fornecer um ponto não negativo, o método coloca $\alpha^\ell = 1$. Na prática, é melhor separar o tamanho do passo para o primal e para o dual; por isso definimos α_P^ℓ e α_D^ℓ .

O critério de parada de algoritmo é baseado nas condições de otimalidade do problema.

Por último, o passo do algoritmo que demanda o maior esforço computacional é a resolução do sistema linear (3.9), motivo pelo qual este passo será tratado num capítulo à parte.

3.3.1 Preditor-Corretor

Existe ainda uma variante do método primal-dual chamada de *preditor-corretor* (Mehrotra [51] e Lustig, Marsten, Shanno [44]). A diferença fundamental entre o método primal-dual descrito pelo Algoritmo 3.1 e esta variante está na maneira de calcular a direção de deslocamento: no preditor-corretor, a direção é obtida resolvendo dois sistemas lineares. Primeiro resolvemos

$$\begin{bmatrix} \hat{A} & 0 & 0 & 0 & 0 \\ I & I & 0 & 0 & 0 \\ 0 & 0 & \hat{A}' & -I & I \\ \widehat{Z} & 0 & 0 & 0 & \widehat{X} \\ 0 & \widehat{W} & 0 & \widehat{S} & 0 \end{bmatrix} \begin{bmatrix} \bar{\Delta}\hat{x} \\ \bar{\Delta}\hat{s} \\ \bar{\Delta}\hat{y} \\ \bar{\Delta}\hat{w} \\ \bar{\Delta}\hat{z} \end{bmatrix} = \begin{bmatrix} \hat{b} - \hat{A}\hat{x} \\ \hat{u} - \hat{x} - \hat{s} \\ \hat{c} - \hat{A}'\hat{y} + \hat{w} - \hat{z} \\ -\widehat{X}\widehat{Z}\mathbf{1} \\ -\widehat{S}\widehat{W}\mathbf{1} \end{bmatrix}$$

que é o mesmo sistema linear (3.7), mas com novas variáveis
$\bar{\Delta} = (\bar{\Delta}\hat{x},\ \bar{\Delta}\hat{s},\ \bar{\Delta}\hat{y},\ \bar{\Delta}\hat{w},\ \bar{\Delta}\hat{z})$, chamadas de *direções afins*. Seja $\bar{Q}$ a matriz deste sistema linear. O segundo sistema linear a ser resolvido é

$$\begin{bmatrix} \hat{A} & 0 & 0 & 0 & 0 \\ I & I & 0 & 0 & 0 \\ 0 & 0 & \hat{A}' & -I & I \\ \hat{Z} & 0 & 0 & 0 & \hat{X} \\ 0 & \hat{W} & 0 & \hat{S} & 0 \end{bmatrix} \begin{bmatrix} \Delta\hat{x} \\ \Delta\hat{s} \\ \Delta\hat{y} \\ \Delta\hat{w} \\ \Delta\hat{z} \end{bmatrix} = \begin{bmatrix} \hat{b} - \hat{A}\hat{x} \\ \hat{u} - \hat{x} - \hat{s} \\ \hat{c} - \hat{A}'\hat{y} + \hat{w} - \hat{z} \\ \mu\mathbf{1} - \hat{X}\hat{Z}\mathbf{1} - \bar{\Delta}\hat{x}\bar{\Delta}\hat{z}\mathbf{1} \\ \mu\mathbf{1} - \hat{S}\hat{W}\mathbf{1} - \bar{\Delta}\hat{s}\bar{\Delta}\hat{w}\mathbf{1} \end{bmatrix}$$

Note que temos a mesma matriz $\bar{Q}$ nos dois sistemas lineares. A diferença entre eles está apenas no lado direito: nas duas últimas componentes do segundo sistema temos a presença dos termos não lineares $\bar{\Delta}\hat{x}\bar{\Delta}\hat{z}$ e $\bar{\Delta}\hat{s}\bar{\Delta}\hat{w}$.

Esta variante reduz o número de iterações do Algoritmo 3.1, mas exige que resolvamos agora dois sistemas lineares por iteração do primal-dual. No entanto, como a matriz destes sistemas é a mesma matriz $\bar{Q}$, os cálculos utilizados para construí-la (ou parte dela) são efetuados uma única vez.

Outra observação é que, transformações similares às que foram feitas para transformar o sistema linear (3.7) no sistema reduzido (3.9) também podem ser efetuadas nesta variante, de modo que os dois sistemas lineares resolvidos em cada iteração do primal-dual possuem a mesma dimensão do sistema (3.9), isto é, $\widehat{m}$ equações e incógnitas.

Não implementamos esta variante porque nosso investimento maior na resolução do sistema linear das direções foi na utilização do método dos gradientes conjugados. Em vez disso, partimos para a obtenção de um precondicionador que tornasse o método dos gradientes conjugados mais eficiente. Utilizando métodos iterativos ao invés de diretos, não podemos aproveitar a fatoração da matriz dos coeficientes na resolução dos dois sistemas lineares. No entanto, esta variante será testada futuramente: talvez a redução do número de iterações do método primal-dual leve a um ganho no tempo de cpu.

3.4 Alguns detalhes da implementação do método primal-dual

Apresentaremos aqui alguns detalhes da implementação do método primal-dual de pontos interiores, tais como os valores dos parâmetros utilizados, o ponto inicial e o critério de parada. A solução do sistema linear das direções será discutida no próximo capítulo.

3.4.1 Parâmetros η e δ

Em geral, os valores teóricos dos parâmetros η^ℓ e δ^ℓ, que aparecem em cada iteração do método primal-dual, são um pouco diferentes dos valores escolhidos nas implementações. Escolhemos $\eta^\ell = 0.1$, constante em todas as iterações do método de pontos interiores, conforme sugerido por Portugal e outros [58].

O próprio cálculo do parâmetro μ^ℓ apresenta variações. Por exemplo, no denominador da expressão de μ^ℓ, Portugal e outros [58] sugerem que se use o número total de variáveis, incluindo as variáveis de folga das canalizações. Outros autores, como Mehrotra, não incluem estas folgas. Outra possibilidade sugerida por vários autores é tomar γ^ℓ como sendo o valor do *gap de dualidade*, ao invés do valor das *folgas complementares*.

Testamos $\mu^\ell = 0.1\ \frac{\gamma^\ell}{(p+1)n}$ e $\mu^\ell = 0.1\ \frac{\gamma^\ell}{(2p+1)n}$. Como os resultados não foram muito diferentes, optamos por

$$\mu^\ell = 0.1\ \frac{\gamma^\ell}{(p+1)n} = 0.1\ \frac{\gamma^\ell}{\bar{n}}\ ,$$

onde γ^ℓ representa o valor das folgas complementares na iteração corrente, ou seja,

$$\gamma^\ell = (\hat{x}^\ell)'\ \hat{z}^\ell + (\hat{w}^\ell)'\ \hat{s}^\ell\ .$$

No teste da razão, o parâmetro δ^ℓ, $0 < \delta^\ell < 1$, também foi tomado como um número fixo, escolhido de experimentos computacionais. As sugestões mais usuais encontradas na literatura são os valores 0.9995, como, por exemplo em Li e Lustig [42] e 0.99995, como, por exemplo, em Lustig e outros [44]. De nossos experimentos, concluímos por $\delta^\ell = \delta = 0.9995$.

3.4.2 Pontos iniciais

Apresentamos aqui três pontos iniciais: desenvolvemos o primeiro deles e estendemos os outros dois para problemas de multifluxo. Os dois últimos podem ser encontrados na literatura: o segundo, para problemas de fluxo de custo mínimo com um único produto e o terceiro, para problemas gerais de programação linear. A razão de colocarmos estes três tipos de pontos iniciais foi estudarmos o comportamento do método primal-dual quando inicializado com cada um deles, medindo o número de iterações efetuadas e o tempo de execução. Os três tipos de pontos usam heurísticas para ficar "perto do centro".

Primeiro ponto:

Começamos com o ponto $(\widehat{x}^0, \widehat{s}^0, \widehat{y}^0, \widehat{w}^0, \widehat{z}^0)$, que é *dual factível* e que satisfaz

$$\widehat{x}^0 > 0\,, \quad \widehat{s}^0 > 0\,, \quad \widehat{w}^0 > 0\,, \quad \widehat{z}^0 > 0\,, \tag{3.14}$$

$$\widehat{x}^0 + \widehat{s}^0 = \widehat{u}\,, \tag{3.15}$$

e

$$\widehat{A}'\widehat{y}^0 - \widehat{w}^0 + \widehat{z}^0 = \widehat{c}\,, \tag{3.16}$$

mas pode não satisfazer

$$\hat{A}\hat{x}^0 = \hat{b}\,. \tag{3.17}$$

Note que a equação (3.16) resulta em

$$\begin{cases} A'(y^k)^0 + (y^{p+1})^0 - (w^k)^0 + (z^k)^0 = c^k\,, \quad k = 1, \cdots, p \\ (y^{p+1})^0 = (-z^{p+1})^0 \end{cases} \tag{3.18}$$

e a equação (3.17) resulta em

$$\begin{cases} A(x^k)^0 = b^k\,, \quad k = 1, \cdots, p \\ \sum_{k=1}^{p+1} (x^k)^0 = d\,. \end{cases} \tag{3.19}$$

Seja $e_t = (i, j)$ um arco da rede, $t = 1, \cdots, n$. Então, para cada bloco k, o primeiro conjunto de equações de (3.18) pode ser reescrito por:

$$(y_i^k)^0 - (y_j^k)^0 + (y_t^{p+1})^0 - (w_t^k)^0 + (z_t^k)^0 = c_{ij}^k \ , \quad k = 1, \cdots, p \ , \quad \forall (i,j) \in \mathcal{A}$$

ou

$$\underbrace{c_{ij}^k - (y_i^k)^0 + (y_j^k)^0 - (y_t^{p+1})^0}_{\bar{c}_{ij}^k} = (z_t^k)^0 - (w_t^k)^0$$

solução dual:

Seja $\widehat{y}^0$ um vetor tal que $(y^k)^0$ seja qualquer, $k = 1, \cdots, p$, mas $(y^{p+1})^0 < 0$. Para cada bloco k, $k = 1, \cdots, p$ e para cada arco $e_t = (i,j)$, $t = 1, \cdots, n$, calculamos os custos reduzidos:

$$\bar{c}_{ij}^k = c_{ij}^k - (y_i^k)^0 + (y_j^k)^0 - (y_t^{p+1})^0 \ .$$

Seja

$$M = \max_{\{k,(ij)\}} \{|\bar{c}_{ij}^k|\} + \max_t \{(|y_t^{p+1}|)^0\}, \quad k = 1, \cdots, p \ , \quad \forall (i,j) \in \mathcal{A} \ \ (M > 0) \ .$$

Assim, para garantirmos $\widehat{w}^0 > 0$ e $\widehat{z}^0 > 0$, tomamos, para $k = 1, \cdots, p$ e $\forall (i,j) \in \mathcal{A}$:

se $\bar{c}_{ij}^k > 0$, então $\begin{cases} (w_t^k)^0 = M \ ; \\ (z_t^k)^0 = \bar{c}_{ij}^k + M \ . \end{cases}$

se $\bar{c}_{ij}^k \leq 0$, então $\begin{cases} (z_t^k)^0 = M \ ; \\ (w_t^k)^0 = -\bar{c}_{ij}^k + M \ . \end{cases}$

Em nossa implementação, iniciamos com:

$$\begin{cases} (y^k)^0 = 0, \quad k = 1, \cdots, p \ ; \\ (y^{p+1})^0 = -\mathbf{1} \ , \end{cases}$$

de modo que $\bar{c}_{ij}^k = c_{ij}^k + 1$, $k = 1, \cdots, p$, $\forall (i,j) \in \mathcal{A}$.
É fácil verificar que esta solução satizfaz (3.14) e (3.16).

solução primal:

Para cada arco $e_t = (i,j)$, $t = 1, \cdots, n$, calculamos

$$\tilde{u} = \sum_{k=1}^{p} u_t^k \ .$$

Se $\tilde{u} < 2d_t$, então $(x_t^k)^0 = u_t^k/2$, $k = 1, \cdots, p$.
Senão, $(x_t^k)^0 = (u_t^k \ d_t) \ / \ 2\tilde{u}$, $k = 1, \cdots, p$.

$$(x_t^{p+1})^0 = d_t - \sum_{k=1}^{p} (x_t^k)^0 \ , \quad t = 1, \cdots, n \ .$$

$$\widehat{s}^0 = \widehat{u} - \widehat{x}^0 \ .$$

Desta maneira, além de serem satisfeitas as condições (3.14) e (3.15), é fácil verificar que as restrições de acoplamento ficam satisfeitas em (3.19), de modo que as únicas restrições que podem não estar satisfeitas são as de oferta/demanda dos produtos.

Segundo ponto:

Este ponto inicial é uma extensão do que Portugal e outros [58] apresentam para problemas de fluxo de custo mínimo com um único produto. Como no caso anterior, o ponto $(\widehat{x}^0, \ \widehat{s}^0, \ \widehat{y}^0, \ \widehat{w}^0, \ \widehat{z}^0)$ é *dual factível* e satisfaz:

$$\widehat{x}^0 > 0, \quad \widehat{s}^0 > 0, \quad \widehat{w}^0 > 0, \quad \widehat{z}^0 > 0,$$

$$\widehat{x}^0 + \widehat{s}^0 = \widehat{u},$$

e

$$\hat{A}'\widehat{y}^0 - \widehat{w}^0 + \widehat{z}^0 = \widehat{c},$$

mas não necessariamente satisfaz

$$\hat{A}\widehat{x}^0 = \widehat{b} \ .$$

Ainda, como no caso anterior, dados $(y^k)^0$ quaisquer e $(y^{p+1})^0 < 0$, calculamos os custos reduzidos:

$$\bar{c}_{ij}^k = c_{ij}^k - (y_i^k)^0 + (y_j^k)^0 - (y_t^{p+1})^0 \ ,$$

para cada arco $e_t = (i,j)$, $t = 1, \cdots, n$ e para $k = 1, \cdots, p$. E agora calculamos:

$$\lambda_1 = \lambda_2 \max_{k,(ij)} \{ |\bar{c}_{ij}^k u_{ij}^k| \ / \ (i,j) \in \mathcal{A} \} \ , \quad \lambda_2 > 0 \ .$$

$$\tilde{c}_{ij}^k = \begin{cases} 0.5 + \frac{\lambda_1}{\bar{c}_{ij}^k u_{ij}^k} - \sqrt{0.25 + (\frac{\lambda_1}{\bar{c}_{ij}^k u_{ij}^k})^2} \ , & se \ \ \bar{c}_{ij}^k > 0 \\ 0.5 + \frac{\lambda_1}{\bar{c}_{ij}^k u_{ij}^k} + \sqrt{0.25 + (\frac{\lambda_1}{\bar{c}_{ij}^k u_{ij}^k})^2} \ , & se \ \ \bar{c}_{ij}^k < 0 \\ 0.5 \ , \quad se \ \ \bar{c}_{ij}^k = 0 \ . \end{cases}$$

$\tilde{c}_{ij}^k$ é obtido quando impomos que as restrições duais sejam satisfeitas, e é possível mostrar que $0 < \tilde{c}_{ij}^k < 1$.

Para cada produto k e para cada arco (i,j), o restante da solução é dado por:

$$(x_{ij}^k)^0 = \tilde{c}_{ij}^k u_{ij}^k$$

$$(s_{ij}^k)^0 = (1 - \tilde{c}_{ij}^k) u_{ij}^k$$

$$(z_{ij}^k)^0 = \frac{\lambda_1}{\tilde{c}_{ij}^k u_{ij}^k}$$

$$(w_{ij}^k)^0 = \frac{\lambda_1}{(1 - \tilde{c}_{ij}^k) u_{ij}^k}$$

Iniciamos com $(y^k)^0 = 0, \ k = 1, \cdots, p$ e $\lambda_2 = 0.2$, conforme sugerido em [58] e ainda tomamos:

$$(y^{p+1})^0 = -1$$

$$f = d - \sum_{k=1}^{p} x^k$$

Se $f_t > 0$, então $(x_j^{p+1})^0 = f_t$
Se $f_t < 0$, então $(x_j^{p+1})^0 = -f_t$
Se $f_t = 0$, então $(x_j^{p+1})^0 = 1$.

Esta solução primal também tenta ficar "próxima ao centro". Como $\lambda_1 > 0$, garantimos que $0 < \tilde{c}_{ij}^k < 1$ e que $(x^k)^0 > 0$, $(z^k)^0 > 0$, $(w^k)^0 > 0$ e $0 < (s^k)^0 < u^k$. Novamente, as equações $Ax^k = b^k$ poderão não estar satisfeitas, nem as restrições de acoplamento (as únicas restrições de acoplamento satisfeitas são aquelas onde $f_t > 0$).

Terceiro ponto:

Este ponto inicial é uma extensão do ponto sugerido por Mehrotra [51] para problemas gerais de programação linear. Não sendo específico para redes, ele deixa de tirar proveito da estrutura de nosso problema. De qualquer maneira, este ponto requer um trabalho maior para ser construído.

Numa primeira etapa, encontramos um ponto $(\widehat{x},\ \widehat{s},\ \widehat{y},\ \widehat{w},\ \widehat{z})$ que satisfaz todas as restrições do par primal/dual, mas não necessariamente as restrições de não negatividade:

$$\widehat{y} = (\hat{A}\hat{A}')^{-1}\hat{A}\hat{c}$$

$$\widehat{z} = 0.5\ (\widehat{c} - \hat{A}'\widehat{y})$$

$$\widehat{w} = -\widehat{z}$$

$$\widehat{x} = 0.5\widehat{u} - \hat{A}'(\hat{A}\hat{A}')^{-1}(0.5\hat{A}\widehat{u} - \widehat{b})$$

$$\widehat{s} = \widehat{u} - \widehat{x}$$

Notamos que a construção de $\widehat{y}$ requer a resolução de um sistema linear com $\widehat{m}$ equações e variáveis. Este sistema é resolvido pelo método dos gradientes conjugados sem precondicionamento, já que sua matriz é $\hat{A}\hat{A}'$. Mas é necessário construir o lado direito $\hat{A}\hat{c} = (Ac^1, \cdots, Ac^p, \sum_{k=1}^{p} c^k)$. A construção de $\widehat{z}$ também envolve um produto matriz-vetor: $\hat{A}'\widehat{y} = (A'y^1 + y^{p+1}, \cdots, A'y^p + y^{p+1}, y^{p+1})$. Finalmente, a construção de $\widehat{x}$ envolve a resolução de outro sistema linear com a matriz $\hat{A}\hat{A}'$ (novamente por gradiente conjugado sem precondicionamento), além de um produto matriz-vetor na construção do lado direito deste sistema.

Numa segunda etapa, é feita uma translação adequada do ponto encontrado, de modo a satisfazer as restrições de não negatividade. Para isso, são usados os parâmetros α_1 e α_2. Calculamos:

$$\beta_P = \max\{-\alpha_1 \min_{i,k}\{x_i^k\},\ -\alpha_1 \min_{i,k}\{s_i^k\},\ 0.01\}$$

$$\beta_D = \max\{-\alpha_1 \min_{i,k}\{z_i^k\},\ -\alpha_1 \min_{i,k}\{w_i^k\},\ 0.01\}$$

$$\delta_P = \beta_P + \alpha_2 \frac{(\widehat{x} + \beta_P\mathbf{1})'\ (\widehat{z} + \beta_D\mathbf{1}) + (\widehat{s} + \beta_P\mathbf{1})'\ (\widehat{w} + \beta_D\mathbf{1})}{(\widehat{z} + \beta_D\mathbf{1})'\ \mathbf{1} + (\widehat{w} + \beta_D\mathbf{1})'\ \mathbf{1}}$$

$$\delta_D = \beta_D + \alpha_2 \frac{(\widehat{x} + \beta_P \mathbf{1})' (\widehat{z} + \beta_D \mathbf{1}) + (\widehat{s} + \beta_P \mathbf{1})' (\widehat{w} + \beta_D \mathbf{1})}{(\widehat{x} + \beta_P \mathbf{1})' \mathbf{1} + (\widehat{s} + \beta_P \mathbf{1})' \mathbf{1}}$$

onde $\mathbf{1} = (1, \cdots, 1)' \in I\!R^{\widehat{n}}$.

Finalmente,
$\widehat{y}^0 = \widehat{y}$
$\widehat{x}^0 = \widehat{x} + \delta_P \mathbf{1}$
$\widehat{s}^0 = \widehat{s} + \delta_P \mathbf{1}$
$\widehat{z}^0 = \widehat{z} + \delta_D \mathbf{1}$
$\widehat{w}^0 = \widehat{w} + \delta_D \mathbf{1}$

É possível mostrar que $\widehat{x}^0$, $\widehat{s}^0$, $\widehat{w}^0$, $\widehat{z}^0$ são estritamente positivos, devido ao cálculo de β_P e β_D anteriores. No entanto, as outras restrições primais/duais não ficam necessariamente satisfeitas. Neste sentido, este ponto se difere bastante dos dois pontos iniciais anteriores. Como sugerido em [51], tomamos $\alpha_1 = 1.5$ e $\alpha_2 = 0.5$. Entretanto, podemos aumentar o valor de α_2 para conseguirmos um ponto "mais próximo ao centro".

Realizamos vários testes computacionais, com problemas de várias dimensões, utilizando um gerador de problemas de multifluxo, para comparar o comportamento do método primal-dual diante destes três pontos iniciais. Para os problemas testados, concluímos que o terceiro ponto, que é mais elaborado, não produz um melhor comportamento — o número de iterações efetuadas pelo método primal-dual não é menor do que nos outros dois e o tempo de execução é um pouco maior. De fato, a variação do número de iterações do método primal-dual é de até duas iterações apenas (para mais ou para menos), para o conjunto de testes realizados com os três pontos iniciais. Embora estes três pontos usem heurísticas para ficar "perto do centro", a infactibilidade gerada pelo terceiro ponto faz com que o método primal-dual efetue um esforço computacional maior por iteração: diferentemente dos dois primeiros pontos iniciais, a canalização não é respeitada ($r_c \neq 0$) e as restrições duais não são satisfeitas ($r_d \neq 0$) . Além disso, é necessária a resolução de dois sistemas lineares extras (obtenção de $\widehat{x}$ e de $\widehat{y}$) . Assim, optamos pelo primeiro ponto, que é mais simples e que produziu praticamente os mesmos resultados que o segundo ponto no método primal-dual de pontos interiores.

3.4.3 Critério de parada

O critério de parada implementado é baseado nas condições de otimalidade do problema: as factibilidades primal e dual devem estar satisfeitas e o gap de dualidade, as folgas complementares e o parâmetro de penalidade devem ser "zero".

Sejam $\widehat{x}$, $\widehat{s}$, $\widehat{y}$, $\widehat{w}$, $\widehat{z}$ os pontos obtidos na iteração corrente do método primal-dual de pontos interiores (estamos omitindo aqui o índice desta iteração). Cada um destes pontos está particionado em $p+1$ blocos: p blocos ($k = 1, \cdots, p$) mais o acoplamento (bloco $p+1$).

Para a factibilidade (relativa), calculamos:

$$M_p = \max_{k=1,\cdots,p} \{ \frac{||b^k - Ax^k||_\infty}{1 + ||x^k||_\infty} \}$$

$$M_d = \max_{k=1,\cdots,p} \{ \frac{||c^k - A'y^k - y^{p+1} + w^k - z^k||_\infty}{1 + ||y^k||_\infty + ||y^{p+1}||_\infty + ||w^k||_\infty + ||z^k||_\infty} \}$$

$$M_a = \frac{||d - \sum_{k=1}^{p+1} x^k||_\infty}{1 + \sum_{k=1}^{p+1} ||x^k||_\infty}$$

$$M_c = \max_{k=1,\cdots,p} \{ \frac{||u^k - x^k - s^k||_\infty}{1 + ||x^k||_\infty + ||s^k||_\infty} \}$$

e

$M_f = \max\{M_p,\ M_d,\ M_a,\ M_c\}$.

Se $M_f < \varepsilon_f$, consideramos a factibilidade satisfeita. Em geral, tomamos $\varepsilon_f = 10^{-5}$.

Para o parâmetro de penalidade e o gap (relativo), verificamos se:

$\mu = \eta\ \gamma\ /\ \widehat{n} \ < \ \varepsilon_\mu$
$gap = \gamma\ /\ (1 + |\ \widehat{c}'\widehat{x} - \widehat{b}'\widehat{y} + \widehat{u}'\widehat{w}\ |\ ) \ < \ \varepsilon_g$
onde $\gamma = (\widehat{x})'\ \widehat{z} + (\widehat{w})'\ \widehat{s}$.

Em geral, tomamos $\varepsilon_\mu = 10^{-7}$ e $\varepsilon_g = 10^{-5}$.

Como salvaguardas, limitamos o número de iterações a um máximo de 50 e, para detectar problemas infactíveis, limitamos também o valor da função

objetivo primal a um mínimo de -10^{20}.

No próximo capítulo, abordaremos a solução do sistema linear das direções.

Capítulo 4

Sistema Linear das Direções

Vimos que, em cada iteração do Algoritmo 3.1, é necessário resolver um sistema linear para o cálculo das direções. Aliás, em todos os algoritmos baseados em métodos de pontos interiores, este é o passo que demanda maior esforço computacional, e podemos dizer que grande parte da eficiência do algoritmo está em construir e resolver bem este sistema. Embora tenhamos trabalhado com o sistema reduzido, isto é, as variáveis que aparecem são apenas $\Delta\widehat{y}$, poderia também ser utilizado o sistema aumentado, onde as variáveis são $\Delta\widehat{y}$ e $\Delta\widehat{x}$ (ver Oliveira [56]).

Primeiramente, vamos discutir a estrutura de nosso sistema.

4.1 Estrutura do sistema

O sistema linear (3.9) é um sistema quadrado de dimensão $\widehat{m}$ dado por:

$$\widehat{Q}\ \Delta\widehat{y} = \widehat{q}$$

onde,

$$\widehat{Q} = \widehat{A}\widehat{\Theta}\widehat{A}' = \left[\begin{array}{cccc|c} A\Theta^1A' & & & & A\Theta^1 \\ & A\Theta^2A' & & & A\Theta^2 \\ & & \ddots & & \vdots \\ & & & A\Theta^pA' & A\Theta^p \\ \hline \Theta^1A' & \Theta^2A' & \cdots & \Theta^pA' & \sum_{k=1}^{p+1}\Theta^k \end{array}\right] = \left[\begin{array}{c|c} B_Q & C \\ \hline C' & D \end{array}\right]$$

$$\Delta\hat{y} = \begin{bmatrix} \Delta y^1 \\ \Delta y^2 \\ \vdots \\ \Delta y^p \\ \hline \Delta y^{p+1} \end{bmatrix} = \begin{bmatrix} \Delta y \\ \hline \Delta y^{p+1} \end{bmatrix} ; \qquad \hat{q} = \begin{bmatrix} q^1 \\ q^2 \\ \vdots \\ q^p \\ \hline q^{p+1} \end{bmatrix} = \begin{bmatrix} q \\ \hline q^{p+1} \end{bmatrix} ;$$

com q e $\Delta y \in \mathbb{R}^{p(m-1)}$; q^{p+1} e $\Delta y^{p+1} \in \mathbb{R}^n$.

A matriz $\hat{Q}$ possui uma estrutura bloco diagonal dada pela matriz $B_Q \in \mathbb{R}^{p(m-1)\times p(m-1)}$, com uma faixa horizontal e outra vertical dadas pelas matrizes C' e C, respectivamente, com $C \in \mathbb{R}^{p(m-1)\times n}$ e ainda uma matriz diagonal $D \in \mathbb{R}^n$. Cada Θ^k é uma matriz diagonal e A é a matriz de incidência nó-arco associada à rede em questão. É importante observar que $\hat{q}$ e Θ^k dependem da solução atual e que, em cada iteração do método primal-dual, $\hat{Q}$ é alterada apenas pela matriz $\hat{\Theta}$. Como será comentado mais tarde, quando as soluções geradas pelo método primal-dual se aproximam de uma solução ótima, as componentes de cada Θ^k tendem a zero ou a infinito.

Sob diversas condições teóricas — nem sempre satisfeitas na prática — considera-se $\hat{Q}$ *simétrica e definida positiva*, pois, tanto B_Q quanto D são simétricas e definidas positivas.

Além disso, $\hat{Q}$ é uma matriz *não singular*, pois é calculada a partir da matriz A , que é a matriz de incidência de uma rede com m nós e n arcos e cada Θ^k é estritamente positiva. O número de linhas linearmente dependentes de A é exatamente o número de componentes da rede. Se a rede for conectada, A possui apenas uma linha linearmente dependente. Em qualquer caso, como as linhas redundantes podem ser retiradas inicialmente, podemos considerar que A possui posto completo (sem perda de generalidade, estamos considerando posto$(A) = m - 1$) .

É importante ressaltar que o sistema linear (3.9) constitui uma aproximação de um sistema não linear que deve ser resolvido a cada iteração do método primal-dual. É comum, na prática, o uso de uma solução aproximada para o sistema não linear. Soluções aproximadas obtidas resolvendo o sistema linear de modo exato são consideradas satisfatórias.

Em problemas gerais de programação linear têm sido utilizadas duas estratégias para resolver o sistema linear das direções em métodos de pontos interiores: a *fatoração de Cholesky*, como, por exemplo, em [1, 2, 43, 49] e o método dos *gradientes conjugados* com *precondicionadores*, como, por exemplo, em [44, 52].

Comparações entre estas duas técnicas (entre outras) foram feitas, por exemplo, por Resende e Veiga [61] e por Portugal e outros [58] para *problemas de fluxo de custo mínimo* com um único produto, usando alguns códigos e o método *dual-afim*. Estas comparações mostraram que, com um bom precondicionador, o uso dos gradientes conjugados é mais vantajoso: o tempo de cpu é menor e esta vantagem aumenta com o tamanho do problema; as direções obtidas por gradientes conjugados (aproximadas) não alteram o número de iterações do primal-dual e a quantidade de memória utilizada por gradientes conjugados é menor. Ainda em Resende e Veiga [60, 63] e considerando *problemas de fluxo de custo mínimo* com um único produto e o método *dual-afim*, é mostrado que, embora a matriz A seja *esparsa* (apenas dois elementos diferentes de zero por coluna), a fatoração da matriz $A\Theta^k A'$ pode produzir um "*fill-in*" considerável.

Passamos a descrever as implementações destas duas técnicas.

4.2 Fatoração de Cholesky

Iniciamos as implementações resolvendo o sistema (3.9) pela *fatoração de Cholesky*. De fato, a matriz $\hat{Q}$ admite esta fatoração, pois é simétrica e definida positiva. Num primeiro passo, $\hat{Q}$ é transformada em uma matriz triangular superior, de maneira que os dois sistemas lineares seguintes são equivalentes:

$$\left[\begin{array}{c|c} B_Q & C \\ \hline C' & D \end{array}\right] \left[\begin{array}{c} \Delta y \\ \hline \Delta y^{p+1} \end{array}\right] = \left[\begin{array}{c} q \\ \hline q^{p+1} \end{array}\right]$$

$$\left[\begin{array}{c|c} B_Q & C \\ \hline 0 & D - C'B_Q^{-1}C \end{array}\right] \left[\begin{array}{c} \Delta y \\ \hline \Delta y^{p+1} \end{array}\right] = \left[\begin{array}{c} q \\ \hline q^{p+1} - C'B_Q^{-1}q \end{array}\right] \qquad (4.1)$$

Resolver (4.1) significa resolver os sistemas lineares:

$$(D - C'B_Q^{-1}C)\ \Delta y^{p+1} = q^{p+1} - C'B_Q^{-1}q \qquad (4.2)$$

$$B_Q \ \Delta y = q - C\Delta y^{p+1} \tag{4.3}$$

A matriz $(D - C'B_Q^{-1}C)$ é conhecida como *complemento de Schur* e será denotada por S. Note que

$$S = D - C'B_Q^{-1}C = \sum_{k=1}^{p+1} \Theta^k - \sum_{k=1}^{p} [\ \Theta^k A' \ (A\Theta^k A')^{-1} \ A\Theta^k \ ] \tag{4.4}$$

e

$$q^{p+1} - C'B_Q^{-1}q = q^{p+1} - \sum_{k=1}^{p} [\ \Theta^k A' \ (A\Theta^k A')^{-1} \ q^k \ ]$$

Assim, para resolver o sistema linear (4.1), resolvemos primeiro o sistema linear (4.2) encontrando Δy^{p+1} e utilizamos este resultado para resolver o sistema (4.3).

O sistema linear (4.3), cuja matriz dos coeficientes é B_Q, pode ser resolvido fazendo-se separadamente a fatoração de Cholesky em cada bloco, pois, A tem posto (linha) completo e as matrizes $A\Theta^k A'$, $k = 1, \cdots, p$ são simétricas e definidas positivas.

Em outras palavras, este sistema se transforma em p sistemas lineares de dimensão $m-1$. Como cada um destes sistemas menores é transformado em dois sistemas triangulares, acabamos resolvendo $2p$ sistemas triangulares de dimensão $m-1$.

Vamos nos deter na construção de cada uma das matrizes $A\Theta^k A'$, $k = 1, \cdots, p$. Já vimos que a rede G (e, consequentemente, a matriz A) pode ser armazenada nos vetores *orig* e *dest*. Seja $e_t = (i, j)$ um arco de G e a_{it} um elemento de A. Então, $a_{it} = +1$ se $orig[t] = i$ e $a_{jt} = -1$ se $dest[t] = j$. Assim, cada matriz $Q^k = A\Theta^k A'$ pode ser obtida com facilidade, como soma de matrizes de posto um. Se a^t é a coluna de A correspondente ao arco e_t, temos que $Q^k = A\Theta^k A' = \sum_{t=1}^{n} a^t \Theta_t^k (a^t)'$, e Q^k pode ser obtida percorrendo os arcos da rede, em $\mathcal{O}(n)$ somas ou subtrações, com o algoritmo:

Algoritmo 4.1 (*constrói* $Q^k = A\Theta^k A'$)

$Q^k \leftarrow 0$;
`for` $arco \leftarrow 1..n$ `do`
`begin`
$i \leftarrow orig[arco]$;
$j \leftarrow dest[arco]$;
$Q^k[i,i] \leftarrow Q^k[i,i] + \Theta^k[arco]$;
$Q^k[j,j] \leftarrow Q^k[j,j] + \Theta^k[arco]$;
`if` $i > j$ `then` $Q^k[j,i] \leftarrow Q^k[j,i] - \Theta^k[arco]$
`else` $Q^k[i,j] \leftarrow Q^k[i,j] - \Theta^k[arco]$;
`endfor`

Note que, como Q^k é simétrica, construímos apenas sua parte triangular superior. Além disso, o Algoritmo 4.1 não considera a eliminação de uma linha de A (ou de uma linha e a coluna correspondente em Q^k).

Um fato interessante é que, se $\Theta^k = I$, um elemento da diagonal $Q^k[i,i]$ é igual ao grau do nó i correspondente; um elemento fora da diagonal, $Q^k[i,j] = Q^k[j,i]$ é igual ao número de arcos ligando os nós i e j, com sinal negativo.

O sistema linear (4.2) é mais difícil de ser resolvido, devido à sua matriz de coeficientes, S ; $S \in I\!R^{n \times n}$ (em geral $n >> m$) é simétrica mas densa, fazendo com que a solução de (4.2) por um método direto não seja viável. Além disso, a obtenção de S é computacionalmente cara.

Numa primeira implementação de nosso estudo, usamos a inversa do fator de Cholesky de cada bloco para construir a matriz S , segundo o esquema abaixo. Os passos (i) até (iv) são efetuados para cada produto k, $k = 1, \cdots, p$:

(i) obter a fatoração de Cholesky de cada bloco: $A\Theta^k A' = G_k (G_k)'$;

(ii) calcular a matriz inversa G_k^{-1} ;

(iii) calcular $H^k = (G_k^{-1})' \, G_k^{-1}$;

(iv) calcular $\bar{H}^k = \Theta^k A' \, (A\Theta^k A')^{-1} \, A\Theta^k = \Theta^k A' \, H^k \, A\Theta^k$;

(v) acumular, por blocos, a matriz $S = \sum_{k=1}^{p+1} \Theta^k - \sum_{k=1}^{p} \bar{H}^k$.

Desta maneira, diminuímos o esforço computacional do cálculo de S, uma vez que para um dado produto k, G_k^{-1} é triangular e é obtida por uma fórmula fechada em ordem de $\mathcal{O}(m^2)$ produtos e somas e $\bar{H}^k$ é calculada em $\mathcal{O}(n^2)$ produtos e somas, como mostramos a seguir.

Sejam g_{ij}^k os elementos de G_k (G_k é triangular inferior) e δ_{ij}^k os elementos de G_k^{-1}. Então,

$$\delta_{ii}^k = 1/g_{ii}^k \ , \quad i = 1, \cdots, m$$

$$\delta_{ij}^k = -\sum_{l=j}^{i-1} (g_{il}^k \ g_{lj}^k) \ / \ g_{ii}^k \ , \quad i = 1, \cdots, m \ ; \quad j = 1, \cdots, i$$

Sejam $H^k[i,j]$ os elementos da matriz simétrica $H^k = (G_k^{-1})' G_k^{-1} = (G_k G_k')^{-1}$. Supondo que a linha m de A foi eliminada, vamos supor zeradas a linha e a coluna m de H^k. Então, os elementos de cada matriz simétrica $\bar{H}^k \in \mathbb{R}^{n \times n}$, $\bar{H}^k = (\Theta^k A') \ H^k \ (A\Theta^k)$ podem ser obtidos pelo algoritmo:

Algoritmo 4.2 (*obtenção de* $\bar{H}^k = (\Theta^k A') \ H^k \ (A\Theta^k)$)

```
for i ← 1..n do
   begin
      i1 ← orig[i] ;
      j1 ← dest[i] ;
      for j ← i..n do
        begin
          i2 ← orig[j] ;
          j2 ← dest[j] ;
          H̄^k[i,j] ← Θ^k_j Θ^k_i (H^k[i1,i2]−H^k[j1,i2]−H^k[i1,j2]+H^k[j1,j2] ) ;
        endforj;
   endfori
```

Obs.: neste algoritmo, não estamos fazendo distinção entre os elementos $H^k[i,j]$ ou $H^k[j,i]$, já que a matriz H^k é simétrica.

Obtida S, resolvemos um sistema linear $n \times n$ para a obtenção de Δy^{p+1} por decomposição LU; corrigimos os lados direitos dos blocos e resolvemos os sistemas triangulares para obtermos Δy^k, $k = 1, \cdots, p$ (de fato, é possível mostrar que S é definida positiva, de modo que a fatoração de Cholesky pode ser utilizada). O problema desta implementação foi o gasto excessivo de memória, o que nos obrigou a trabalhar com problemas pequenos. Embora tenhamos tirado proveito do fato das matrizes envolvidas serem simétricas — cada uma delas foi armazenada num vetor (por linha ou por coluna, dependendo da conveniência) — são várias matrizes e, mais ainda, uma para cada produto. Um próximo passo seria a implementação considerando técnicas de esparsidade na construção de cada fator G_k, de modo a preservar a estrutura bloco-angular do problema. Isto não foi feito. De fato, Resende e Veiga [60] já haviam mostrado que estes fatores não são esparsos, mesmo quando a matriz A é esparsa.

Em uma segunda implementação, foi feita a fatoração de Cholesky no sistema (4.1) de maneira global, isto é, a fatoração efetuada em cada bloco da matriz B_Q teve seu pivoteamento estendido para as faixas C e C', para o lado direito $\hat{q}$ e para a matriz diagonal D. A resolução dos p subsistemas é idêntica à implementação anterior. Novamente, o gasto com memória foi excessivo (uma das faixas — a horizontal ou a vertical — precisa ser armazenada), inviabilizando a resolução de problemas com dimensões maiores. Armazenamos cada fator G_k e cada inversa $(G_k)^{-1}$, embora de cada uma delas tenha sido armazenado a metade dos elementos mais a diagonal. Na vetorização destas matrizes, foram feitas algumas variações quanto ao seu armazenamento, uma vez que estes fatores podem ser armazenados por linhas ou por colunas, ou ainda, a parte triangular inferior (G_k) ou a parte triangular superior (G_k').

Com o objetivo de considerar técnicas de esparsidade, exibimos (para problemas pequenos obtidos com o gerador de problemas de multifluxo) a estrutura das matrizes $A\Theta^k A'$, G_k e $A\Theta^k$ — antes e depois dos pivoteamentos. Pudemos concluir claramente que $A\Theta^k A'$ e G_k são matrizes densas (embora A seja esparsa) e que, embora $A\Theta^k$ seja esparsa, depois dos pivoteamentos ela também se torna densa. Dadas as sugestões da literatura quanto ao uso do método dos gradientes conjugados nestes casos, decidimos não investir em técnicas de esparsidade.

Como exemplo, podemos observar na Figura 4.1 uma matriz de incidência A (40×240), com 480 elementos não nulos, obtida através do gerador de problemas de multifluxo. Sua porcentagem de esparsidade (número de elementos não nulos $\times$ 100 / número total de elementos) é de 5%. No entanto, a matriz AA' (40×40) possui 454 elementos não nulos, com uma porcentagem de esparsidade de 28.38%. Além disso, os elementos não nulos de A são apenas $+1$ e -1 (dois elementos não nulos por coluna), enquanto que, ao trabalharmos com $A\Theta^k A'$ temos 454 elementos quaisquer.

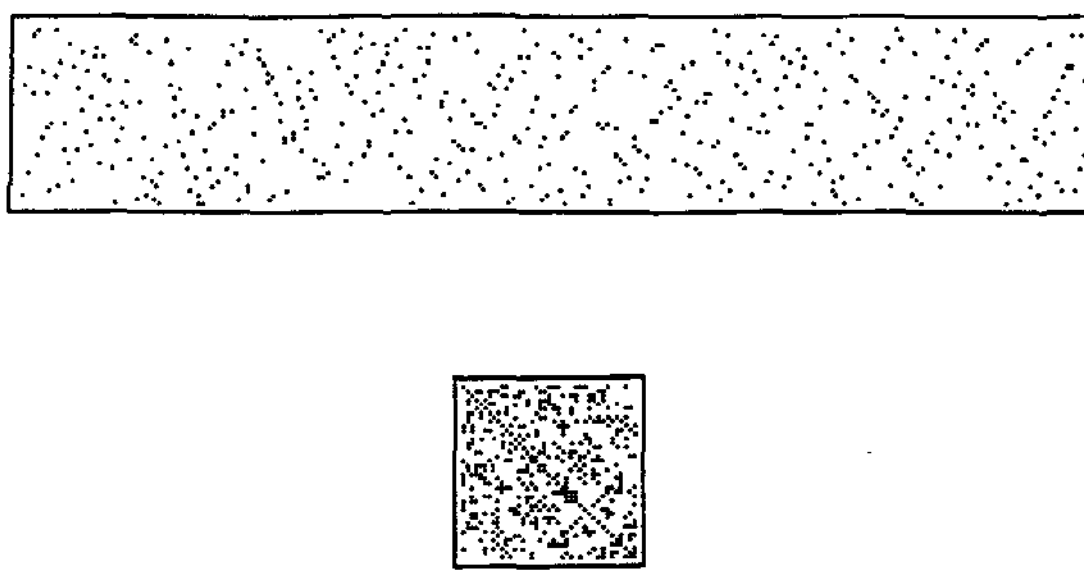


Figura 4.1: *Elementos não nulos de* A e AA'

Li e Lustig [42] fizeram uma implementação do método primal-dual utilizando processadores paralelos na construção da matriz S e resolvendo todos os subsistemas de (4.3) e mais o sistema (4.2) por fatoração de Cholesky.

Castro [10] mostra que a matriz S torna-se completamente densa, afirmando ser proibitivo o uso de um método direto para a resolução de (4.2). Ele resolve este sistema por gradientes conjugados (precondicionado), mas resolve os subsistemas de (4.3) por fatoração de Cholesky esparsa (heurística "minimum degree" para a obtenção de uma permutação de nós da rede).

Como observação, a variante *preditor-corretor* é bastante usada com a fatoração de Cholesky, uma vez que este fator pode ser armazenado para ser utilizado nos dois sistemas lineares envolvidos. Acreditamos que nesta variante não seja comum o uso do método dos *gradientes conjugados*, uma vez que precisaríamos aplicar este método duas vezes. Por isso, não utilizamos a variante preditor-corretor em nosso estudo.

Cabe notar que, em todas as implementações, exploramos as facilidades de trabalhar com uma matriz de incidência nó-arco. Por exemplo, se v é um vetor de dimensão apropriada e D é uma matriz diagonal com elementos d_j, produtos do tipo Av, $A'v$, ADv, ou $DA'v$ podem ser efetuados com facilidade quando percorremos os arcos da rede. Os dois primeiros produtos são efetuados em $\mathcal{O}(n)$ somas ou subtrações e os dois últimos, em $\mathcal{O}(n)$ somas (ou subtrações) e produtos. Apresentamos, a seguir, o cálculo dos dois últimos. Neles, estamos supondo que a linha m da matriz A foi eliminada.

produto $g = ADv$:

```
g ← 0 ;
for arco ← 1..n do
  begin
    i ← orig[arco] ;
    j ← dest[arco] ;
    g[i] ← g[i] + d[arco] * v[arco] ;
    g[j] ← g[j] − d[arco] * v[arco] ;
  endfor;
g[m] ← 0
```

produto $g = DA'v$:

```
v[m] ← 0 ;
for arco ← 1..n do
  begin
    i ← orig[arco] ;
    j ← dest[arco] ;
    g[arco] ← d[arco] * (v[i] − v[j]) ;
  endfor
```

Dado o excesso de memória utilizada, o que inviabilizou nossas implementações com a fatoração de Cholesky, consideramos, nas demais implementações, o uso de **métodos iterativos**.

4.3 Método dos gradientes conjugados

Os métodos iterativos, além de preservarem a estrutura original do problema (como a matriz Q é esparsa, isto implica numa quantidade menor de memória utilizada), apresentam um esforço computacional por iteração pequeno, uma vez que apenas efetuam produtos matriz-vetor. Além disso, são fáceis de serem implementados, mas são viáveis somente quando seu número de iterações não for grande.

Convém salientar que, dentre os métodos iterativos, foi feita uma tentativa com o método de *Gauss-Seidel* aplicado ao sistema (3.9). Embora a convergência deste processo seja teoricamente garantida ($\widehat{Q} = \widehat{A}\widehat{\Theta}\widehat{A}'$ é simétrica e definida positiva), esta convergência se deu de maneira extremamente lenta, mesmo considerando problemas pequenos. O esquema implementado foi o seguinte:

(i) $\ell \leftarrow 0$; fornecer $(\Delta y^{p+1})^{\ell}$;

(ii) resolver p sistemas lineares:
$(A\Theta^k A')\ \Delta y^k = q^k - A\Theta^k\ (\Delta y^{p+1})^{\ell}\ ,\quad k = 1,\ \cdots,\ p$;

(iii) corrigir Δy^{p+1} resolvendo o sistema diagonal:
$(\ \sum_{k=1}^{p+1}\ \Theta^k)\ (\Delta y^{p+1})^{\ell+1} = q^{p+1} - \sum_{k=1}^{p}(\Theta^k A')\ \Delta y^k$;

(iv) $\ell \leftarrow \ell + 1$;
repetir (ii) e (iii) até que um critério de parada seja satisfeito.

Consideramos então o método dos *gradientes conjugados* com o uso de *precondicionadores* — o mais recomendado na literatura para sistemas definidos positivos. Este método apresenta convergência rápida se for usado um bom precondicionador, que se torna mais necessário à medida que o método primal-dual de aproxima de uma solução ótima do problema, já que a matriz $\hat{Q}$ se torna muito mal-condicionada.

Vamos então discutir um pouco sobre precondicionadores.

4.3.1 Introduzindo precondicionadores

O método dos gradientes conjugados, teoricamente, fornece a solução do sistema linear (3.9) após a realização de $\widehat{m}$ iterações, onde $\widehat{m} = p(m-1)+n$ é a dimensão do sistema (supondo a rede conectada). No entanto, como os números são armazenados em palavras com um número fixo de bits, o mau condicionamento da matriz pode prejudicar os cálculos, ampliando os efeitos dos erros de arredondamento, de modo que o método pode convergir muito lentamente ou mesmo não convergir. Cabe dizer também que, se $\widehat{m}$ for grande, $\widehat{m}$ não é um número de iterações aceitável: a literatura considera satisfatório um número de iterações da ordem de $\mathcal{O}(\sqrt{\widehat{m}})$.

Uma medida importante da sensibilidade do sistema é dada pelo *número de condição* $\kappa(\hat{Q})$ da matriz $\hat{Q}$, definido por $\kappa(\hat{Q}) = ||\hat{Q}||\ ||\hat{Q}^{-1}||$. Se a matriz for singular, dizemos que seu número de condição é infinito. Se $\kappa(\hat{Q})$ for um número grande, dizemos que $\hat{Q}$ é *mal condicionada*. Note que $\kappa(\hat{Q})$ depende da norma utilizada. Se a norma utilizada for a norma-2, denotaremos o número de condição por $\kappa_2(\hat{Q})$, que, neste caso, é dado pela razão entre o maior e o menor autovalor de $\hat{Q}$.

Através desta definição, podemos observar que o mau condicionamento do sistema ocorre quando os autovalores da matriz se distribuem num intervalo muito grande ou quando existem autovalores muito próximos de zero.

Golub e Van Loan [19] e Axelsson e Barker [6] apresentam resultados de convergência para o método dos gradientes conjugados envolvendo o número

de condição da matriz $\hat{Q}$ do sistema. Entre eles, um resultado apresentado é um limitante para o valor do erro, dado por:

$$||\Delta\hat{y} - (\Delta\hat{y})^{\ell}||_{\hat{Q}} \leq 2||\Delta\hat{y} - (\Delta\hat{y})^{0}||_{\hat{Q}} \left(\frac{\sqrt{\kappa_2(\hat{Q})} - 1}{\sqrt{\kappa_2(\hat{Q})} + 1}\right)^{\ell}$$

onde $(\Delta\hat{y})^{\ell}$ é a direção obtida na ℓ- ésima iteração do gradiente conjugado, $(\Delta\hat{y})^{0}$ é a direção inicial e $||v||_{\hat{Q}}^2 \equiv v'\hat{Q}v$.

Daqui observamos que o método dos gradientes conjugados é eficiente se $k_2(\hat{Q}) \approx 1$. Em outras palavras, o método dos gradientes conjugados trabalha bem com matrizes que são bem condicionadas ou que possuem poucos autovalores distintos. Por isso, uma técnica muito empregada é transformar o sistema original em um sistema equivalente cuja matriz possua um melhor condicionamento. Entretanto, os cálculos envolvendo a matriz do novo sistema não devem ser caros.

É importante acrescentar que, quando resolvemos o problema do multifluxo pelo método primal-dual de pontos interiores, a matriz $\hat{Q}$ do sistema linear (3.9) é modificada em cada iteração pela matriz $\hat{\Theta}$. Lembrando que $\hat{\Theta}$ é formada por $p+1$ matrizes diagonais Θ^k cujos elementos são dados em (3.11) e (3.12), podemos verificar que, à medida que a solução fornecida pelo método primal-dual se aproxima de uma solução ótima, os elementos de cada Θ^k se aproximam de zero ou de infinito, causando um mau condicionamento na matriz $\hat{Q}$ (o índice k se refere a um bloco):

$$\Theta_j^{p+1} = x_j^{p+1} \;/\; z_j^{p+1} \;, \quad j = 1, \cdots, n$$

se $x_j^{p+1} > 0$, então $z_j^{p+1} \approx 0 \Rightarrow \Theta_j^{p+1} \approx \infty$;
se $z_j^{p+1} > 0$, então $x_j^{p+1} \approx 0 \Rightarrow \Theta_j^{p+1} \approx 0$

$$\Theta_j^k = (x_j^k s_j^k) \;/\; (s_j^k z_j^k + x_j^k w_j^k) \;, \quad k = 1, \cdots, p, \quad j = 1, \cdots, n$$

se $x_j^k \approx 0$, então $s_j^k \approx u_j^k \Rightarrow \Theta_j^k \approx 0$;
se $x_j^k \approx u_j^k$, então $s_j^k \approx 0 \Rightarrow \Theta_j^k \approx 0$;
se $0 < x_j^k < u_j^k$, então $z_j^k \approx 0$ e $w_j^k \approx 0 \Rightarrow \Theta_j^k \approx \infty$.

Considerando então o sistema linear simétrico e definido positivo $\hat{Q}\ \Delta\hat{y} = \hat{q}$, construímos o sistema linear equivalente

$$P^{-1}\hat{Q}(P^{-1})'\ \tilde{\Delta}\hat{y} = \tilde{q}$$

onde $\tilde{\Delta}\hat{y} = P'\ \Delta\hat{y}$ e $\tilde{q} = P^{-1}\hat{q}$.

Este novo sistema é chamado de *sistema transformado* ou *sistema precondicionado* e a matriz $\tilde{Q} = P^{-1}\hat{Q}(P^{-1})'$ é chamada de *matriz precondicionada*. A matriz P é chamada de *matriz precondicionadora* ou, simplesmente, *precondicionador*. Em geral, o precondicionador é uma matriz simétrica e definida positiva, embora outros tipos de precondicionadores também possam ser considerados. Se $\hat{Q}$ é simétrica, então $\tilde{Q}$ também é simétrica.

Para que a convergência do método dos gradientes conjugados seja mais rápida, $\tilde{Q}$ deve estar "mais perto" da matriz identidade do que $\hat{Q}$. E a nova distribuição dos autovalores de $\tilde{Q}$ deve ser tal que a distância entre eles diminui — desta maneira, a razão entre o maior e o menor deles também diminui. Por isso, um bom precondicionador deve possuir as seguintes propriedades:

(i) $\kappa(\tilde{Q})$ deve ser bem menor do que $\kappa(\hat{Q})$;

(ii) os cálculos adicionais no método dos gradientes conjugados devem ser efetuados com facilidade, isto é, sistemas lineares do tipo $PP'r = g$ devem ser resolvidos mais facilmente do que $\hat{Q}\ \Delta\hat{y} = \hat{q}$;

(iii) dependendo do precondicionador, é desejável que o padrão de esparsidade de $\tilde{Q}$ seja bastante parecido com o padrão de esparsidade de $\hat{Q}$.

Com o precondicionamento, resolvemos um sistema linear envolvendo a matriz P em cada iteração do método dos gradientes conjugados. Por isso, o precondicionador deve ser razoavelmente simples e deve procurar respeitar as propriedades anteriores, sendo esta a razão de se usar freqüentemente precondicionadores diagonais ou triangulares. Por outro lado, precondicionadores muito simples podem não ser efetivos, isto é, podem não conseguir diminuir suficientemente o número de condição da matriz, e a ordem de convergência do método dos gradientes conjugados fica comprometida. Desta maneira,

muitas vezes criamos um impasse: os precondicionadores mais efetivos são mais caros de serem construídos. Concluindo, um bom precondicionador, além de diminuir o número de iterações do método dos gradientes conjugados, não deve ser tão difícil de ser construído, a ponto de comprometer o tempo total de execução. No entanto, quando há necessidade de se utilizar um precondicionador mais caro, podemos mantê-lo durante algumas iterações do método primal-dual.

Antes de discutirmos os precondicionadores propriamente, vamos colocar um resumo do método dos gradientes conjugados.

4.3.2 O método do gradiente conjugado precondicionado

Como o método dos gradientes conjugados precondicionado está bastante discutido na literatura, vamos colocar aqui apenas seu algoritmo. Para maiores detalhes ver, por exemplo, Golub e Van Loan [19] e Axelsson e Barker [6].

Sejam $\hat{Q}$ e $\tilde{Q} = P^{-1}\hat{Q}(P^{-1})'$ matrizes simétricas e positivas definidas e sejam $\tilde{\Delta}\hat{y} = P'\,\Delta\hat{y}$ e $\tilde{q} = P^{-1}\hat{q}$.

Neste algoritmo, consideramos:
Sistema original:

$$\hat{Q}\,\Delta\hat{y} = \hat{q} \tag{4.5}$$

Sistema transformado:

$$\tilde{Q}\,\tilde{\Delta}\hat{y} = \tilde{q} \tag{4.6}$$

Resíduo do sistema original:

$$r = \hat{q} - \hat{Q}\,\Delta\hat{y}\,,$$

Resíduo do sistema precondicionado:

$$\tilde{r} = \tilde{q} - \tilde{Q}\,\tilde{\Delta}\hat{y} = P^{-1}\hat{q} - P^{-1}\hat{Q}(P^{-1})'(P'\Delta\hat{y}) = P^{-1}(\hat{q} - \hat{Q}\,\Delta\hat{y}) = P^{-1}r\,.$$

O algoritmo seguinte fornece a solução do sistema linear (4.5) diretamente, ou seja, a solução aproximada do sistema original é fornecida sem calcular a solução aproximada do sistema precondicionado (4.6). Esta é a versão usual que aparece na literatura, com $M = PP^t$. Note que, se $P = I$, o sistema não é precondicionado. $(\Delta\widehat{y})^0$ é a aproximação inicial e $(\Delta\widehat{y})^l$ é a aproximação da solução encontrada na iteração l. Alguns detalhes da implementação, incluindo o critério de parada, são discutidos a seguir.

Algoritmo 4.3 (*gradiente conjugado precondicionado*)

Fornecer $\widehat{Q}$, $\widehat{q}$, M, $(\Delta\widehat{y})^0$;
$\ell \leftarrow 0$; $r^0 \leftarrow \widehat{q} - \widehat{Q}(\Delta\widehat{y})^0$;
Resolver $Mz^0 = r^0$;
$d^0 \leftarrow z^0$;

Enquanto o critério de parada não for satisfeito faça
$p^\ell \leftarrow \widehat{Q}\ d^\ell$;
$\alpha^\ell \leftarrow (r^t)^\ell\ z^\ell\ /\ (d^t)^\ell\ p^\ell$;
$(\Delta\widehat{y})^{\ell+1} \leftarrow (\Delta\widehat{y})^\ell + \alpha^\ell\ d^\ell$;
$r^{\ell+1} \leftarrow r^\ell - \alpha^\ell\ p^\ell$;
resolver $M\ z^{\ell+1} = r^{\ell+1}$;
$\beta^\ell \leftarrow (r^t)^{\ell+1}\ z^{\ell+1}\ /\ (r^t)^\ell\ z^\ell$;
$d^{\ell+1} \leftarrow z^{\ell+1} + \beta^\ell\ d^\ell$;
$\ell \leftarrow \ell + 1$;
$\Delta\widehat{y} \leftarrow (\Delta\widehat{y})^\ell$.

Observando o algoritmo, fica claro que o sistema linear $Mz = r$, com $M \in \mathbb{R}^{\widehat{m}\times\widehat{m}}$, z e $r \in \mathbb{R}^{\widehat{m}}$, que aparece em cada iteração do gradiente conjugado, deve ser fácil de ser resolvido. É comum também considerar $(\Delta\widehat{y})^0 = 0$. Neste caso, teremos $r^0 = \widehat{q}$ e eliminamos um produto matriz-vetor. Outra sugestão é colocar $(\Delta\widehat{y})^0 = \widehat{q}$, o que resulta numa economia de uma iteração neste método.

Na implementação do Algoritmo 4.3, seu critério de parada se dá quando a primeira das três condições seguintes ocorre:

(i) $(r^t)^\ell\ r^\ell \leq \varepsilon_{gc}$ (consideramos $\varepsilon_{gc} = 10^{-8}$) ;

(ii) $kmax = \widehat{m}$, onde $kmax$ é o número máximo de iterações permitido;

(iii) $|1 - \cos\sigma| < \varepsilon_{\cos}$, onde σ é o ângulo entre $\widehat{Q}(\Delta\widehat{y})^{\ell}$ e $\widehat{q}$;

$$\cos\sigma = \frac{|\widehat{q}\,'\ \widehat{Q}(\Delta\widehat{y})^{\ell}|}{||\widehat{q}||\ ||\widehat{Q}(\Delta\widehat{y})^{\ell}||}$$; geralmente tomamos $\varepsilon_{\cos} = 10^{-8}$.

O item (iii) é uma sugestão é de Resende e outros [58, 59, 61] para problemas de fluxo de custo mínimo com um único produto.

Podemos observar que o cálculo de $\cos\sigma$ é computacionalmente caro, tendo a complexidade de uma iteração do gradiente conjugado. Por este motivo, substituímos a expressão de $\cos\sigma$ por um cálculo aproximado: como $r = \widehat{q} - \widehat{Q}\ \Delta\widehat{y} \Rightarrow \widehat{q} - r = \widehat{Q}\ \Delta\widehat{y}$, consideramos a aproximação $\widehat{q} - r^{\ell} \approx \widehat{Q}\ \Delta\widehat{y}^{\ell}$, onde r^{ℓ} é o resíduo obtido na iteração ℓ dos gradientes conjugados. Em outras palavras, trabalhamos com a seguinte aproximação:

$$\cos\sigma \approx \frac{|\widehat{q}\,'\ (\widehat{q} - r^{\ell})|}{||\widehat{q}||\ ||(\widehat{q} - r^{\ell})||}$$

Esta última expressão pode ser calculada em cada iteração dos gradientes conjugados por ser mais barata do que a original. De fato, esta aproximação pode ser utilizada, pois, na prática, o método dos gradientes conjugados oferece direções boas (desde que o precondicionador usado seja efetivo). Outra sugestão de Portugal e Resende [58] para este critério de parada é iniciar com $\varepsilon_{cos} = 10^{-3}$ na primeira iteração do método primal-dual e, nas iterações seguintes, diminuir esta tolerância para $\varepsilon_{cos} \leftarrow 0.95\ \varepsilon_{cos}$. Outros detalhes do critério de parada estão melhor discutidos no Capítulo 6.

Cabe salientar que, quando aplicamos o Algoritmo 4.3 ao sistema linear (3.9), não precisamos armazenar explicitamente a matriz $\widehat{Q} = \widehat{A}\widehat{\Theta}\widehat{A}'$. Como já foi comentado anteriormente, exploramos as facilidades de trabalhar com uma matriz de incidência nó-arco. Assim, por exemplo, o produto matriz-vetor $\widehat{Q}v$, onde v é um vetor coluna de dimensão apropriada, será:

$$\begin{bmatrix} A\Theta^1A' & & & & A\Theta^1 \\ & A\Theta^2A' & & & A\Theta^2 \\ & & \ddots & & \vdots \\ & & & A\Theta^pA' & A\Theta^p \\ \Theta^1A' & \Theta^2A' & \cdots & \Theta^pA' & \sum_{k=1}^{p+1}\Theta^k \end{bmatrix} \begin{bmatrix} v^1 \\ v^2 \\ \vdots \\ v^p \\ v^{p+1} \end{bmatrix} = \begin{bmatrix} A\Theta^1A'\ v^1 + A\Theta^1\ v^{p+1} \\ A\Theta^2A'\ v^2 + A\Theta^2\ v^{p+1} \\ \vdots \\ A\Theta^pA'\ v^p + A\Theta^p\ v^{p+1} \\ \bar{v} \end{bmatrix}$$

onde $\bar{v} = \sum_{k=1}^{p} (\Theta^k A' \, v^k) + (\sum_{k=1}^{p+1} \Theta^k) \, v^{p+1}$.

Cada componente $g = A\Theta^k A' \, v^k + A\Theta^k \, v^{p+1}$ pode ser obtida por (o nó raiz é o nó m):

```
g ← 0 ;
v^k[m] ← 0 ;
for arco ← 1..n do
  begin
    i ← orig[arco] ;
    j ← dest[arco] ;
    g[i] ← g[i] + Θ^k[arco] * (v^k[i] − v^k[j] + v^{p+1}[arco]) ;
    g[j] ← g[j] + Θ^k[arco] * (v^k[j] − v^k[i] + v^{p+1}[arco]) ;
  endfor;
g[m] ← 0
```

Assim, cada vetor $g = A\Theta^k A' v^k + A\Theta^k v^{p+1}$ é calculado com $2n$ produtos, $4n$ somas e $2n$ subtrações. Em p blocos efetuamos então $2np$ produtos e $6np$ somas e subtrações. A última "linha" de $\hat{Q}v$, $\bar{v}$, é calculada com $(p+1)n$ produtos, pn subtrações e pn somas. Isto nos leva a um total de $8pn$ somas e subtrações e $(3p+1)n$ produtos, isto é, o produto $\hat{Q}v$ é efetuado em $\mathcal{O}(np)$ produtos e somas (subtrações).

Outro comentário é sobre o ponto inicial para o método dos gradientes conjugados: inicialmente, começamos colocando $(\Delta\hat{y})^0 = 0$. Em outra implementação, a direção inicial foi tomada como sendo a direção $\Delta\hat{y}$ obtida na iteração anterior do método primal-dual. A idéia era que, se $\hat{\Theta}$ e $\hat{q}$ não variam muito de uma iteração para a seguinte no método primal-dual, então a direção produzida pelo gradiente conjugado também não deve alterar muito. Os testes computacionais mostraram que quase não existe diferença entre estas duas versões, pois, nas primeiras iterações do método primal-dual as direções são bastante diferentes entre si e, nas últimas iterações, quando a solução tende a zero, elas apresentam ordem de grandeza distintas, não fazendo muita diferença então colocar ou não $(\Delta\hat{y})^0 = 0$.

4.3.3 Precondicionadores utilizados

Em métodos de pontos interiores para problemas gerais de programação linear, uma classe de precondicionadores bastante usada é a *fatoração incompleta* (ver, por exemplo, Golub e Van Loan [19] e Axelsson e Barker [6]). Nesta classe, encontramos a *fatoração LU incompleta* e a *fatoração de Cholesky incompleta*, caso a matriz do sistema $\hat{Q}$ seja simétrica e definida positiva. Como o padrão de esparsidade de L não necessariamente coincide com o padrão de esparsidade da parte inferior de $\hat{Q}$, em geral são feitas permutações em $\hat{Q}$ de modo a diminuir o "*fill-in*" de L (a heurística "*minimum degree*" é bastante utilizada). A idéia da fatoração incompleta é ignorar os *fill-in's*, deixando L com a mesma estrutura de esparsidade que a parte inferior de $\hat{Q}$. Axelsson e Barker [6] apresentam alguns resultados sobre a fatoração incompleta. Apresenta também outro tipo de precondicionador para o gradiente conjugado — o *SSOR (Symmetric Sucessive OverRelaxation)* e uma análise de sua convergência. Oliveira [56] apresenta uma classe mais geral de precondicionadores, trabalhando com o sistema aumentado (isto é, as variáveis envolvidas no sistema das direções são Δx e Δy), ao invés do sitema reduzido (onde as variáveis são apenas Δy). Ele mostra que os precondicionadores utilizados no sistema reduzido têm seu equivalente no sistema aumentado e, em vez de calcular a fatoração de Cholesky incompleta para o sistema reduzido, calcula a fatoração LU de um conjunto de colunas linearmente independentes de $\hat{Q}$. Suas experiências mostraram que este tipo de precondicionador apresenta a vantagem de ter uma maior eficiência nas últimas iterações do método primal-dual, ao contrário do que geralmente acontece, pois os precondicionadores costumam ser menos efetivos nestas últimas iterações.

Precondicionadores diagonais também são utilizados por todos estes autores, principalmente devido à sua simplicidade.

Para *problemas de fluxo de custo mínimo* com um único produto, os precondicionadores sugeridos são os de Resende e Veiga [60, 61, 62] — precondicionador *diagonal* e precondicionador *árvore (ou floresta) geradora máxima*, que descreveremos a seguir; o de Portugal e outros [57] (para problemas de transporte) — *fatoração QR incompleta*, que é baseado no precondicionador árvore geradora máxima (onde é feita a fatoração QR incompleta) e o de Mehrotra e Wang [53], baseado na fatoração de Cholesky incompleta e que combina as idéias de Resende e Veiga [62] e de Portugal [57] (a fatoração é

feita nos arcos correspondentes à árvore com a adição de uma matriz diagonal).

Precondicionador diagonal

Neste precondicionador, simplesmente tomamos

$$M = diag(\hat{Q}) = diag(\hat{A}\hat{\Theta}\hat{A}')$$

(ou $P = M^{1/2}$). Especificamente,

$$M = (\mathcal{D}^1, \cdots, \mathcal{D}^p, \ \mathcal{D}^{p+1})$$

onde

$$\mathcal{D}^k = diag(A\Theta^k A') \ , \quad k = 1, \cdots, p$$

e

$$\mathcal{D}^{p+1} = \sum_{k=1}^{p+1} \Theta^k$$

Lembramos que $\hat{Q}$ não é montada explicitamente. Para construir M , podemos utilizar parte do Algoritmo 4.1 para montar as p matrizes diagonais $\mathcal{D}^k$, onde efetuamos $2np$ somas para os p produtos. Na construção de $\mathcal{D}^{p+1} = \sum_{k=1}^{p+1} \Theta^k$ utilizamos np somas.

O sistema linear dos resíduos, $Mz = r$, que é resolvido em cada iteração do Algoritmo 4.3, necessita $\widehat{m}$ divisões sendo que, se um elemento de M for menor do que 10^{-8} , será designado o valor zero à componente correspondente em z (todos os cálculos foram feitos em precisão dupla).

Embora este precondicionador seja muito simples de ser calculado, ele só é efetivo nas primeiras iterações do método primal-dual. Por isso, seu uso costuma ser combinado com outros precondicionadores mais efetivos nas iterações finais.

Precondicionador floresta geradora máxima

Este é um precondicionador bastante efetivo para *problemas de fluxo de custo mínimo* com um único produto, com a vantagem adicional de convergir

à base ótima no final do método de pontos interiores, se o problema for não degenerado. Para problemas com um produto apenas, este precondicionador é uma *árvore geradora com raiz* da rede, onde os pesos nos arcos são dados pelos maiores elementos da matriz de "*scaling*".

Para apresentar este precondicionador, vamos nos concentrar inicialmente em um único produto, cuja matriz de coeficientes é dada por $A\Theta^k A'$.

Em nossa discussão sobre o comportamento da matriz de *scaling* Θ^k , vimos que uma indicação para que um arco seja básico é que ele apresente um valor grande para a componente correspondente Θ^k_j . Daí ser uma boa indicação tomar os arcos com os maiores pesos. Lembramos que, caso a rede seja desconectada, teremos uma *floresta geradora* em vez de uma *árvore geradora* com raiz.

A idéia deste precondicionador é então construir uma floresta geradora máxima. Na prática, podemos usar por exemplo, o algoritmo de *Kruskal* para construir a floresta geradora máxima correspondente a cada produto k , $k = 1, \cdots, p$, onde a estrutura de *heap* é explorada — o heap é feito em função dos arcos do produto k que possuem os maiores valores na matriz Θ^k correspondente. Como foi descrito no Capítulo 2 (Algoritmos 2.3 e 2.4), o algoritmo encontra os arcos de maior peso que formam a floresta geradora máxima do produto k, que fica bem caracterizada com a estrutura de dados definida anteriormente, ou seja, com os vetores *pred* (predecessor), *fio* (fio direto) e *rev* (fio reverso).

Seja B^k a floresta geradora máxima para o produto k, numa certa iteração do método primal-dual, cujos arcos selecionados são os que possuem maior peso na matriz Θ^k correspondente. Assim, B^k é uma submatriz de A cujas colunas linearmente independentes são dadas pelos arcos $e_{i_1}, e_{i_2}, \cdots, e_{i_{\bar{m}}}$, isto é,

$$B^k = \{e_{i_1}, e_{i_2}, \cdots, e_{i_{\bar{m}}}\}$$

onde $\bar{m}$ é o número de arcos de B^k . Se o grafo for conectado, $\bar{m} = m - 1$. O valor de $\bar{m}$ é o número de nós menos o número de componentes do grafo. Seja

$$\Theta^k_B = diag(\ \Theta^k_{e_{i1}},\ \Theta^k_{e_{i2}},\ \cdots,\ \Theta^k_{e_{i_{\bar{m}}}}\ )$$

O precondicionador chamado de *floresta geradora máxima* é então composto por uma floresta geradora máxima com raiz para cada produto k, $k = 1, \cdots, p$ e, para o acoplamento, ele ainda é formado pela diagonal correspondente de $\hat{Q}$, isto é,

$$P = \left[\begin{array}{ccc|c} B^1(\Theta_B^1)^{1/2} & & & \\ & B^2(\Theta_B^2)^{1/2} & & \\ & & \ddots & \\ & & B^p(\Theta_B^p)^{1/2} & \\ \hline & & & (\mathcal{D}^{p+1})^{1/2} \end{array}\right]$$

e

$$M = PP' = \left[\begin{array}{ccc|c} B^1\Theta_B^1(B^1)' & & & \\ & B^2\Theta_B^2(B^2)' & & \\ & & \ddots & \\ & & B^p\Theta_B^p(B^p)' & \\ \hline & & & \mathcal{D}^{p+1} \end{array}\right]$$

Assim, em cada iteração do gradiente conjugado precondicionado, o sistema linear dos resíduos, $Mz = r$, é formado por $p+1$ sistemas independentes (estamos omitindo aqui também o índice relativo ao número da iteração do gradiente conjugado):

$$B^k\Theta_B^k(B^k)'\, z^k = r^k\,, \qquad k = 1, \cdots, p \tag{4.7}$$

$$\mathcal{D}^{p+1} z^{p+1} = r^{p+1} \tag{4.8}$$

onde $z = (z^1\ z^2\ \cdots\ z^p\ z^{p+1})'$, $r = (r^1\ r^2\ \cdots\ r^p\ r^{p+1})'$ com z^k e $r^k \in R^{\bar{m}}$, $k = 1, \cdots, p$; z^{p+1} e $r^{p+1} \in \mathbb{R}^n$ ($\bar{m}$ é o número de arcos de B^k).

No Capítulo 2, vimos que a matriz associada à uma árvore geradora (base para o problema) pode ter suas linhas/colunas permutadas de modo a ficar na forma triangular. Isto é feito em $\mathcal{O}(m)$ operações.

Vamos supor que B^k é uma matriz triangular inferior. Cada sistema linear de (4.7) é resolvido em três partes:

(i) $B^k \bar{r}^k = r^k$

(ii) $\Theta_B^k \bar{z}^k = \bar{r}^k$

(iii) $(B^k)' z^k = \bar{z}^k$

Conforme o Capítulo 2, cada sistema linear triangular (i) é resolvido com operações de soma e produto apenas, percorrendo a árvore B^k no sentido das folhas para a raiz (Algoritmo 2.1). O mesmo se dá com cada sistema linear (iii) (triangular superior), mas agora percorremos a árvore no sentido da raiz para as folhas (Algoritmo 2.2). Finalmente, cada sistema linear (ii) é diagonal, e é resolvido efetuando-se $\bar{m}$ divisões.

O sistema (4.8) também é diagonal e é resolvido com n divisões.

Precondicionador combinado

Conforme sugerido na literatura para *problemas de fluxo de custo mínimo* com um produto, na prática se utiliza o precondicionador diagonal nas primeiras iterações do método de pontos interiores e, nas últimas, o precondicionador da floresta geradora. É necessário fazer um monitoramento das iterações do método de pontos interiores para se fazer a troca de precondicionadores, e isto não é muito fácil. Neste sentido, um número de iterações considerado bom para o método dos gradientes conjugados é $\omega\sqrt{\widehat{m}}$, onde $\omega > 0$ não é grande e $\widehat{m}$ é a dimensão do sistema linear. A quantidade $\omega\sqrt{\widehat{m}}$ costuma ser um indicador para a troca de precondicionadores: enquanto um precondicionador produz, no método dos gradientes conjugados, um número de iterações desta ordem, ele é usado. Caso contrário, o precondicionador é substituído por outro.

Numa primeira implementação do método dos gradientes conjugados, usamos apenas o *precondicionador diagonal.* Como já foi dito, este precondicionador não foi efetivo, mesmo para problemas considerados pequenos. Por exemplo, para $m = 200$ nós, $n = 300$ arcos e $p = 10$ produtos, o que resulta num sistema linear de dimensão $\widehat{m} = p(m-1)+n = 2290$ (redes conectadas), o algoritmo dos gradientes conjugados utilizou todas as iterações permitidas ($kmax = 2290$) e não alcançou a precisão desejada.

Numa segunda implementação, usamos uma combinação dos precondicionadores *diagonal* e *floresta geradora máxima*. De fato, podemos combinar estes dois precondicionadores de várias maneiras: podemos utilizar somente o diagonal, somente a floresta geradora, ou iniciar com o diagonal e passar para a floresta geradora. Foram realizados vários testes computacionais, e deles concluímos que, nas primeiras iterações do primal-dual, é indiferente utilizar um ou outro precondicionador. Como o diagonal é bem mais barato, é conveniente começar com ele. Por outro lado, nas últimas iterações do primal-dual, embora o precondicionador floresta geradora também perca sua eficiência, ele ainda é bem mais efetivo do que o diagonal: isto justifica o fato de que, uma vez abandonado o precondicionador diagonal, não devemos voltar a usá-lo. Por isso, também concluímos que a melhor estratégia é começar com o diagonal e terminar com a floresta geradora. Assim, trocamos de precondicionador quando o número de iterações do método dos gradientes conjugados é maior do que $\sqrt{\widehat{m}}$ e o número da iteração do método primal-dual é maior do que "$it.fg$", um parâmetro que devemos fornecer. Para vários testes efetuados, em problemas com dimensão até $m = 400$, $n = 800$, $p = 10$ $(\widehat{m} = 4790)$, o melhor valor para $it.fg$ (valor que diminuiu o número de iterações no gradiente conjugado, obtido experimentalmente) ficou em torno de 7 (ver Capítulo 6).

Portugal e outros [57], para problemas de transporte, comentam sobre a dificuldade de se concluir em qual iteração do método de pontos interiores se deve trocar de precondicionador. Eles supõem que o número desta melhor iteração é conhecido a priori, sendo um dado de entrada em sua implementação. Depois de vários experimentos, concluem que esta melhor iteração é a número 8. Aliás, esta é uma vantagem do precondicionador que apresentam (fatoração QR incompleta): ele tem um comportamento "parecido" com o diagonal nas primeiras iterações do método de pontos interiores e "parecido" com a floresta geradora nas últimas iterações, sem apresentar o problema da troca.

Castro [10] aplica gradientes conjugados apenas ao sistema linear (4.2). Ele mostra que a matriz S deste sistema é simétrica e definida positiva em cada iteração do método de pontos interiores e apresenta um precondicionador que é uma aproximação para S^{-1}.

4.3.4 Considerações sobre a floresta geradora máxima

Vamos iniciar esta discussão considerando o *problema de fluxo de custo mínimo* com um único produto, cuja matriz $\hat{Q}$ correspondente é $\hat{Q} = A\Theta A'$. Seja B uma base (floresta geradora com raiz) para este problema e vamos considerar a partição nas matrizes A e Θ dada por:

$$A = \begin{bmatrix} B & N \end{bmatrix} \; ; \qquad \Theta = \begin{bmatrix} \Theta_B & \\ & \Theta_N \end{bmatrix}$$

onde,
N: submatriz de A formada pelos arcos não básicos;

Θ_B: matriz diagonal cujos elementos correspondem aos arcos básicos;

Θ_N: matriz diagonal cujos elementos correspondem aos arcos não básicos.

A matriz transformada $\tilde{Q}$ é dada por $\tilde{Q} = P^{-1}\hat{Q}(P^{-1})'$, onde $P = B\Theta_B^{1/2}$. Assim,

$$\tilde{Q} = \Theta_B^{-1/2}B^{-1}\ \hat{Q}\ (B^{-1})'\Theta_B^{-1/2} = \Theta_B^{-1/2}B^{-1}\ A\Theta A'\ (B^{-1})'\Theta_B^{-1/2}\ .$$

Incluindo as partições de A e Θ , temos que

$$\tilde{Q} = \Theta_B^{-1/2}B^{-1}\begin{bmatrix} B & N \end{bmatrix}\begin{bmatrix} \Theta_B & \\ & \Theta_N \end{bmatrix}\begin{bmatrix} B' \\ N' \end{bmatrix}(B^{-1})'\Theta_B^{-1/2} =$$

$$= \Theta_B^{-1/2}B^{-1}\ (B\Theta_B B' + N\Theta_N N')\ (B^{-1})'\Theta_B^{-1/2} = I + \Theta_B^{-1}\ (B^{-1}N)\ \Theta_N\ (B^{-1}N)'\ .$$

Observamos então que, para problemas não degenerados, nas últimas iterações do método primal-dual de pontos interiores, como Θ_N se aproxima de zero, $\tilde{Q}$ se aproxima de I. Logo, $M = B\Theta_B B'$ se aproxima de $\hat{Q} = A\Theta A'$. Em outras palavras, B se aproxima da base ótima do problema, e esta é uma boa razão para se usar este precondicionador.

No problema de *multifluxo*, o precondicionador correspondente à floresta geradora máxima do problema de *fluxo de custo mínimo* com um único produto é uma *base* para o problema do multifluxo. Na prática, não é fácil obter esta base, principalmente para problemas degenerados — é difícil reconhecer se um determinado arco e_j de um produto k, que possui um valor grande de

Θ_j^k, está na floresta geradora deste produto ou está formando um ciclo com esta floresta. Tentamos algumas heurísticas neste sentido.

Vejamos o que acontece com a matriz transformada $\tilde{Q}$ neste caso, onde $\tilde{Q} = P^{-1}\hat{Q}(P^{-1})'$.

Definindo $Q^k = A\Theta^k A'$, $k = 1, \cdots, p$ e usando as definições anteriores de $\hat{Q}$, P e $\mathcal{D}^{p+1}$, temos que $\tilde{Q}$ é formada pelo produto das três matrizes seguintes:

$$\tilde{Q} = \left[\begin{array}{ccc|c} (B^1(\Theta_B^1)^{1/2})^{-1} & & & \\ & \ddots & & \\ & & (B^p(\Theta_B^p)^{1/2})^{-1} & \\ \hline & & & ((\mathcal{D}^{p+1})^{1/2})^{-1} \end{array}\right]$$

$$\left[\begin{array}{ccc|c} Q^1 & & & A\Theta^1 \\ & \ddots & & \vdots \\ & & Q^p & A\Theta^p \\ \hline \Theta^1 A' & \cdots & \Theta^p A' & \mathcal{D}^{p+1} \end{array}\right]$$

$$\left[\begin{array}{ccc|c} (B^1(\Theta_B^1)^{1/2})^{-1} & & & \\ & \ddots & & \\ & & (B^p(\Theta_B^p)^{1/2})^{-1} & \\ \hline & & & ((\mathcal{D}^{p+1})^{1/2})^{-1} \end{array}\right]'$$

Agora, similarmente ao *problema de fluxo de custo mínimo* com um único produto, vamos tomar cada matriz $Q^k = A\Theta^k A'$ e considerar a partição em A dada por:

$$A = [B^k | R^k | N^k]$$

onde,
B^k: base (floresta geradora com raiz) para o produto k ;

R^k: matriz composta por arcos que fazem ciclos com a floresta geradora do produto k (pode ser vazia; ver Capítulo 2);

N^k: matriz composta por arcos fora da floresta geradora do produto k e fora da base de multifluxo.

Estendendo estas partições para as matrizes Θ^k e $\mathcal{D}^{p+1}$, temos:

$$\Theta^k = \begin{bmatrix} \Theta_B^k & & \\ & \Theta_R^k & \\ & & \Theta_N^k \end{bmatrix} \quad ; \quad \mathcal{D}^{p+1} = \begin{bmatrix} \mathcal{D}_B^{p+1} & & \\ & \mathcal{D}_R^{p+1} & \\ & & \mathcal{D}_N^{p+1} \end{bmatrix}$$

Com estas partições, a matriz $\tilde{Q}$ pode ser calculada por partes: primeiramente, vamos calcular seus blocos; depois, suas faixas.

- Primeiro, observamos que $Q^k = A\Theta^k A'$ pode ser reescrita por:

$$Q^k = \begin{bmatrix} B^k & R^k & N^k \end{bmatrix} \begin{bmatrix} \Theta_B^k & & \\ & \Theta_R^k & \\ & & \Theta_N^k \end{bmatrix} \begin{bmatrix} (B^k)' \\ (R^k)' \\ (N^k)' \end{bmatrix} =$$

$$= B^k\Theta_B^k(B^k)' + R^k\Theta_R^k(R^k)' + N^k\Theta_N^k(N^k)'$$

Assim, cada bloco k de $\tilde{Q}$ é:

$$(\Theta_B^k)^{-1/2}(B^k)^{-1}\, Q^k\, [(B^k)^{-1}]'(\Theta_B^k)^{-1/2} =$$

$$= (\Theta_B^k)^{-1/2}(B^k)^{-1}\, [B^k\Theta_B^k(B^k)' + R^k\Theta_R^k(R^k)' + N^k\Theta_N^k(N^k)']\, ((B^k)^{-1})'(\Theta_B^k)^{-1/2} =$$

$$= I + (\Theta_B^k)^{-1}\, [(B^k)^{-1}\, R^k]\, \Theta_R^k\, [(B^k)^{-1}R^k]' + (\Theta_B^k)^{-1}\, [(B^k)^{-1}N^k]\, \Theta_N^k\, [(B^k)^{-1}N^k]'$$

Chamando de:

$$\bar{R}^k = (\Theta_B^k)^{-1}\, [(B^k)^{-1}R^k]\, \Theta_R^k\, [(B^k)^{-1}R^k]' \tag{4.9}$$

$$\bar{N}^k = (\Theta_B^k)^{-1}\, [(B^k)^{-1}N^k]\, \Theta_N^k\, [(B^k)^{-1}N^k]' \tag{4.10}$$

temos que cada bloco k de $\tilde{Q}$ é:

$$(\Theta_B^k)^{-1/2}(B^k)^{-1}\, Q^k\, [(B^k)^{-1}]'(\Theta_B^k)^{-1/2} = I + \bar{R}^k + \bar{N}^k \qquad (4.11)$$

• O último elemento de $\tilde{Q}$ ("linha" $p+1$ e "coluna" $p+1$) será a matriz identidade de ordem n.

• Por último, cada elemento da última "coluna" ("linha") de $\tilde{Q}$ é dado por (faixa vertical):

$$[B^k(\Theta_B^k)^{1/2}]^{-1} A\Theta^k \mathcal{D}^{-1/2}\ ,\quad k = 1, \cdots, p\ .$$

Colocando as partições em A , Θ^k e $\mathcal{D}^{p+1}$, temos:

$$(\Theta_B^k)^{-1/2}(B^k)^{-1} A\Theta^k \mathcal{D}^{-1/2} =$$

$$= (\Theta_B^k)^{-1/2}(B^k)^{-1} \left[\begin{array}{ccc} B^k & R^k & N^k \end{array}\right] \left[\begin{array}{ccc} \Theta_B^k & & \\ & \Theta_R^k & \\ & & \Theta_N^k \end{array}\right] \left[\begin{array}{ccc} \mathcal{D}_B^{-1/2} & & \\ & \mathcal{D}_R^{-1/2} & \\ & & \mathcal{D}_N^{-1/2} \end{array}\right] =$$

$$= (\Theta_B^k)^{-1/2}(B^k)^{-1} \left[\begin{array}{ccc} B^k\Theta_B^k\mathcal{D}_B^{-1/2} & R^k\Theta_R^k\mathcal{D}_R^{-1/2} & N^k\Theta_N^k\mathcal{D}_N^{-1/2} \end{array}\right]$$

Chamando de:

$$E_B^k = (\Theta_B^k)^{1/2}\, \mathcal{D}_B^{-1/2} \qquad (4.12)$$

$$E_R^k = (\Theta_B^k)^{-1/2}\, [(B^k)^{-1}R^k]\, \Theta_R^k\mathcal{D}_R^{-1/2} \qquad (4.13)$$

$$E_N^k = (\Theta_B^k)^{-1/2}\, [(B^k)^{-1}N^k]\, \Theta_N^k\mathcal{D}_N^{-1/2} \qquad (4.14)$$

temos que cada elemento da faixa vertical de $\tilde{Q}$ é:

$$(\Theta_B^k)^{-1/2}(B^k)^{-1} A\Theta^k \mathcal{D}^{-1/2} = \left[\begin{array}{ccc} E_B^k & E_R^k & E_N^k \end{array}\right] \qquad (4.15)$$

Finalmente, voltamos à expressão de $\tilde{Q}$ e substituímos cada bloco pela expressão dada por (4.11) e a última "coluna/linha" por (4.15), obtendo:

$$\tilde{Q} = \left[\begin{array}{ccc|c} I + \bar{R}^1 + \bar{N}^1 & & & [E_B^1 \mid E_R^1 \mid E_N^1] \\ & \ddots & & \vdots \\ & & I + \bar{R}^p + \bar{N}^p & [E_B^p \mid E_R^p \mid E_N^p] \\ \hline [E_B^1 \mid E_R^1 \mid E_N^1]' & \cdots & [E_B^p \mid E_R^p \mid E_N^p]' & I \end{array}\right]$$

Como foi discutido anteriormente, nas últimas iterações do método primal-dual de pontos interiores, Θ_N^k se aproxima de zero. Isto significa que as matrizes $\bar{N}^k$, dadas por (4.10) e E_N^k, dadas por (4.14) se aproximam da matriz nula. No entanto, os elementos de Θ_R^k são grandes, impedindo que os blocos de $\tilde{Q}$ se aproximem da identidade. Por outro lado, observamos também que o último "elemento" de $\tilde{Q}$ é a matriz identidade, ou seja, para este último bloco, o precondicionador diagonal é adequado.

Também salientamos que:
- as matrizes E_B^k dadas por (4.12) são matrizes diagonais;

- cada coluna $\hat{a}^{ij}$ de $(B^k)^{-1}R^k$ (ou de $(B^k)^{-1}N^k$) corresponde à uma cadeia do nó i ao nó j através da floresta geradora do produto k, e é fácil de ser obtida (ver Capítulo 2);

- em vista deste fato, as matrizes $\bar{R}^k$, $\bar{N}^k$, E_R^k e E_N^k , dadas respectivamente por (4.9), (4.10), (4.13) e (4.14) podem ser obtidas com facilidade, pois são formadas por produtos de matrizes diagonais e de cadeias em florestas geradoras.

Assim, com um esforço computacional maior, é possível obter um melhor precondicionador com o uso adicional de uma fatoração de Cholesky incompleta em $\tilde{Q}$ (desconsiderando os arcos não básicos). Observe que o número de elementos não nulos além da diagonal principal de $\tilde{Q}$ tende a diminuir com este procedimento, principalmente quando o número de restrições de acoplamento encontra-se restrito a alguns poucos arcos, como em muitos problemas práticos.

Capítulo 5

Heurística para obtenção de uma base ótima

Neste capítulo, apresentamos uma heurística para determinar uma solução básica ótima a partir de uma solução "quase-ótima" interior obtida ao final do algoritmo do método primal-dual de pontos interiores.

Para facilitar a apresentação, vamos supor que a rede associada ao problema é conectada. Isto quer dizer que a dimensão de qualquer base do problema é $\widehat{m} = (m-1)p + n$.

No Capítulo 2, caracterizamos uma base para o problema de multifluxo e vimos que ela é composta por três tipos de variáveis: arcos que formam uma árvore geradora em cada produto, arcos que formam ciclos com sua respectiva árvore geradora e as variáveis de folga das restrições de acoplamento. Este dois últimos conjuntos totalizam n variáveis. Em nossa heurística, vamos tentar identificar estas variáveis utilizando as matrizes de *scaling* Θ^k .

Já vimos que, à medida que a solução fornecida pelo método primal-dual de pontos interiores se aproxima de uma solução ótima, os elementos Θ_j^k se aproximam de zero ou de infinito.

Supondo que o problema é *não degenerado*, vemos que se $\Theta_j^k \to 0$, então, ou $x_j^k \to 0$ ou $x_j^k \to u_j^k$; isto é, o arco j do produto k deve ser não básico. Por outro lado, se $\Theta_j^k \to \infty$, então $0 < x_j^k < u_j^k$ e o arco j do produto k deve ser básico. Resta saber se este arco está na árvore geradora do produto k ou se está fazendo um ciclo com ela. Em outras palavras, seguindo a partição que usamos anteriormente, no primeiro caso teríamos um arco fazendo parte da matriz N^k. No último caso, o arco poderia estar tanto na matriz B^k quanto

na matriz R^k — e nossa maior dificuldade está exatamente em distingüir arcos de B^k de arcos de R^k.

Para o acoplamento, as conclusões são semelhantes: se $\Theta_j^{p+1} \to 0$, então $x_j^{p+1} \to 0$, isto é, a variável de folga da j-ésima restrição de acoplamento deve ser não básica. Por outro lado, se $\Theta_j^{p+1} \to \infty$, então $x_j^{p+1} > 0$, isto é, a variável de folga da j-ésima restrição de acoplamento deve ser básica.

Utilizaremos o vetor α, com n valores inteiros, para guardar a informação sobre qual produto cada um dos n arcos está representando, isto é,
$\alpha_j = k$ significa que o arco j está representando o produto k (forma um ciclo com a árvore geradora do produto k) e
$\alpha_j = p + 1$ significa que o arco j está representando uma variável de folga das restrições de acoplamento.

Iniciamos nossa heurística fazendo

$$\alpha_j = p+1 \ , \quad \text{se} \quad \Theta_j^{p+1} > 1 \ ;$$

$$\alpha_j = 0 \ , \quad \text{caso contrário.}$$

Vamos supor que n_a componentes do vetor α são nulas. Note que, se $n_a = 0$, temos um problema fácil de resolver, pois ele é totalmente desacoplado: todas as n variáveis de folga das restrições de acoplamento estão na base. Os outros $(m-1)p$ arcos básicos estão formando árvores geradoras em seus produtos. Vamos então supor que $n_a > 0$.

Os arcos j tais que $\alpha_j = 0$ são identificados no vetor β de n_a componentes inteiras, isto é,

$$\beta_t = j \quad \text{se } \alpha_j = 0 \ , \quad t = 1, \cdots, n_a \ .$$

Utilizaremos também a matriz Ψ, de dimensão $p \times n_a$, formada pelos valores inteiros 0, 1 ou 2, para assinalar o estado de um arco j que está no vetor β (ou seja, ainda não sabemos qual produto este arco está representando).

Inicialmente, fazemos:

$$\Psi_{k,t} = 1 \ , \quad \text{se} \quad \Theta_j^k > 1$$

$$\Psi_{k,t} = 0 \ , \quad \text{caso contrário}$$

para $k = 1, \cdots, p$, $t = 1, \cdots, n_a$ e $j = \beta_t$.

Um elemento que vale 1 em Ψ significa que o arco correspondente pode fazer parte da árvore geradora do produto k; se o elemento for nulo, este arco não pode fazer parte desta árvore.

Por exemplo, supondo um problema com 3 produtos, numa rede com 8 nós e 15 arcos, poderíamos ter, numa primeira alocação para o vetor α:

$$\alpha = (4,\ 0,\ 4,\ 4,\ 0,\ 4,\ 4,\ 4,\ 4,\ 4,\ 0,\ 4,\ 4,\ 0,\ 0)$$

Nesta caso, teremos $n_a = 5$ e $\beta = (2,\ 5,\ 11,\ 14,\ 15)$.

A primeira alocação para a matriz Ψ poderia ser (dependendo das matrizes Θ^k):

$\Psi =$

produtos	e_2	e_5	e_{11}	e_{14}	e_{15}
1	1	1	0	0	0
2	1	1	0	1	1
3	1	1	1	0	0

Na próxima etapa, vamos tentar construir uma árvore geradora para cada produto k. Os arcos que podem fazer parte desta árvore são aqueles que possuem Θ_j^k e Θ_j^{p+1} grandes. Nesta primeira tentativa, ou conseguimos formar uma árvore geradora para cada produto ou conseguimos uma floresta para o produto (caso os arcos que podem fazer parte da árvore não formem um subgrafo conectado com $m-1$ nós). Neste último caso, vamos atualizar a matriz Ψ: vamos buscar no vetor β os arcos que possam completar a árvore geradora deste produto. Estes arcos são os que não formarão ciclos na floresta deste produto. Isto é feito da seguinte maneira:
trocamos $\Psi_{k,t} = 1$ para $\Psi_{k,t} = 2$ se o produto k ainda não possui sua árvore geradora e o t-ésimo arco do vetor β (arco j) não forma um ciclo na floresta deste produto. Em outras palavras, $\Psi_{k,t} = 2$ significa que o arco j tanto pode fazer parte da matriz B^k quanto da matriz R^k.

Continuando a heurística, pesquisamos a matriz Ψ com o objetivo de atualizar os elementos nulos do vetor α pelo número do produto ao qual o arco será alocado.

Se $\Psi_{k,t} = 1$, o arco j correspondente ($j = \beta_t$) será alocado ao produto k, isto é, $\alpha_j = k$. Além disso, vamos zerar $\Psi_{k,t}$ para indicar que este arco já foi eliminado (alocado). De fato, esta coluna de Ψ pode ser eliminada. Percorremos toda a coluna t de Ψ e procuramos por elementos com valor 2: para cada linha onde ocorrer $\Psi_{k,t} = 2$, o arco j será alocado na floresta do produto k correspondente (ver observação no final do Algoritmo 5.1).

Em seguida, pesquisamos os elementos de Ψ com valor 2 — sempre em colunas que correspondem aos arcos que ainda não foram alocados.

Vamos pesquisar as *colunas*. Se existe um único elemento igual a 2 na coluna t e este elemento ocorre na linha k de Ψ, então fazemos $\Psi_{k,t} = 1$ e o arco j correspondente será alocado ao produto k, isto é, $\alpha_j = k$. Novamente, esta coluna de Ψ é eliminada.

Agora, vamos pesquisar as *linhas*. Se existe um único elemento igual a 2 na linha k de Ψ, e este elemento está na coluna t, então o arco j correspondente é colocado na floresta do produto k (este arco é necessário para completar a árvore geradora deste produto) e fazemos $\Psi_{k,t} = 0$.

Note que, em todas as situações onde um arco é colocado na floresta de um produto, é verificado antes se ele não forma um ciclo nesta floresta.

Este processo é repetido enquanto pudermos encontrar os casos que descrevemos anteriormente. Como a matriz Ψ tem seus valores sistematicamente alterados, ao final deste processo esta matriz apresenta a seguinte propriedade: ou ela se transformou na matriz nula, ou ela apresenta pelo menos dois coeficientes 2 em cada linha e em cada coluna. Assim, temos duas situações possíveis:

(i) a matriz Ψ foi completamente zerada. Isto significa que conseguimos encontrar as árvores geradoras de todos os produtos e todos os n arcos foram alocados (ou para um produto ou para o acoplamento), isto é, o vetor α foi completamente determinado. Podemos prosseguir com a resolução dos sistemas lineares básicos;

(ii) restam na matriz Ψ vários elementos iguais a 2. Isto significa que nem todos os produtos possuem ainda suas árvores geradoras, e o vetor α ainda possui elementos nulos. Neste caso, podemos escolher qualquer produto que não possui árvore geradora, digamos, o produto k e escolhemos ainda um arco j tal que $\alpha_j = 0$ para completar a árvore deste produto ($\Psi_{k,t} = 2$, para $j = \beta_t$) . Repetimos o processo, até que a situação (i) seja alcançada.

As árvores geradoras estarão armazenadas conforme a estrutura descrita no Capítulo 2, isto é, nos vetores *pred*, *fio* e *rev*. Os arcos destas árvores formam as matrizes B^k. Os vetores α e β indicam os arcos que formam as matr izes R^k.

Em resumo, esta heurística pode ser colocada no Algoritmo 5.1 seguinte.

Neste algoritmo, utilizamos os seguinte vetores e matrizes:
$\alpha[1..n]$: vetor inteiro (já descrito);
$\beta[1..n_a]$: vetor inteiro (já descrito);
$peso[1..n]$: vetor real; contém os pesos dos arcos que vão compor a floresta geradora de um produto;
$\Psi[1..n_a, 1..n_a]$: matriz inteira (já descrita);
$\Theta[1..(p+1), 1..n]$: matriz real; contém as matrizes de *scaling* de cada um dos produtos, mais a do acoplamento.

Algoritmo 5.1 (*obtenção de uma base ótima*)

• (*Inicialização*)
$n_a \leftarrow 0$;
para $k \leftarrow 1, \cdots, p$ e $j \leftarrow 1, \cdots, n$, fazer:
  se $\Theta[k,j] > 1$ então $\alpha[j] \leftarrow p+1$
    senão
      $\alpha[j] \leftarrow 0$;
      $n_a \leftarrow n_a + 1$;
      $\beta[n_a] \leftarrow j$;
para $k \leftarrow 1, \cdots, p$ e $t \leftarrow 1, \cdots, n_a$, fazer:
  se $\Theta[k, \beta[t]] > 1$ então $\Psi[k,t] \leftarrow 1$
    senão $\Psi[k,t] \leftarrow 0$;

• (*primeira tentativa de árvore geradora*)
para $k \leftarrow 1, \cdots, p$, fazer:
  para $j \leftarrow 1, \cdots, n$, fazer:
    se $\Theta[k,j] > 1$ e $\Theta[p+1,j] > 1$
      então $peso[j] \leftarrow \Theta[k,j]$;
      senão $peso[j] \leftarrow -1$;
    montar a floresta do produto k ;
    se for desconectada, então
      para $t \leftarrow 1, \cdots, n_a$, fazer:
        se $\Psi[k,t] = 1$ e o arco $j = \beta[t]$ pode ser conectado à floresta do produto k sem formar ciclo, então $\Psi[k,t] \leftarrow 2$;

• (*correção da matriz* Ψ)
Enquanto for possível encontrar na matriz Ψ um elemento igual a 1 ou um único elemento igual a 2 em uma linha ou coluna, repetir os procedimentos seguintes:

-(*procurar* 1)
para $k \leftarrow 1, \cdots, p$ e para $t \leftarrow 1, \cdots, n_a$, fazer:
se $\Psi[k,t] = 1$ e $\alpha[\beta[t]] = 0$
então
$\alpha[\beta[t]] \leftarrow k$;
$\Psi[k,t] \leftarrow 0$;
para $\bar{k} = 1, \cdots, p$ fazer:
se $\Psi[\bar{k}, \beta[t]] = 2$ então colocar o arco $\beta[t]$ na floresta do produto $\bar{k}$ e eliminar a coluna t de Ψ ;

-(*procurar único* 2 *na coluna*)
para $t \leftarrow 1, \cdots, n_a$, fazer:
se existe um único 2 na coluna t (e ocorre na linha k)
então
$\Psi[k,t] \leftarrow 1$;
$\alpha[\beta[t]] \leftarrow k$;
eliminar a coluna t de Ψ ;

-(*procurar único* 2 *na linha*)
para $k \leftarrow 1, \cdots, p$, fazer:
se existe um único 2 na linha k (e ocorre na coluna t)
então colocar o arco $j = \beta[t]$ na floresta do produto k ;

• (*teste de parada*)
Se (o número de árvores geradoras for p) e ($\alpha[j] \neq 0$, $\forall j = 1, \cdots, n$)
então fim do procedimento
senão
selecionar $j = \beta[t]$ tal que $\alpha[j] = 0$;
selecionar k tal que este produto não possua árvore geradora;
colocar o arco j na floresta do produto k
voltar à correção da matriz Ψ .

obs. 1: o procedimento "colocar o arco j na floresta do produto k " inclui atualizações na matriz Ψ tais como:
(i) $\Psi[k,t] = 0$;
(ii) se a árvore geradora deste produto foi completada com este arco, então, percorrer a linha k de Ψ e procurar elementos iguais a 2: se $\Psi[k,.] = 2$, fazer $\Psi[k,.] = 1$;
(iii) caso a árvore geradora não tenha sido completada, percorrer ainda a linha k de Ψ e procurar elementos iguais a 2. Verificar, então, se o arco correspondente não forma ciclo com esta árvore geradora. Neste caso, estes elementos de Ψ são trocados por 1.

obs. 2: no "teste de parada" do Algoritmo 5.1, a seleção do arco j com $\alpha[j] = 0$ é feita simplismente tomando o primeiro arco j que possui $\alpha[j] = 0$. Da mesma forma, a seleção do produto k que ainda não possui árvore geradora é feita tomando o primeiro produto que ainda não possui árvore geradora. Outros critérios serão testados futuramente, talvez ainda em função dos elementos da própria matriz Θ^k .

Notamos também que, dado o alto número de buscas em linhas e colunas que é necessário fazer na matriz Ψ, é desejável que sua dimensão (n_a) seja pequena. Em outras palavras, para que esta heurística seja eficiente, é desejável que $0 < n_a << n$, isto é, apenas algumas restrições de acoplamento estão, de fato, amarrando o problema. Se $n_a = n$, o problema é totalmente acoplado, já que todas as variáveis de folga do acoplamento são nulas. Este é o caso mais difícil de resolver, já que teremos que alocar todos os n arcos aos p produtos.

Finalmente, os sistemas lineares que encontram as variáveis básicas — primais e duais — são resolvidos conforme os esquemas descritos no Capítulo 2.

Lembramos que, para obter a solução primal, resolvemos:
• p sistemas lineares triangulares, cada um com a matriz B^k de um produto;
• um sistema linear quadrado com a matriz $\bar{F}$, dado por (2.3), cuja dimensão é n_a: a soma do número de arcos em ciclo em todos os produtos;
• novamente, p sistemas lineares triangulares com as matrizes B^k .

Similarmente, para obter a solução dual, resolvemos:

- p sistemas lineares triangulares, cada um com a matriz $(B^k)'$ de um produto;
- um sistema linear com a matriz $\bar{F}'$;
- novamente, p sistemas lineares triangulares com as matrizes $(B^k)'$.

Exemplo 1

Considere um problema com 2 produtos, 4 nós e 6 arcos, cuja rede é dada pela Figura 5.1. Os arcos são dados por $\{e_1, \cdots, e_6\}$.

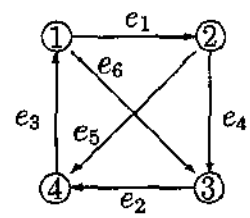


Figura 5.1: *rede exemplo para a heurística*

Vamos supor que, ao final do método primal-dual de pontos interiores, temos, para os produtos 1 e 2 e para o acoplamento, respectivamente:

$\Theta^1 = diag\ (\infty,\ \infty,\ \infty,\ \infty,\ 0,\ 0)$
$\Theta^2 = diag\ (\infty,\ \infty,\ 0,\ 0,\ \infty,\ \infty)$
$\Theta^3 = diag\ (\infty,\ 0,\ \infty,\ \infty,\ 0,\ \infty)$

Aplicando o Algoritmo 5.1, teremos:

vetor α inicial: $\alpha = (3,\ 0,\ 3,\ 3,\ 0,\ 3)$

vetor β: $\beta = (2,\ 5)$

matriz Ψ inicial: $\Psi = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$

Primeira tentativa de árvore geradora:

produto 1:

vetor *peso*: $peso = (\infty,\ -1,\ \infty,\ \infty,\ -1,\ -1)$

arcos candidatos à floresta do produto 1: $e_1,\ e_3,\ e_4$

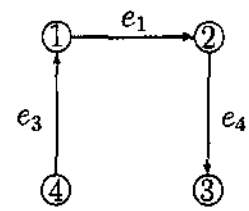


Figura 5.2: *floresta do produto 1*

produto 2:

vetor *peso*: $peso = (\infty,\ -1,\ -1,\ -1,\ -1,\ \infty)$

arcos candidatos à floresta do produto 2: $e_1,\ e_6$

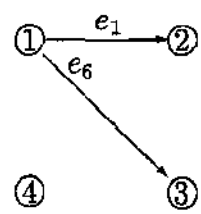


Figura 5.3: *floresta do produto 2*

Temos então uma árvore geradora apenas para o produto 1; o produto 2 apresenta uma floresta desconectada. As correções na matriz Ψ e no vetor α nos levam sucessivamente à:

$$\Psi : \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} . & 0 \\ . & 1 \end{bmatrix} \rightarrow \begin{bmatrix} . & . \\ . & . \end{bmatrix}$$

$$\alpha : (3,\ 0,\ 3,\ 3,\ 0,\ 3\ ) \rightarrow (3,\ 1,\ 3,\ 3,\ 0,\ 3\ ) \rightarrow (3,\ 1,\ 3,\ 3,\ 2,\ 3\ )$$

Isto significa que o arco e_2 vai completar a árvore geradora do produto 2, como pode ser visto na Figura 5.4; este arco representa o produto 1 e o arco e_5 representa o produto 2 .

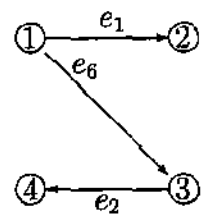


Figura 5.4: *árvore geradora do produto 2*

Assim, conseguimos as árvores geradoras para os dois produtos, o vetor α ficou completo e a matriz Ψ tornou-se vazia. Podemos então passar à resolução dos sistemas lineares básicos para a obtenção das variáveis primais e duais, uma vez que:

$$B^1 = (e_1,\ e_3,\ e_4)\ ;\quad R^1 = (e_2)\ ;\quad N^1 = (e_5,\ e_6)\ .$$

$$B^2 = (e_1,\ e_6,\ e_2)\ ;\quad R^2 = (e_5)\ ;\quad N^2 = (e_3,\ e_4)\ .$$

Exemplo 2

Vamos tomar ainda um problema com dois produtos e com a mesma rede do exemplo 1 (Figura 5.1). Agora, vamos supor que, ao final do método primal-dual de pontos interiores, temos, para os produtos 1 e 2 e para o acoplamento, respectivamente:

$\Theta^1 = diag\ (\infty,\ \infty,\ \infty,\ \infty,\ 0,\ 0)$
$\Theta^2 = diag\ (\infty,\ \infty,\ 0,\ 0,\ \infty,\ \infty)$
$\Theta^3 = diag\ (\ 0,\ 0,\ \infty,\ \infty,\ \infty,\ \infty)$

Aplicando o Algoritmo 5.1, teremos:

vetor α inicial: $\alpha = (0,\ 0,\ 3,\ 3,\ 3,\ 3)$

vetor β: $\beta = (1,\ 2)$

matriz Ψ inicial: $\Psi = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Primeira tentativa de árvore geradora:

produto 1:

vetor *peso*: $peso = (-1,\ -1,\ \infty,\ \infty,\ -1,\ -1)$

arcos candidatos à floresta do produto 1: $e_3,\ e_4$

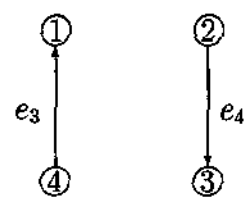


Figura 5.5: *floresta do produto 1*

produto 2:

vetor *peso*: $peso = (-1,\ -1,\ -1,\ -1,\ \infty,\ \infty)$

arcos candidatos à floresta do produto 2: $e_5,\ e_6$

Neste caso, não temos árvore geradora para nenhum dos produtos. As correções na matriz Ψ nos levam a:

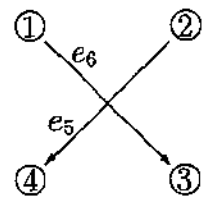


Figura 5.6: *floresta do produto 2*

$$\Psi : \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

Não temos a situação de um único elemento de valor 2 em uma linha ou coluna da matriz Ψ ; os elementos de valor 2 na matriz Ψ formam um ciclo.

Selecionamos então o produto 1 e o arco e_1 para ser colocado em sua floresta geradora, completando a árvore geradora deste produto, conforme a Figura 5.7 .

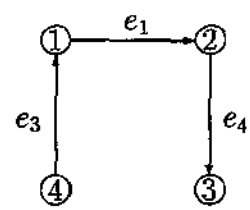


Figura 5.7: *árvore geradora do produto 1*

Isto nos leva sucessivamente à:

$$\alpha : (0,\ 0,\ 3,\ 3,\ 3,\ 3\ ) \rightarrow (2,\ 0,\ 3,\ 3,\ 3,\ 3\ ) \rightarrow (2,\ 1,\ 3,\ 3,\ 3,\ 3\ )$$

$$\Psi : \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 \\ 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & . \\ 1 & . \end{bmatrix} \rightarrow \begin{bmatrix} . & . \\ . & . \end{bmatrix}$$

E, para o produto 2, sua árvore geradora é completada com o arco e_2, conforme a Figura 5.8 .

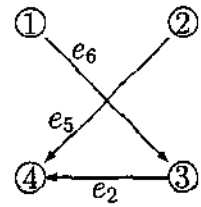


Figura 5.8: *árvore geradora do produto 2*

Novamente, conseguimos as árvores geradoras para os dois produtos, o vetor α ficou completo e agora a matriz Ψ tornou-se vazia. Podemos então passar à resolução dos sistemas lineares básicos para a obtenção das variáveis primais e duais, considerando:

$$B^1 = (e_3,\ e_4,\ e_1)\ ;\quad R^1 = (e_2)\ ;\quad N^1 = (e_5,\ e_6)\ .$$

$$B^2 = (e_5,\ e_6,\ e_2)\ ;\quad R^2 = (e_1)\ ;\quad N^2 = (e_3,\ e_4)\ .$$

Convém observar que, inicialmente, elaboramos uma heurística mais simples para obter uma base ótima. Nesta heurística mais simples, anterior à dada pelo Algoritmo 5.1, não utilizamos a matriz Ψ. O vetor α era inicializado como no Algoritmo 5.1 e os pesos para as árvores geradoras dos produtos também, mas permitíamos que arcos com peso -1 fizessem parte das árvores geradoras dos produtos, isto é, estas árvores geradoras eram formadas de uma única vez. Depois de construídas as ávores geradoras, o preenchimento das posições nulas do vetor α era feito simplesmente assinalando o produto de maior peso, ou seja, se para o arco j, $\alpha[j] = 0$, então $\alpha[j]$ era alterado para $\alpha[j] = \bar{k}$, onde $\bar{k} = \max_{k=1,\cdots,p}\{\Theta_j^k\}$. Construída a partição B^k, R^k, N^k, $\forall k$, a montagem das matrizes e a resolução dos sistemas lineares é idêntica nos dois casos.

Infelizmente, esta heurística mais simples falhou em muitos casos; daí elaborarmos uma mais complexa, dada pelo Algoritmo 5.1.

Finalizando, podemos colocar pelo menos duas utilidades para esta heurística:

Em primeiro lugar, ela pode antecipar a parada do método primal-dual: o número de iterações e, consequentemente, o tempo de cpu, podem ser reduzidos se esta heurística for utilizada. Com alguns critérios sobre a factibilidade da solução corrente apresentada pelo método primal-dual e sobre a otimalidade desta solução, podemos fazer uma tentativa de encontrar uma solução básica ótima através desta heurística, antecipando o término do método primal-dual.

Outra possibilidade seria utilizar esta heurística como precondicionador no método do gradiente conjugado. Neste caso, seria utilizada a base encontrada por ela (e não a solução básica associada). Este precondicionador poderia ser tentado nas últimas iterações do método primal-dual, uma vez que ele é computacionalmente caro e necessita de mais informações sobre a solução corrente.

Capítulo 6

Resultados Computacionais

6.1 Introdução

Neste capítulo, vamos apresentar os experimentos computacionais obtidos com diversas aplicações do método primal-dual de pontos interiores especializado para o problema de multifluxo. Estes experimentos comparam duas expressões para o parâmetro de centragem μ, os dois tipos de precondicionadores apresentados no Capítulo 4, os três pontos iniciais apresentados no Capítulo 3, alguns critérios de parada para o método do gradiente conjugado, o aproveitamento ou não da direção anterior no método do gradiente conjugado, mostrando ainda algumas informações que podem ser obtidas com as matrizes de *scaling* e alguns testes com a heurística apresentada no Capítulo 5. Resolvemos ainda alguns problemas da Netlib e comparamos nosso programa com o programa Lipsol.

Em todos os experimentos, consideramos o seguinte par primal-dual correspondente ao problema de multifluxo (já na forma padrão):

$$\begin{aligned} \min \quad & \sum_{k=1}^{p} (c^k)' x^k \\ s.a. \quad & \begin{cases} Ax^k = b^k, \quad k = 1, \ldots, p \\ \sum_{k=1}^{p} x^k + x^{p+1} = d \\ x^k + s^k = u^k, \quad k = 1, \ldots, p \\ x^k, \; s^k \geq 0, \; k = 1, \ldots, p, \; x^{p+1} \geq 0 \end{cases} \end{aligned} \tag{6.1}$$

$$\begin{aligned} \max \quad & \{ \sum_{k=1}^{p} [\ (b^k)'y^k - (u^k)'w^k\ ] + d'y^{p+1}\ \} \\ s.a. \quad & \begin{cases} A'y^k + y^{p+1} - w^k + z^k = c^k, \quad k = 1, \ldots, p \\ w^k,\ z^k \geq 0,\ k = 1, \ldots, p,\ y^{p+1} \leq 0 \end{cases} \end{aligned} \tag{6.2}$$

E na aplicação do algoritmo primal-dual de pontos interiores, o par primal-dual se expressa como:

$$\begin{aligned} \min \quad & \widehat{c}'\widehat{x} \\ s.a. \quad & \begin{cases} \widehat{A}\widehat{x} = \widehat{b} \\ \widehat{x} + \widehat{s} = \widehat{u} \\ \widehat{x},\ \widehat{s} \geq 0 \end{cases} \end{aligned} \tag{6.3}$$

$$\begin{aligned} \max \quad & \{\ \widehat{b}'\widehat{y} - \widehat{u}'\widehat{w}\ \} \\ s.a. \quad & \begin{cases} \widehat{A}'\widehat{y} - \widehat{w} + \widehat{z} = \widehat{c} \\ \widehat{w},\ \widehat{z} \geq 0 \end{cases} \end{aligned} \tag{6.4}$$

6.2 Gerador de problemas

Para realizar os experimentos computacionais, desenvolvemos um gerador de problemas de multifluxo, onde os problemas são gerados a partir da especificação de um *tamanho* e de uma *semente*. A rede associada ao problema é conectada e os vetores u^k e c^k, $k = 1, \cdots, p$, são gerados com distribuição uniforme em intervalos cujos limites são fornecidos. Além disso, podemos gerar problemas primal-dual factíveis com opção de deixar as restrições de acoplamento mais "apertadas" ou mais "folgadas".

Os parâmetros de entrada para este gerador são os valores m, n, p, *semente*, *cmin*, *cmax*, *umin*, *umax*, $inft$, *pcan* e fa, que passamos a descrever:

• m , n , p: número de nós da rede, número de arcos da rede e número de produtos, respectivamente. Estes três dados são parâmetros inteiros e

definem o tamanho do problema. A matriz de restrições $\widehat{A}$ possui $pm + n$ linhas e $(p+1)n$ colunas.

• *semente:* parâmetro inteiro que é utilizado no gerador de números aleatórios. Se queremos gerar (ou recuperar) um mesmo problema, basta mantermos o mesmo valor para *semente* e demais parâmetros.

• $[cmin, cmax]$: intervalo de números reais para geração dos custos, com distribuição uniforme.

• $[umin, umax]$: intervalo de números reais para geração das canalizações dos arcos, com distribuição uniforme. Supomos que os limitantes inferiores de todas as variáveis de fluxo são nulos.

• $inft$: parâmetro real que representa "infinito". É usada, por exemplo, no teste da razão e como limitante superior de variáveis não canalizadas.

• *pcan:* parâmetro real que representa a probabilidade de cada variável ser canalizada ($pcan \in [0\ ;\ 1]$). Se $pcan = 1$, todas as variáveis do problema serão canalizadas; se $pcan = 0$, todas as variáveis do problema terão como limitante superior $inft$.

• fa: parâmetro real que representa uma "folga" que é dada nas restrições de acoplamento, na geração do vetor d. Com este parâmetro, é formado o vetor de n componentes $f = (fa, \cdots, fa)$. Quanto maior for o valor de fa, mais desacoplado será o problema; quanto menor for seu valor, mais difícil poderá ser o problema.

Com os parâmetros m e n , é gerada uma rede cuja matriz de incidência A é armazenada nos vetores *orig* (origem de cada arco) e *dest* (destino de cada arco). Caso a rede não seja conectada, são criados novos arcos de modo a conectá-la. Por isso, o número final de arcos (n) pode ser maior do que o que foi especificado.

Para se ter alguma garantia de que o problema terá solução, é gerado um vetor x^k cujas componentes têm distribuição uniforme em $[0,\ u^k]$, $k = 1, \cdots, p$. A partir de x^k , calculamos $b^k = Ax^k$ de modo que, para cada bloco k, a igualdade $\sum_{i=1}^{m} b_i^k = 0$ seja satisfeita.

O lado direito das restrições de acoplamento é calculado fazendo $d = \sum_{k=1}^{p} x^k + f$, onde f é o vetor formado pelas n componentes fa, descrito anteriormente.

Observamos ainda, nos vários experimentos realizados, que se geramos dados *inteiros*, há uma probabilidade maior de gerar problemas *degenerados*. Isto é interessante no estudo deste tipo de problemas. Por outro lado, se geramos dados *reais* (números reais), geralmente obtemos problemas *não degenerados*. Por isso, temos duas verões do gerador: uma que trabalha com números reais e outra que trabalha com números inteiros.

6.3 O programa

A implementação do Algoritmo 3.1 está feita no programa **mult**, que foi codificadoo em Pascal, versão 2.0, com dupla precisão. Todos os experimentos foram realizados numa Sun Super SPARC II, com 70 MHz e 32 Mbytes de memória RAM. O programa **mult** resolve problemas gerais de multifluxo, e não apenas uma classe específica deles.

6.3.1 Parâmetros e tolerâncias

Alguns dos **parâmetros** utilizados em **mult** já foram discutidos no Capítulo 3. Em particular, para o cálculo de μ, tomamos (aqui, ℓ é o número da iteração do método primal-dual):

$$\mu^\ell = \eta^\ell \, \frac{\gamma^\ell}{\widehat{n}} = 0.1 \, \frac{(\widehat{x}^\ell)' \, \widehat{z}^\ell + (\widehat{w}^\ell)' \, \widehat{s}^\ell}{(p+1)n}$$

onde
$\eta^\ell = \eta = 0.1$ e γ^ℓ é a folga complementar.

Fizemos também vários testes considerando, no cálculo de μ, as variáveis de folga das canalizações, como será mostrado mais adiante. Neste caso, a expressão para μ tem seu denominador alterado para $(2p+1)n$.

No teste da razão, também tomamos δ^ℓ como um número fixo:

$$\delta^\ell = \delta = 0.9995 \, .$$

Tanto η quanto δ foram escolhidos a partir de experimentos computacionais.

No caso de δ, os resultados praticamente não se alteraram quando foi colocado 0.9995 ou 0.99995 , sugestões mais comumente encontradas na literatura. Assim, optamos por 0.9995 .

O programa **mult** utiliza ainda os parâmetros:

$it.max = 50$: limitante para o número de iterações do método primal-dual.

$mxfun = -10^{20}$: limitante para o valor da função objetivo.

$it.dg$: parâmetro que indica o número da iteração do método primal-dual a partir da qual é utilizado o precondicionador *diagonal* no método dos gradientes conjugados. Se $1 \leq it.dg \leq it.max$, o precondicionador *diagonal* é utilizado a partir da iteração $it.dg$ do método primal-dual. Se $it.dg > it.max$, este precondicionador nunca é utilizado.

$it.fg$: parâmetro que indica o número da iteração do método primal-dual a partir da qual é utilizado o precondicionador *floresta geradora máxima* no método dos gradientes conjugados. Se $1 \leq it.fg \leq it.max$, o precondicionador *floresta geradora máxima* é utilizado a partir da iteração $it.fg$ do método primal-dual. Se $it.fg > it.max$, este precondicionador nunca é utilizado.

Dependendo dos valores de $it.dg$ e de $it.fg$, podemos fazer algumas combinações quanto ao uso dos precondicionadores. Contudo, devido aos vários experimentos computacionais, decidimos por começar com o *diagonal* e terminar com a *floresta geradora máxima* uma vez que, nas primeiras iterações do método primal-dual, por um lado, não possuímos boas informações para formar as florestas e, por outro lado, o precondicionador diagonal é computacionalmente mais barato.

Se queremos utilizar apenas o *diagonal*, colocamos $it.dg = 1$ e $it.fg > it.max$. Se queremos utilizar apenas a *floresta geradora máxima*, colocamos $it.dg > it.max$ e $it.fg = 1$. Uma combinação que apresentou um bom desempenho é a dada por $it.dg = 1$ e $it.fg = 7$: até a sexta iteração do primal-dual, usamos o precondicionador *diagonal* e, a partir da sétima (inclusive), entramos com a *floresta geradora máxima* (de fato, a troca do precondicionador depende ainda se o número de iterações do gradiente conjugado já ultrapas-

sou $\sqrt{\widetilde{m}}$) .

As **tolerâncias** utilizadas pelo programa são (critério de parada):

$\varepsilon_f = 10^{-5}$: tolerância de factibilidade;
$\varepsilon_\mu = 10^{-7}$: tolerância para o parâmetro de centragem μ;
$\varepsilon_g = 10^{-5}$: tolerância para o gap de dualidade;
$\varepsilon_{gc} = 10^{-8}$: tolerância para o resíduo no método dos gradientes conjugados;
ε_{cos} : tolerância para o critério do cosseno no método dos gradientes conjugados. Em geral, tomamos $\varepsilon_{cos} = 10^{-8}$.

Quanto a esta última tolerância, observamos que, em algumas implementações este valor foi alterado, como mostraremos em alguns experimentos computacionais. Em outras implementações ainda, iniciamos com $\varepsilon_{cos} = 10^{-t}$, e, na seqüência de iterações do método primal-dual, fizemos $\varepsilon_{cos} \leftarrow 0.95\ \varepsilon_{cos}$. Resende e outros [58, 59, 61], para problemas de fluxo de custo mínimo com um único produto, sugerem $t = 3$. Castro [10] sugere $t = 2$, com o objetivo de reduzir ainda mais o número de iterações nos gradientes conjugados, mas alerta para o aumento do número de iterações no primal-dual, bem como sua perda de precisão.

O critério de parada utilizado é o descrito no Capítulo 3. Podemos trabalhar com tolerâncias mais acuradas, caso o problema assim o exija. No entanto, nos problemas obtidos do gerador, não achamos isso necessário, uma vez que conseguimos boas aproximações para as soluções ótimas com estes valores.

6.4 Experimentos computacionais com o gerador

Os experimentos computacionais que se seguem consideram os seguintes parâmetros:

$[cmin\ ,\ cmax] = [0\ ,\ 5]$
$[umin\ ,\ umax] = [1\ ,\ 5]$
$inft = 10^{32}$
$pcan = 1$
$fa = 1$

Para cada tamanho de problema (m, n, p), os resultados apresentados são dados pela média aritmética de 5 problemas gerados (*semente* = 32, 9753, 7319, 3715, 2681) .

Quando o ponto inicial não for especificado, o que está sendo usado é o "primeiro ponto inicial", descrito no Capítulo 3.

Vários experimentos mostrados aqui têm como referência os problemas da Tabela 6.1. Em todos estes problemas, fixamos o número de produtos p em 10.

A Tabela 6.1 mostra, em cada coluna, respectivamente: o número do problema, o número de nós, m , o número de arcos, n , e a dimensão do sistema linear que é resolvido pelo método dos gradientes conjugados a cada iteração do método primal-dual, $\widehat{m} = p(m-1) + n$.

Em várias tabelas apresentadas neste capítulo, aparecerão as colunas *cpu*, *it.pd* e *mit.gc*, onde:

cpu: tempo total de cpu gasto no método primal-dual (em segundos);

it.pd : número de iterações efetuadas no método primal-dual;

mit.gc : média aritmética do número de iterações efetuadas no método dos gradientes conjugados.

As medidas *cpu* e *it.pd* são comumente encontradas em comparações que se obtém de experimentos computacionais. Escolhemos ainda a quantidade *mit.gc* como uma medida de eficiência do método dos gradientes conjugados, ou do próprio precondicionador que está sendo utilizado. Se este número for "razoável", podemos considerar que o método dos gradientes conjugados teve um bom desempenho. Se ele chegar a $\widehat{m}$, temos o pior desempenho.

problema	m	n	$\widehat{m}$
1	10	20	110
2	20	40	230
3	20	50	240
4	40	60	450
5	50	80	570
6	50	100	590
7	80	100	890
8	100	150	1140
9	100	200	1190
10	100	250	1240
11	150	200	1690
12	150	250	1740
13	150	300	1790
14	200	300	2290
15	200	400	2390
16	300	400	3390
17	300	500	3490
18	300	600	3590
19	400	600	4590
20	400	800	4790
21	440	800	5190

Tabela 6.1: *problemas do gerador* ($p = 10$)

6.4.1 O parâmetro μ

Nos experimentos a seguir, comparamos duas possíveis expressões para o parâmetro de centragem μ — segundo sugestões que aparecem frequentemente na literatura. Na primeira delas, não consideramos as variáveis de folga das canalizações, isto é,

$$\mu^\ell = 0.1 \frac{\gamma^\ell}{(p+1)n}$$

e na segunda, estas variáveis foram consideradas, ou seja,

$$\mu^\ell = 0.1 \frac{\gamma^\ell}{(2p+1)n}$$

A Tabela 6.2 mostra os resultados do programa **mult** para alguns problemas, quando a primeira expressão é utilizada. A Tabela 6.3 mostra os resultados de **mult** para os mesmos problemas, quando a segunda expressão é utilizada.

A coluna $\widehat{m}$ indica a dimensão do sistema linear que é resolvido a cada iteração do método primal-dual.

m	n	p	$\widehat{m}$	*cpu*	*it.pd*	*mit.gc*
100	200	10	1190	32.6	22	84
100	200	20	2180	41.3	23	51
200	400	10	2390	142.8	27	146
300	600	10	3590	455.1	29	298
400	800	10	4790	506.8	27	258
300	400	20	6380	413.9	27	171
300	500	20	6480	510.6	29	173

Tabela 6.2: *resultados para* $\mu = 0.1 \frac{\gamma^\ell}{(p+1)n}$

Comparando as duas tabelas, podemos observar que o ganho da segunda expressão está no número médio de iterações do gradiente conjugado e, assim mesmo, esta diferença não é grande. Nas outras duas colunas, a primeira expressão se comporta um pouco melhor. Dada a semelhança destes (e de outros) resultados, optamos pela primeira expressão, que é a que está sendo utilizada em todos os outros experimentos deste capítulo.

m	n	p	$\widehat{m}$	*cpu*	*it.pd*	*mit.gc*
100	200	10	1190	33.1	22	81
100	200	20	2180	44.1	25	49
200	400	10	2390	151.2	30	141
300	600	10	3590	455.3	29	255
400	800	10	4790	510.2	27	244
300	400	20	6380	455.5	28	177
300	500	20	6480	515.7	31	166

Tabela 6.3: *resultados para* $\mu = 0.1\ \frac{\gamma^t}{(2p+1)n}$

6.4.2 Comparando precondicionadores

Uma comparação entre os precondicionadores *diagonal* e *floresta geradora máxima* pode ser vista na Tabela 6.4 e nos dois gráficos seguintes. Nestes experimentos, utilizamos como ponto inicial para o método primal-dual o primeiro ponto inicial descrito no Capítulo 3. Além disso, tomamos $\varepsilon_{cos} = 10^{-8}$ (fixo). Obtivemos uma boa aproximação para a solução ótima em todos os problemas.

Nesta tabela, temos as seguintes indicações para suas colunas:

só diagonal: indica que estamos utilizando apenas o precondicionador *diagonal* em todas as iterações do método primal-dual.

só floresta: indica que estamos utilizando o precondicionador *floresta geradora máxima* em todas as iterações do método primal-dual.

r: indica a seguinte razão:
(tempo de cpu usando só *diagonal*) / (tempo de cpu usando só *floresta*).

Observando a Tabela 6.4, claramente vemos que o precondicionador *diagonal* sozinho apresenta um desempenho pior: o tempo de cpu é bem maior do que quando utilizamos a *floresta geradora*, o número médio de iterações no método dos gradientes conjugados é bem mais elevado e o número de iterações no método primal-dual também é maior. No entanto, o método primal-dual converge mesmo quando o número de iterações do gradiente conjugado atinge seu limite máximo ($\widehat{m}$) .

problema	$\widehat{m}$	só diagonal			só floresta			r
		cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	
1	110	1.6	16	67	0.5	16	14	3.2
2	230	10.1	22	145	1.5	17	20	6.7
3	240	10.4	20	149	1.7	17	20	6.1
4	450	55.6	31	329	4.2	18	32	13.2
5	570	94.9	32	442	7.6	18	46	12.5
6	590	72.8	26	381	9.9	19	53	7.4
7	890	257.9	36	694	18.5	19	74	13.9
8	1140	440.5	38	889	34.9	21	97	12.6
9	1190	427.1	32	811	31.7	21	78	13.5
10	1240	365.8	29	730	44.3	23	89	8.3
11	1690	868.1	37	1238	60.2	21	111	14.4
12	1740	1190.8	44	1325	84.9	23	142	14.0
13	1790	1063.6	38	1231	91.3	23	138	11.6
14	2290	1085.5	29	1417	123.7	23	157	8.8
15	2390	1189.6	29	1399	190.1	25	202	6.3
16	3390	2420.8	31	2110	342.9	24	270	7.1
17	3490	2644.1	32	2010	333.9	25	259	7.9
18	3590	2637.8	31	1894	453.1	27	308	5.8
19	4590	4017.2	31	2361	741.3	27	409	5.4
20	4790	4412.8	31	2429	735.9	26	372	6.0
21	5190	4033.0	29	2255	868.9	27	414	4.6

Tabela 6.4: *comparação entre precondicionadores*

Nos gráficos dados pelas Figuras 6.1 e 6.2, cujos dados foram retirados da Tabela 6.4, vemos, respectivamente, a razão entre os tempos de cpu entre os dois precondicionadores puros e o número médio de iterações efetuadas pelo método dos gradientes conjugados. No primeiro, podemos observar que esta razão é sempre maior que 1 (na verdade, maior que 3), mostrando claramente o melhor comportamento da *floresta geradora máxima*. No segundo gráfico, podemos comparar a média de iterações do gradiente conjugado dos dois precondicionadores com a dimensão do sistema linear ($\widehat{m}$) que é resolvido a cada iteração do método primal-dual, ou seja, o número máximo de iterações permitidas no gradiente conjugado. Observamos que, na média, este máximo nunca é atingido. No entanto, o precondicionador *diagonal* atinge este número máximo $\widehat{m}$ em muitos problemas, fato que não ocorre com o precondicionador *floresta geradora máxima*.

No gráfico dado pela Figura 6.3, podemos observar a evolução do número de iterações no gradiente conjugado em cada iteração do método primal-dual, quando resolvemos um dos cinco problemas de número 20 ($m = 400$, $n = 800$, $p = 10 \Rightarrow \widehat{m} = 4790$) , utilizando os dois preconcicionadores puros. Notamos que o comportamento da *floresta geradora máxima* é bem melhor do que o *diagonal*, principalmente nas últimas iterações do método primal-dual. Com a *floresta geradora máxima* temos 26 iterações do primal- dual, contra 37 para o *diagonal*. Neste último, a partir da iteração 19 do primal-dual, foi atingido o número máximo de iterações no gradiente conjugado: 4790. Este fato não acontece quando utilizamos a *floresta geradora máxima*, embora, com este precondicionador, o número de iterações do gradiente conjugado tenha chegado a 1090 (isoladamente) na iteração 24 do primal-dual. Outro fato que podemos observar é que, nas primeiras 4 iterações do primal-dual, o número de iterações do gradiente conjugado é idêntico nos dois precondicionadores (53, 28, 54, 77). Isto mostra que poderíamos ter utilizado o *diagonal* até a quarta iteração, inclusive, introduzindo a *floresta geradora máxima* a partir da quinta iteração do primal-dual: manteríamos o mesmo número de iterações do primal-dual e, provavelmente, teríamos um ganho no tempo.

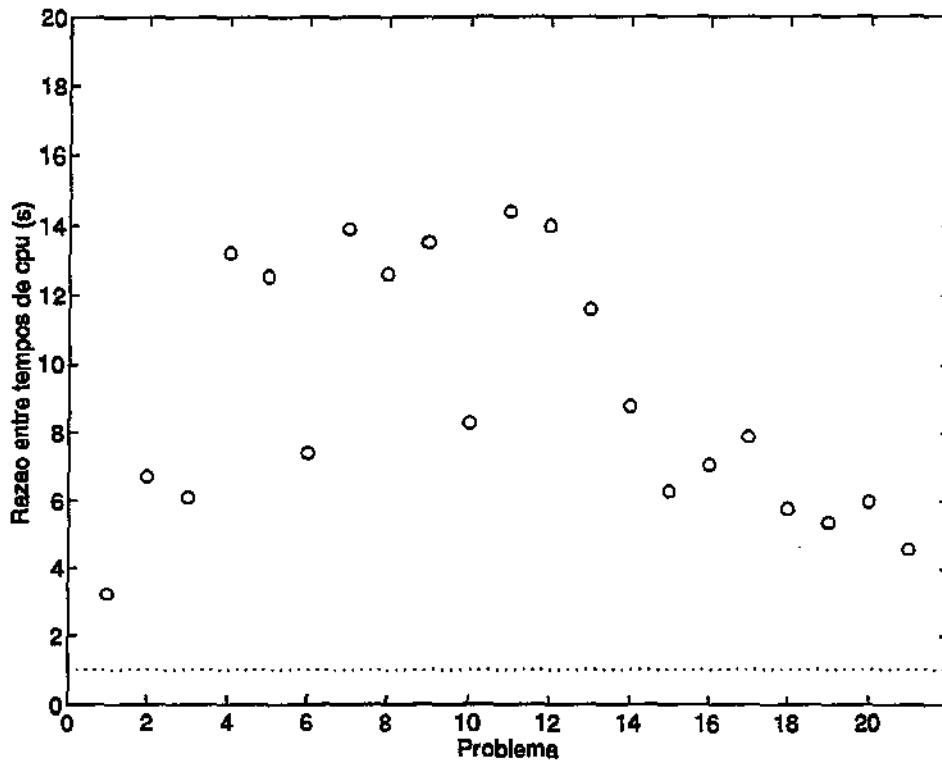


Figura 6.1: *razão entre os tempos de cpu nos precondicionadores*

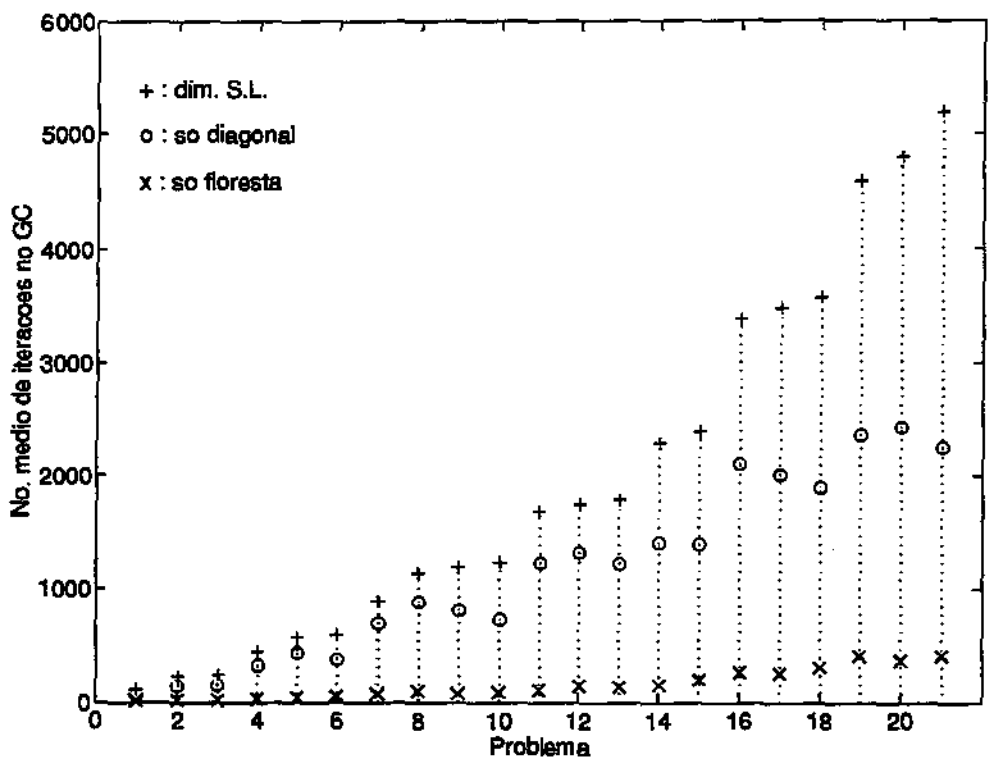


Figura 6.2: *número médio de iterações no gradiente conjugado*

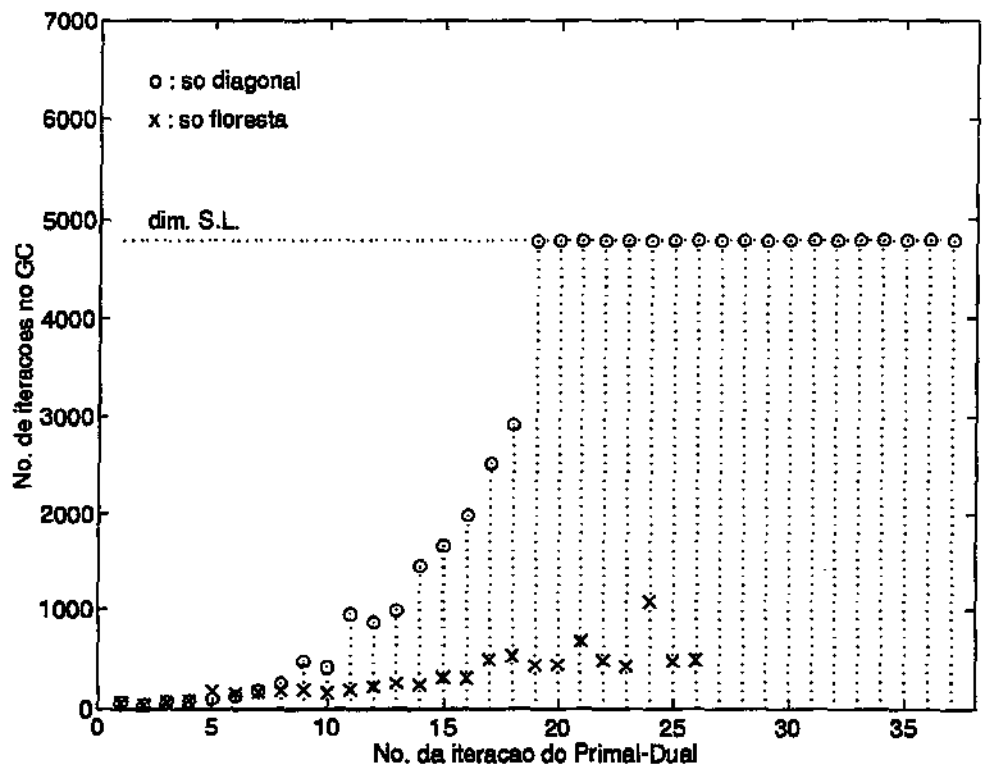


Figura 6.3: *problema* 400 x 800 x 10 : *evolução do número de iterações no gradiente conjugado*

Passamos agora à comparação entre a *floresta geradora máxima* pura e a combinada (diagonal + floresta).

Uma vez que a *floresta geradora máxima* apresenta um desempenho bem melhor do que o *diagonal*, a idéia de se fazer a combinação dos dois precondicionadores se baseia no fato de que, nas primeiras iterações do método primal-dual, temos pouca informação disponível para construir florestas geradoras que se aproximem das florestas ótimas dos blocos.

O precondicionador *floresta geradora máxima* utiliza um tempo maior para ser construído do que o *diagonal*, além de exigir a resolução de dois sistemas lineares triangulares, contrapondo com o sistema linear diagonal do precondicionador *diagonal*.

É difícil encontrar o número ideal da iteração do método primal-dual ($it.fg$) onde devemos efetuar a troca do precondicionador. Para um certo grupo de problemas, podemos concluir por um número médio para $it.fg$. Nos muitos experimentos efetuados, pudemos concluir que existe este valor ideal para $it.fg$, e que ele ocorre nas primeiras iterações do primal-dual, mas não necessariamente é o mesmo para os vários tipos e tamanhos de problemas.

A Tabela 6.5 faz uma comparação entre a floresta pura e a combinada. As colunas $\widetilde{m}$, *cpu*, *it.pd* e *mit.gc* já foram descritas anteriormente, assim como a coluna *só floresta*. As colunas m, n, p indicam o tamanho do problema (número de nós, de arcos e de produtos, respectivamente). A coluna *diagonal + floresta* indica que estamos utilizando o precondicionador *diagonal* nas primeiras iterações do método primal-dual e a *floresta geradora máxima* nas últimas iterações. A coluna $it.fg$ indica em qual iteração do método primal-dual foi feita a troca de precondicionador.

Observamos, da Tabela 6.5, que sempre é possível encontrar um valor para o parâmetro $it.fg$ de modo que o precondicionador combinado utilize um tempo de cpu menor do que quando utilizamos a *floresta geradora* pura desde o início. Os valores presentes na última coluna desta tabela são alguns "bons" valores obtidos para o parâmetro $it.fg$. De fato, é mais vantajoso utilizar o precondicionador combinado quando o tamanho do problema cresce, pois florestas maiores levam mais tempo para serem construídas. Quanto ao número de iterações do método primal-dual (*it.pd*) e o número médio de iterações no gradiente conjugado (*mit.gc*), não existe diferença significativa entre um caso e outro.

Podemos também observar o comportamento do tempo de cpu (*cpu*),

m	n	p	$\widehat{m}$	só floresta			diagonal + floresta			
				cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*it.fg*
100	200	10	1190	31.7	21	78	30.9	24	69	3
100	200	20	2180	44.8	22	56	38.4	21	51	6
100	200	30	3170	88.8	23	71	66.4	23	55	6
150	250	10	1740	84.9	23	142	82.2	23	149	7
200	300	10	2290	123.7	23	157	107.8	22	153	6
200	300	20	4280	234.9	25	142	223.6	24	151	7
200	300	30	6270	261.7	25	106	258.7	25	106	5
200	400	10	2390	190.1	25	202	159.4	24	189	6
200	400	20	4380	297.1	26	130	228.3	25	127	7
300	400	10	3390	342.9	24	270	260.4	23	234	6
300	400	20	6380	487.4	24	209	461.7	25	199	7
300	400	30	9370	1040.3	28	236	889.2	28	231	7
300	500	10	3490	333.9	25	259	284.5	25	206	5
300	600	10	3590	453.1	27	308	302.7	24	232	6
400	600	10	4590	741.3	27	409	649.9	26	381	7
400	800	10	4790	735.9	26	372	693.5	26	365	6
440	800	10	5190	868.9	27	414	664.4	25	349	5

Tabela 6.5: *comparação entre floresta pura e combinada*

do número de iterações do método primal-dual (*it.pd*) e do número médio de iterações do gradiente conjugado (*mit.gc*) em um problema particular, onde alteramos o valor do parâmetro $it.fg$. As dimensões do problema são $m = 200$, $n = 400$, $p = 20$ e $\widehat{m} = 4380$. As primeiras iterações continuam sendo efetuadas com o precondicionador *diagonal* ($it.dg = 1$). No gráfico dado pela Figura 6.4, temos os seguintes valores para $it.fg$: 1, 5, 6, 7, 8, 10 e 12 . O melhor tempo ocorreu para $it.fg = 7$ (195.3 segundos). O pior tempo ocorreu para $it.fg = 12$ (323.4 segundos). Para $it.fg = 1$ (só floresta), temos o terceiro maior tempo de cpu: 227.2 segundos. O número médio de iterações do gradiente conjugado tem quase que o mesmo comportamento do tempo de cpu. No entanto, o número de iterações do primal-dual praticamente não se altera.

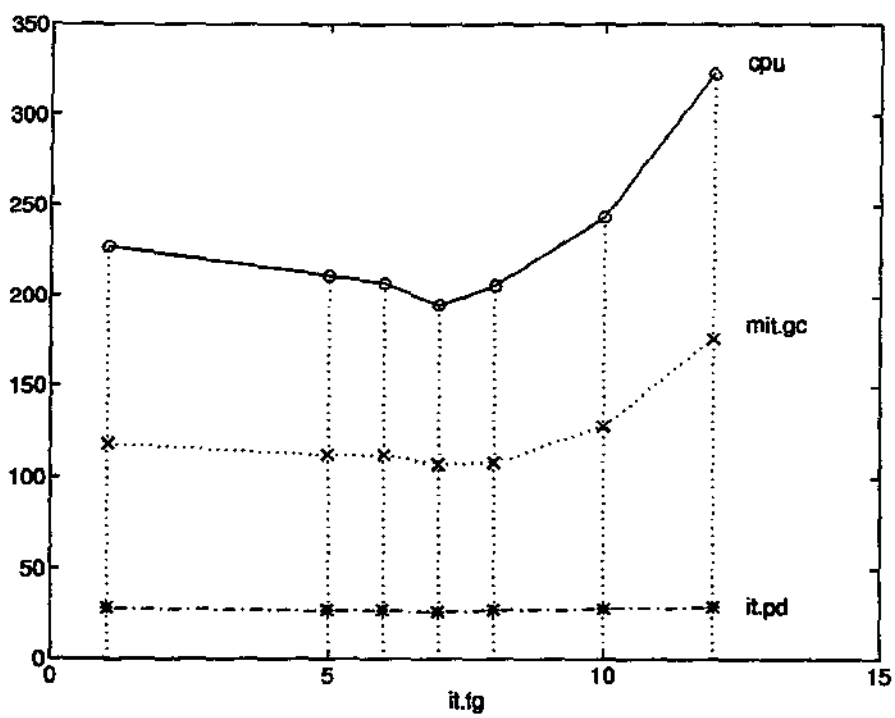


Figura 6.4: *problema* 200 x 400 x 20 : *variação do parâmetro* $it.fg$

6.4.3 Comparando pontos iniciais

Uma comparação entre os três pontos iniciais discutidos no Capítulo 3 pode ser vista na Tabela 6.6 e nos dois gráficos seguintes, cujos dados foram retirados desta tabela. Aqui, estamos tomando 15 problemas da Tabela 6.1. Além disso, estamos utilizando os seguintes valores para os parâmetros *it.dg* e $it.fg$: $it.dg = 1$ e $it.fg = 7$. Tomamos $\varepsilon_{cos} = 10^{-8}$ (fixo). Encontramos uma boa aproximação para a solução ótima em todos os casos.

As colunas "1^0 ponto inicial", "2^0 ponto inicial" e "3^0 ponto inicial", se referem aos três pontos iniciais descritos no Capítulo 3, respectivamente.

problema	1^0 ponto inicial			2^0 ponto inicial			3^0 ponto inicial		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
2	2.0	17	34	2.1	16	39	2.5	18	30
4	6.3	18	59	6.6	18	61	7.2	19	49
6	10.9	20	64	11.3	19	66	13.2	21	58
9	32.5	21	88	33.9	20	93	37.9	23	78
11	66.3	21	138	67.7	22	140	80.7	23	127
12	82.2	23	149	93.7	22	158	96.2	24	142
13	92.2	23	146	95.8	24	152	101.8	25	135
14	122.9	23	165	128.7	23	177	135.4	24	157
15	180.2	25	200	181.5	24	210	196.5	27	186
16	294.7	24	268	302.8	23	278	334.3	26	258
17	320.1	25	255	315.8	24	256	366.3	27	247
18	468.3	28	305	468.9	27	308	494.7	29	298
19	649.9	26	381	669.4	26	387	763.2	28	379
20	731.9	27	370	796.9	28	395	789.0	28	360
21	762.8	26	384	798.1	26	392	823.0	27	371

Tabela 6.6: *comparação entre pontos iniciais*

Podemos observar que o tempo total gasto pelo primeiro e segundo pontos é bastante parecido, ao passo que o terceiro ponto utiliza um tempo um pouco maior. Esta diferença diminui quando o tamanho do problema aumenta. Provavelmente isto se deva ao fato de que o terceiro ponto trabalha com mais infactibilidades do que os dois primeiros, conforme discutido no Capítulo 3.

Além disso, para este ponto é necessário resolver dois sistemas lineares de dimensão $\widehat{m}$ com a matriz $\hat{A}\hat{A}'$, onde é utilizado o método dos gradientes conjugados sem precondicionamento (poderia ser utilizado o precondicionador *diagonal*, por exemplo), e ainda efetua três produtos matriz-vetor com a matriz $\hat{A}$. Por estes motivos, a construção deste ponto é computacionalmente mais cara do que a dos dois primeiros.

Embora o terceiro ponto seja mais elaborado, podemos observar na Tabela 6.6 que o número de iterações efetuadas pelo método primal-dual não diminui com este ponto: temos uma ou duas iterações a mais. Quanto a este item, o segundo ponto se comporta um pouco melhor.

No entanto, quanto ao número médio de iterações no método dos gradientes conjugados (*mit.gc*), o terceiro ponto tem um comportamento melhor do que os dois primeiros, indicando provavelmente uma maior estabilidade.

No gráfico dado pela Figura 6.5, temos o número do problema contra o tempo de cpu (em segundos) gasto pelo método primal-dual de pontos interiores, quando este é iniciado por cada um dos três pontos iniciais. Notamos o comportamento um pouco melhor do primeiro ponto em relação aos outros dois. Além disso, dada a sua simplicidade, optamos por utilizá-lo na maioria das vezes.

No gráfico dado pela Figura 6.6, estamos comparando o número médio de iterações efetuadas no método dos gradientes conjugados em cada um dos problemas, quando o método primal-dual é iniciado com cada um dos três pontos iniciais. Notamos que o melhor comportamento é o do terceiro ponto inicial, embora a diferença não seja muito significativa. Isto provavelmente se deve ao fato de que este ponto está mais próximo da trajetória central do que os outros dois.

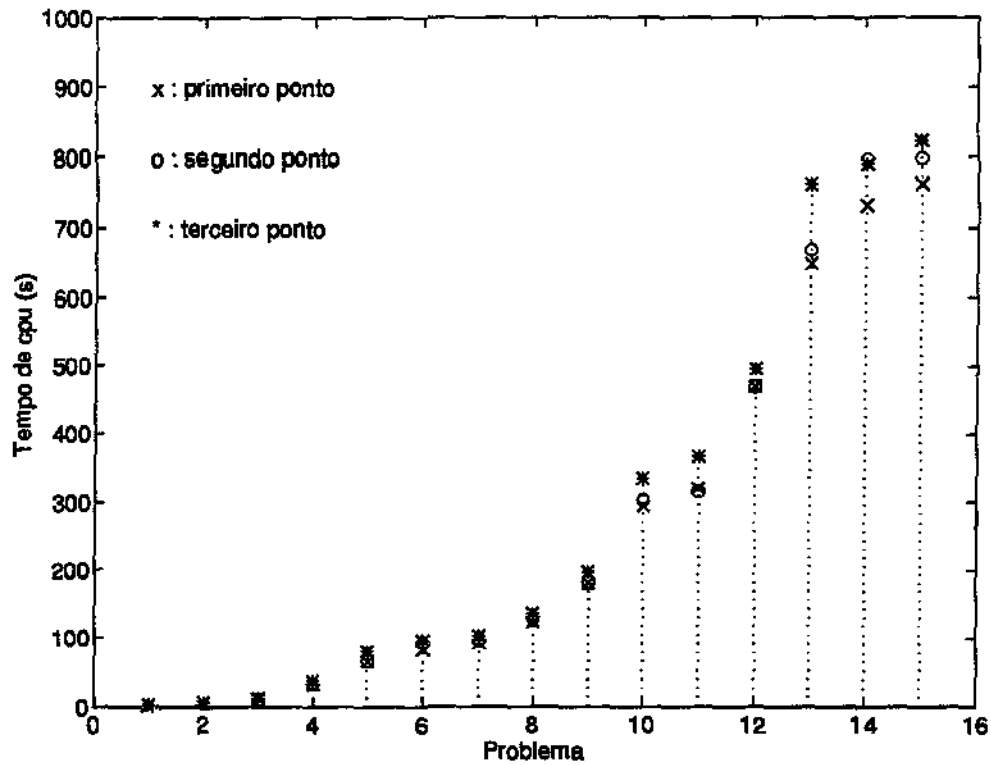


Figura 6.5: *comparação entre os tempos de cpu nos três pontos iniciais*

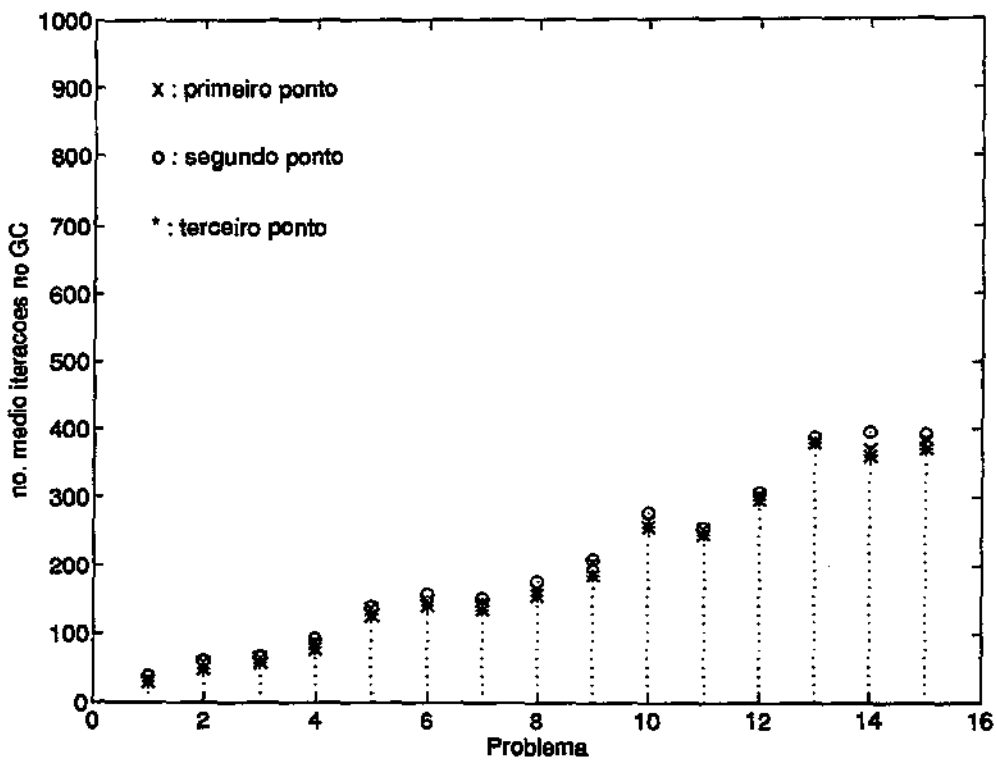


Figura 6.6: *número médio de iterações no gradiente conjugado, com os três pontos iniciais*

6.4.4 Critério de parada do gradiente conjugado

Consideramos três critérios de parada no método dos gradientes conjugados:

(i) parada pelo resíduo;

(ii) parada pelo cosseno;

(iii) número máximo de iterações efetuadas.

O critério (iii) é usado apenas como salvaguarda, caso os outros dois não tenham ocorrido antes. Este critério simplesmente limita o número de iterações do gradiente conjugado à dimensão do sistema linear, isto é, $\widehat{m}$.

Em cada um dos outros dois, consideramos dois casos:

(i)resíduo $\begin{cases} (i_1) & \text{resíduo absoluto} \\ (i_2) & \text{resíduo relativo} \end{cases}$

(ii)cosseno $\begin{cases} (ii_1) & \text{cosseno fixo} \\ (ii_2) & \text{cosseno variável} \end{cases}$

(i) Parada pelo resíduo

O critério de parada mais comumente encontrado na literatura para o método dos gradientes conjugados é o chamado *critério do resíduo:* se ℓ é a iteração corrente do método dos gradientes conjugados, é verificado se $(r')^\ell \, r^\ell$ é "suficientemente pequeno". Nossos experimentos verificaram este critério de duas maneiras, que chamamos de *resíduo absoluto* e *resíduo relativo.*

(ii_1) resíduo absoluto

Neste critério, a parada do gradiente conjugado se dá quando $(r')^\ell \, r^\ell \leq \varepsilon_{gc}$. Usualmente, tomamos $\varepsilon_{gc} = 10^{-8}$, embora este valor possa ser alterado.

(ii_2) resíduo relativo

Lembrando que o sistema linear resolvido pelo método dos gradientes conjugados é $\widehat{Q} \, \Delta\widehat{y} = \widehat{q}$, este critério considera a parada do gradiente conjugado quando $(r')^\ell \, r^\ell \leq \varepsilon_{gc} \, (\widehat{q}\widehat{q})$. Embora este critério seja menos usado do que o anterior (ao menos explicitamente, não encontramos referências quanto ao

seu uso), ele é bastante útil em nosso caso, uma vez que o lado direito do sistema, $\hat{q}$, costuma apresentar uma norma não muito pequena. Também aqui, normalmente, tomamos $\varepsilon_{gc} = 10^{-8}$.

Uma comparação entre estas duas medidas do resíduo pode ser vista na Tabela 6.7. Podemos observar que o número de iterações do primal-dual é praticamente o mesmo nos dois casos. No entanto, o tempo de cpu e o número médio de iterações do gradiente conjugado são menores quando tomamos o resíduo relativo. Isto se deve ao fato de que os valores $\hat{q}^t\hat{q}$ são da ordem de 10^4 em quase todas as iterações do método primal-dual e em quase todos os problemas considerados na Tabela 6.7 (também aparecem alguns valores da ordem de 10^3 ou 10^5). Como $\varepsilon_{gc} = 10^{-8}$, os resíduos relativos ao final de cada execução do gradiente conjugado ficam na ordem de 10^{-4} ou 10^{-5} (ao invés de 10^{-9} do caso dos resíduos absolutos). Daí termos o número (médio) de iterações do gradiente conjugado diminuído e, consequentemente, o tempo de cpu.

Estas observações podem reforçadas nos gráficos dados pelas Figuras 6.7, 6.8, 6.9, 6.10, onde colocamos, para os problemas da Tabela 6.7, respectivamente:

- o tempo de cpu,
- o número médio de iterações do gradiente conjugado,
- a razão: *cpu* com resíduo absoluto / *cpu* com resíduo relativo,
- a razão: *mit.gc* com resíduo absoluto / *mit.gc* com resíduo relativo.

Podemos observar, nas duas últimas figuras, que as razões permanecem entre 1 e 2, indicando claramente que o resíduo relativo é mais vantajoso do que o absoluto. Assim concluímos que, usando o critério do resíduo isoladamente, é conveniente utilizar o resíduo relativo.

problema	resíduo absoluto			resíduo relativo		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
1	0.8	16	30	0.7	16	27
2	2.3	16	48	1.8	17	37
4	7.2	17	91	5.7	17	71
6	15.9	20	116	9.8	19	68
8	44.7	20	186	35.8	20	134
9	45.5	21	148	29.5	21	92
10	45.2	22	125	38.8	22	92
11	89.6	20	168	59.5	21	148
12	141.1	22	248	79.6	23	156
13	121.2	23	227	83.9	23	155
14	144.4	21	222	122.6	23	176
15	212.5	23	303	155.0	24	207
16	371.2	23	420	280.0	24	295
17	376.8	24	374	329.3	25	275
18	626.9	25	550	446.7	27	328
19	813.9	24	629	576.2	25	408
20	1140.6	25	723	698.4	27	405
21	1235.5	24	799	775.6	26	449

Tabela 6.7: *comparação entre resíduo absoluto e relativo*

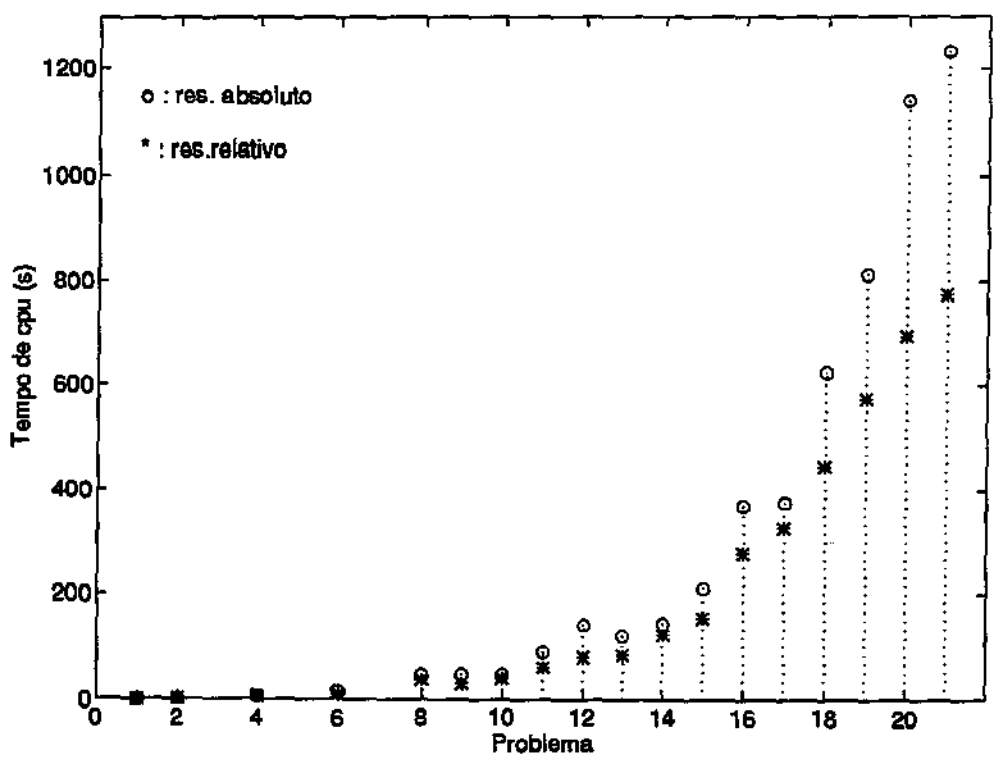


Figura 6.7: *tempo de cpu: resíduo absoluto* × *relativo*

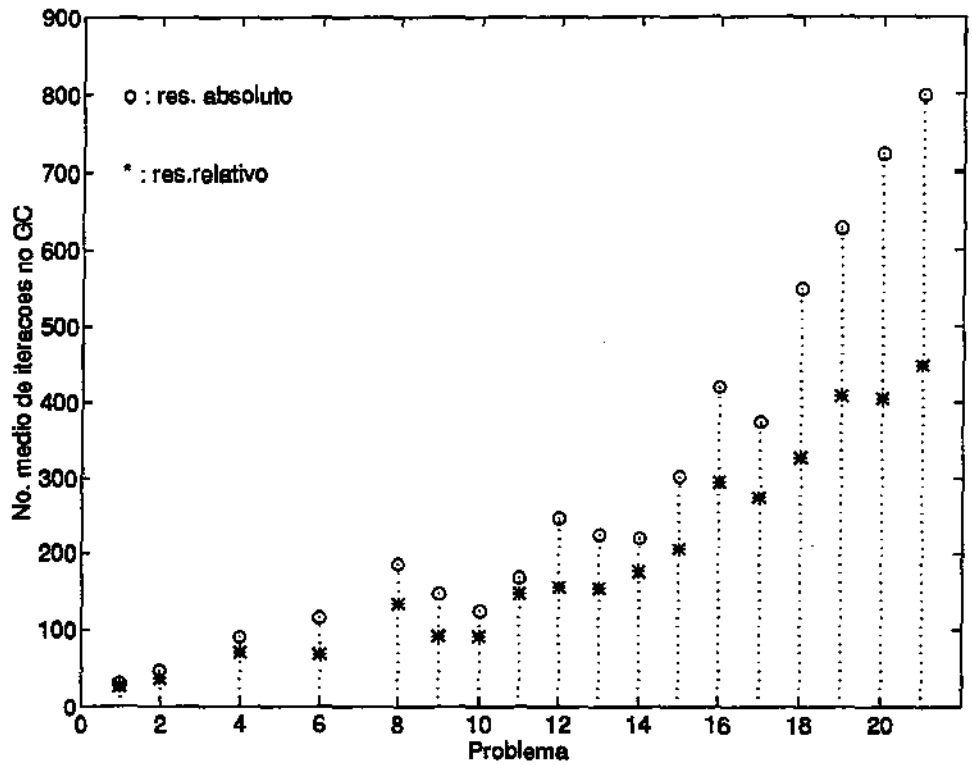


Figura 6.8: *no. médio de iterações do GC: resíduo absoluto × relativo*

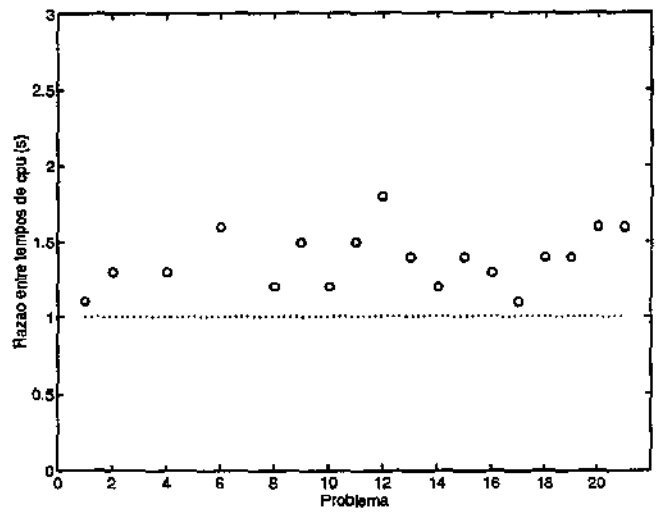


Figura 6.9: *razão entre tempos de cpu: resíduo absoluto e relativo*

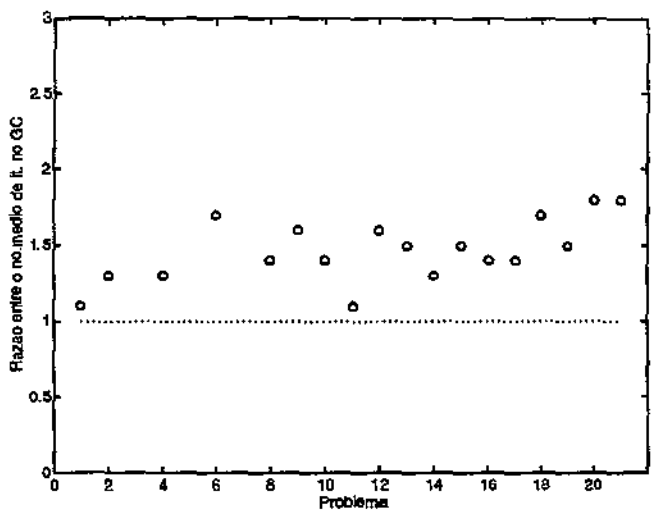


Figura 6.10: *razão entre no. médio de it. do GC: resíduo absoluto e relativo*

(ii) **Parada pelo cosseno**

Conforme sugerido por Resende e outros [58, 59, 61] (para problemas de fluxo de custo mínimo com um único produto), podemos utilizar também, como critério de parada na iteração l do método dos gradientes conjugados, o cosseno do ângulo σ formado por $\hat{q}$ e $\widehat{Q}(\Delta\hat{y})^l$ (Capítulo 4). De fato, os experimentos computacionais mostraram que este critério de parada geralmente é alcançado antes que o critério usual (critério do resíduo) o seja.

(ii_1) **cosseno fixo**

Neste caso, a parada pelo cosseno se dá quando $|1 - \cos\sigma| < \varepsilon_{\cos}$. Usualmente, tomamos $\varepsilon_{cos} = 10^{-8}$.

Se relaxamos $\varepsilon_{\cos}$, diminuímos o número de iterações no método dos gradientes conjugados. O critério do resíduo (absoluto) não fica satisfeito e, parar o método dos gradientes conjugados com resíduos grandes pode nos levar a direções ruins (mal aproximadas). Isto geralmente resulta num aumento do número de iterações no método primal-dual de pontos interiores, mas, não necessariamente num aumento do tempo total de cpu. Precisamos então encontrar um ponto de equilíbrio para obtermos direções "razoáveis" ao mesmo tempo que tentamos obter um menor número de iterações no método dos gradientes conjugados.

Estas observações podem ser conferidas na Tabela 6.8, onde colocamos três valores para ε_{cos} : 10^{-8}, 10^{-7}, 10^{-5} . As subcolunas são as mesmas já descritas anteriormente. Durante estes experimentos, foram medidos os resíduos ao final dos gradientes conjugados, isto é, $r'r$. Para $\varepsilon_{\cos} = 10^{-8}$, estes resíduos foram da ordem de 10^{-5} até o problema 7 e da ordem de 10^{-4} nos demais problemas. Para $\varepsilon_{\cos} = 10^{-7}$, tivemos da ordem de 10^{-4} até o problema 5, da ordem de 10^{-3} até o problema 17 e da ordem de 10^{-2} nos demais problemas. Já para $\varepsilon_{\cos} = 10^{-5}$, estes resultados são da ordem de de 10^{-2} até o problema 5, da ordem de 10^{-1} até o problema 17 e da ordem de 1.5 nos demais problemas. Em todos os experimentos efetuados, utilizamos o primeiro ponto inicial, $it.dg = 1$ e $it.fg = 7$.

Os dados dos próximos três gráficos foram retirados da Tabela 6.8.

No gráfico dado pela Figura 6.11, observamos o crescimento do tempo de cpu quando o valor de $\varepsilon_{\cos}$ diminui.

Na Figura 6.12, fica mais clara a diferença entre o número médio de iterações no gradiente conjugado à medida que aumentamos $\varepsilon_{\cos}$. No entanto,

a diminuição deste número de iterações nos leva a resíduos grandes no final do método dos gradientes conjugados, isto é, $r^t r >> 10^{-8}$, acarretando alguma perda de factibilidade da solução.

Na Figura 6.13, tomamos apenas os casos $\varepsilon_{\cos} = 10^{-8}$ e $\varepsilon_{\cos} = 10^{-5}$ para comparar o número de iterações do método primal-dual. Note que, para $\varepsilon_{\cos} = 10^{-5}$ este número passa de 40 nos problemas maiores, mas nunca chega a 30 para $\varepsilon_{\cos} = 10^{-8}$.

problema	$\varepsilon_{\cos} = 10^{-8}$			$\varepsilon_{\cos} = 10^{-7}$			$\varepsilon_{\cos} = 10^{-5}$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
1	0.7	16	25	0.7	16	21	0.5	18	12
2	2.0	17	34	1.8	17	29	1.3	19	16
3	2.2	17	30	1.9	17	26	1.5	20	14
4	6.3	18	59	4.9	19	42	3.5	22	22
5	10.4	18	75	8.1	19	56	5.1	22	24
6	10.9	20	64	11.3	20	53	6.7	24	27
7	24.1	19	109	20.1	19	88	15.5	26	45
8	40.6	21	125	32.7	21	98	20.0	26	44
9	32.5	21	88	27.6	22	69	19.8	28	36
10	39.0	22	86	34.8	23	73	21.1	29	33
11	66.3	21	138	53.4	21	109	46.2	31	57
12	82.2	23	149	68.4	24	115	52.7	33	57
13	92.2	23	146	70.3	25	98	71.7	39	63
14	122.9	23	165	93.7	23	122	85.8	33	67
15	180.2	25	200	154.7	28	152	155.1	41	95
16	294.7	24	268	272.3	26	221	196.2	34	107
17	320.1	25	255	277.9	27	191	214.8	35	102
18	468.3	28	305	372.0	29	208	264.2	40	112
19	649.9	26	381	501.4	28	271	466.6	42	153
20	731.9	27	370	711.4	30	307	634.4	43	195
21	762.8	26	384	725.9	29	326	760.1	43	206

Tabela 6.8: *comparação entre tolerâncias para o cosseno*

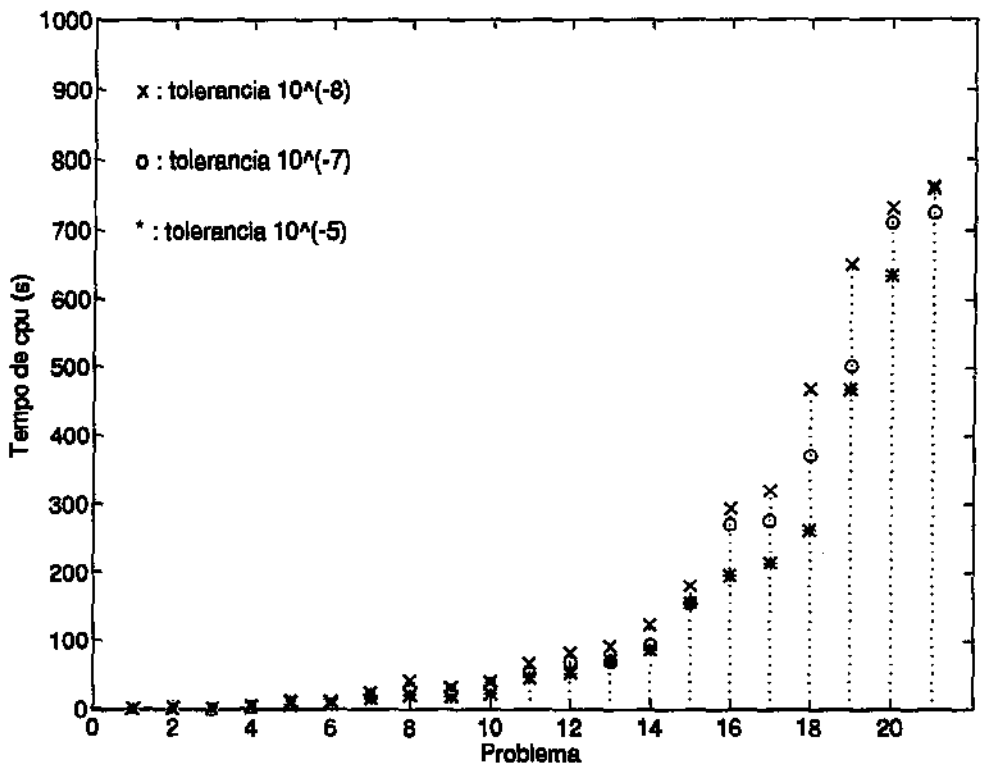


Figura 6.11: *comparação entre tempos de cpu variando* $\varepsilon_{\cos}$

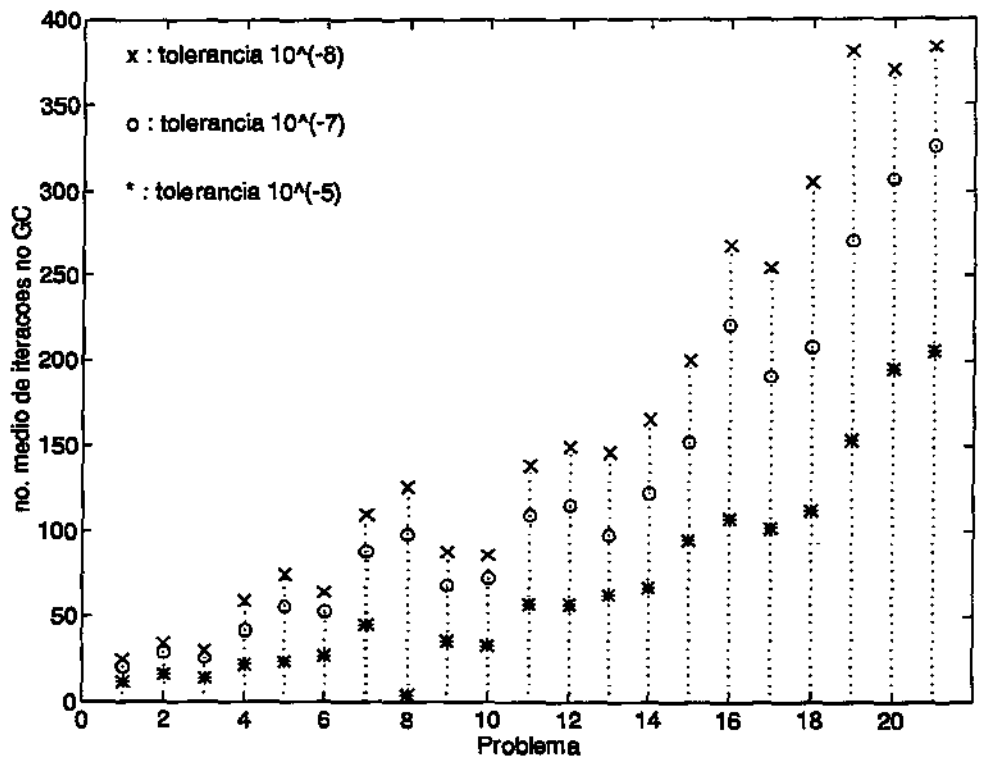


Figura 6.12: *número médio de iterações no gradiente conjugado variando* ε_{cos}

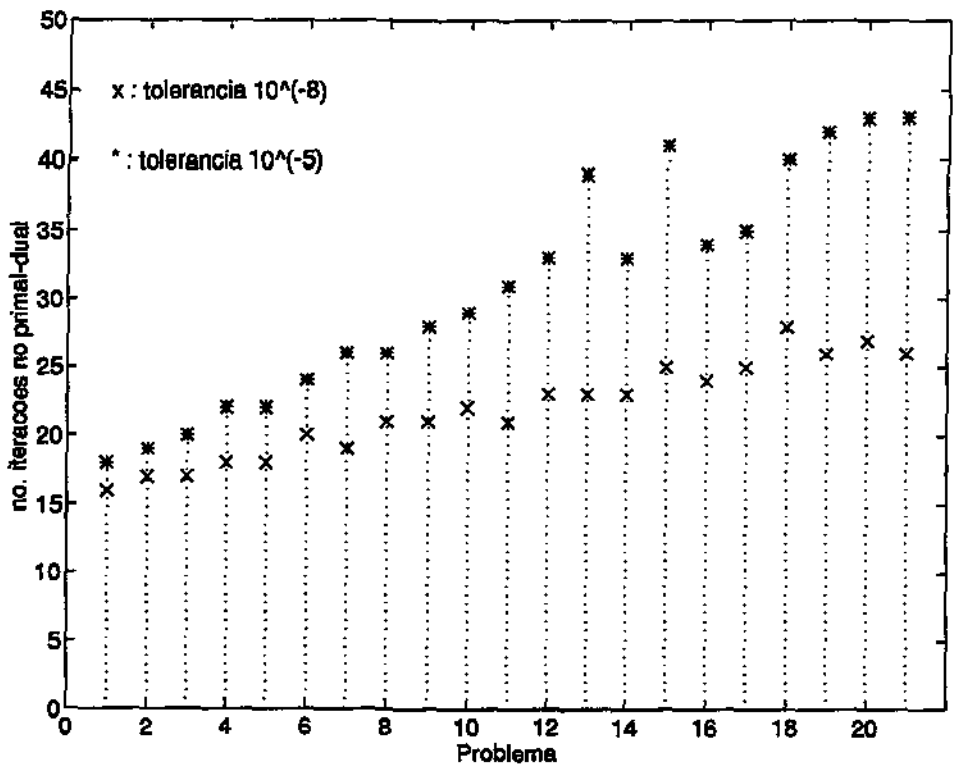


Figura 6.13: *número de iterações no primal-dual variando* $\varepsilon_{\cos}$

(ii_2) cosseno variável

Nos próximos experimentos, não mantemos mais ε_{cos} fixo: ao invés disso, iniciamos ε_{cos} com um certo valor e, no decorrer das iterações do método primal-dual, fazemos $\varepsilon_{cos} \leftarrow 0.95\ \varepsilon_{cos}$, conforme sugerido por Resende e outros em [59], por exemplo. Mantivemos fixo o valor de 0.95, mas uma redução maior poderia ser testada. Esta alteração nos parece bastante razoável uma vez que, à medida que o número de iterações do método primal-dual aumenta, o valor de ε_{cos} diminui. Nossa dificuldade está em saber qual é o melhor valor inicial para ε_{cos}, de modo que o número de iterações no método dos gradientes conjugados e no método primal-dual sejam razoáveis. Novamente, se relaxamos o valor inicial de ε_{cos}, o número de iterações do gradiente conjugado diminui, mas as direções geradas podem não ser boas, aumentando assim o número de iterações do método primal-dual.

Na Tabela 6.9, a coluna "sem cosseno" mostra os resultados obtidos quando o critério de parada pelo cosseno é eliminado do método dos gradientes conjugados. Neste caso, a parada do gradiente conjugado se dá com o critério do *resíduo absoluto*, isto é, quando $r'r < 10^{-8}$. As outras duas colunas mostram os resultados quando iniciamos ε_{cos} com 10^{-7} e 10^{-5}, respectivamente. Os valores desta tabela podem ser comparados com os da Tabela 6.8. Os resultados são bem semelhantes: vemos um aumento no tempo de cpu na Tabela 6.9, decorrente do aumento do número médio de iterações do gradiente conjugado, uma vez que ε_{cos} vai diminuindo a cada iteração do primal-dual. Isto causa uma redução nos resíduos (isto é, $r'r$) ao final do gradiente conjugado: para $\varepsilon_{cos} = 10^{-7}$ (inicial), estes resíduos foram da ordem de 10^{-5} até o problema 6, da ordem de 10^{-4} até o problema 15 e da ordem de 10^{-3} nos demais problemas. Para $\varepsilon_{cos} = 10^{-5}$ (inicial), tivemos da ordem de 10^{-3} até o problema 6, da ordem de 10^{-2} até o problema 12 e da ordem de 10^{-1} nos demais problemas. Quanto ao número de iterações do primal-dual, temos uma pequena redução.

Fizemos ainda outros experimentos relaxando mais o valor inicial de ε_{cos}, mas os resultados não foram animadores. Por exemplo, iniciando com $\varepsilon_{cos} = 10^{-3}$, os problemas 9, 14 e 16 apresentaram, ao final dos gradientes conjugados, resíduos (absolutos) da ordem de 10. Consequentemente, foram efetuadas todas as 50 iterações do método primal-dual sem que a solução ótima fosse alcançada (este fato também ocorre em outros problemas). Isto se explica porque o gradiente conjugado foi abandonado muito cedo: o número

médio de iterações do gradiente conjugado para os problemas 9, 14 e 16 foi, respectivamente, 42, 58 e 139 contra 148, 222 e 420 que foram os valores obtidos quando o teste de parada pelo cosseno foi abandonado.

O gráfico dado pela Figura 6.14, cujos dados foram retirados da tabela 6.9, mostra uma comparação entre o número médio de iterações no método dos gradientes conjugados quando tomamos os problemas da Tabela 6.9, para $\varepsilon_{\cos} = 10^{-7}$ (inicial), para $\varepsilon_{\cos} = 10^{-7}$ fixo (Tabela 6.8) e quando não consideramos o teste de parada pelo cosseno. Podemos notar como os valores dados por este último são maiores do que os dois primeiros.

problema	sem cosseno			$\varepsilon_{\cos} = 10^{-7}$			$\varepsilon_{\cos} = 10^{-5}$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
1	0.8	16	30	0.7	16	21	0.5	18	13
2	2.3	16	48	1.8	17	29	1.3	19	16
4	7.2	17	91	5.0	19	45	3.3	21	24
6	15.9	20	116	9.7	20	55	7.1	23	29
8	44.7	20	186	34.7	21	105	22.2	26	49
9	45.5	21	148	29.5	22	75	20.2	26	41
10	45.2	22	125	35.2	23	76	28.1	30	43
11	89.6	20	168	56.1	22	114	40.7	27	61
12	141.1	22	248	76.1	23	130	52.9	30	67
13	121.2	23	227	79.0	25	112	71.9	36	70
14	144.4	21	222	97.0	23	130	93.1	32	85
15	212.5	23	303	177.3	27	172	113.0	34	88
16	371.2	23	420	309.8	25	238	211.0	33	135
17	376.8	24	374	332.7	27	242	229.0	34	132
18	626.9	25	550	359.8	27	242	434.3	41	187
19	813.9	24	629	504.1	27	290	569.6	40	212
20	1140.6	25	723	626.0	28	313	779.1	41	254
21	1235.5	24	799	657.3	26	325	943.5	42	260

Tabela 6.9: *outra comparação entre tolerâncias para o cosseno*

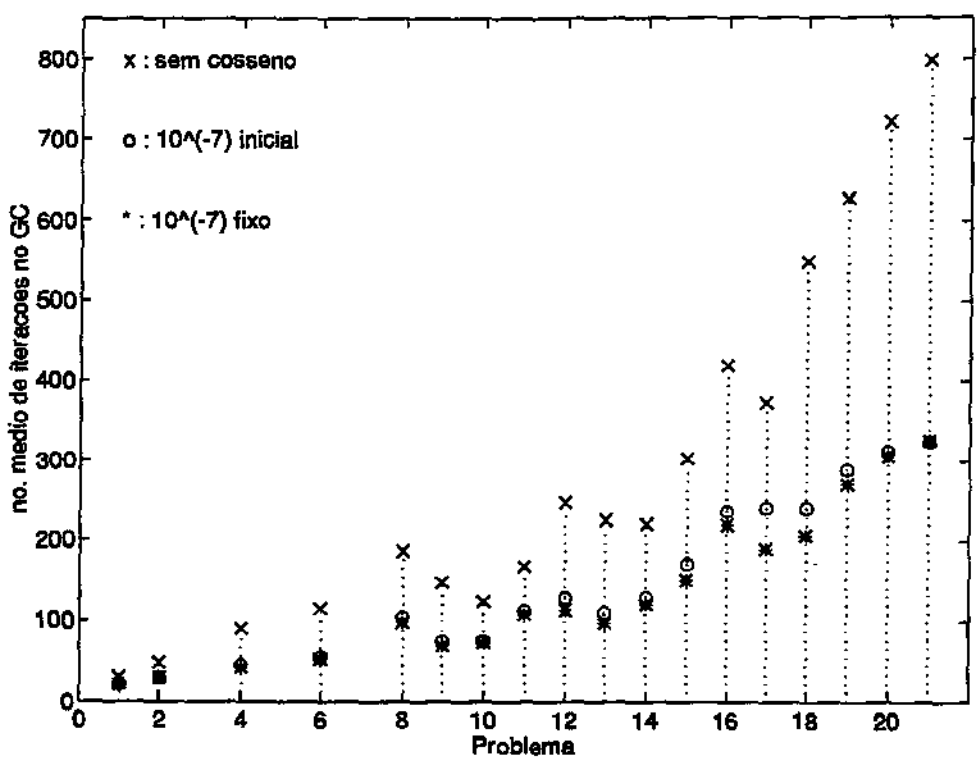


Figura 6.14: *outra comparação entre o número médio de iterações no gradiente conjugado alterando ε_{cos} : fixo e variável*

As Figuras 6.15, 6.16 e 6.17 mostram, respectivamente, o comportamento de um dos cinco problemas número 9 para dez valores iniciais de $\varepsilon_{\cos}$: 10^{-3}, 10^{-5}, 10^{-7}, 10^{-8}, 10^{-9}, 10^{-10}, 10^{-12}, 10^{-15}, 10^{-16}, e 10^{-17} .

O gráfico da Figura 6.15 mostra a ordem dos resíduos (isto é, $r'r$) obtidos ao final do método dos gradientes conjugados. Podemos notar a melhora dos resíduos à medida em que $\varepsilon_{\cos}$ inicial diminui. Note que, para $\varepsilon_{\cos}$ inicial 10^{-15}, 10^{-16}, 10^{-17} , os resíduos são da ordem de 10^{-9}, o que significa que o teste de parada pelo cosseno fica inócuo, ou seja, a parada do gradiente conjugado se dá pelo teste usual do resíduo.

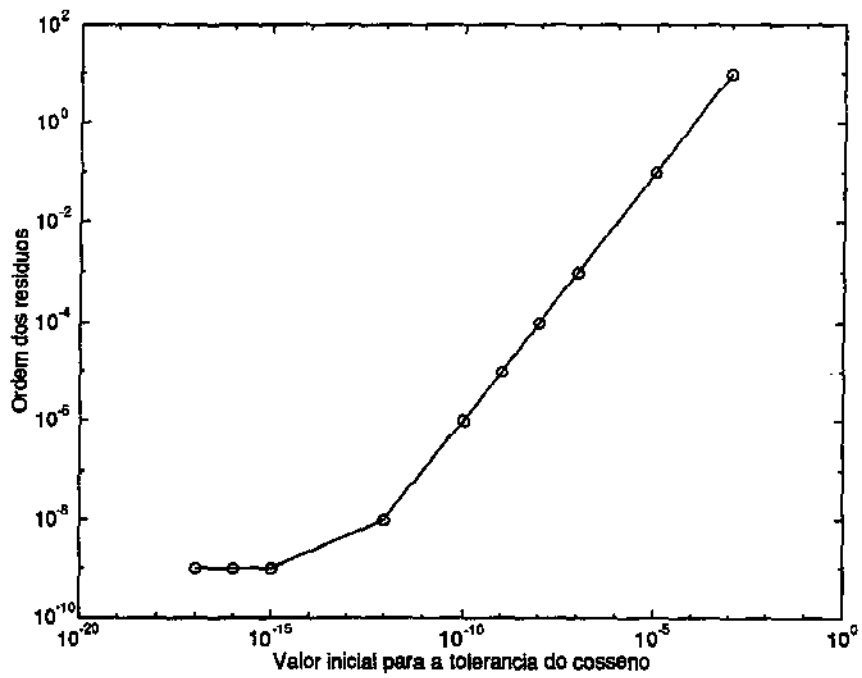


Figura 6.15: *problema* 200 x 400 x 10 : *ordem dos resíduos ao final do GC*

O gráfico da Figura 6.16 mostra o número médio de iterações efetuadas pelo gradiente conjugado para cada valor inicial de $\varepsilon_{\cos}$. Novamente, para os valores 10^{-15}, 10^{-16}, 10^{-17} , este número médio de iterações é 301, e este valor diminui quando relaxamos $\varepsilon_{\cos}$ inicial.

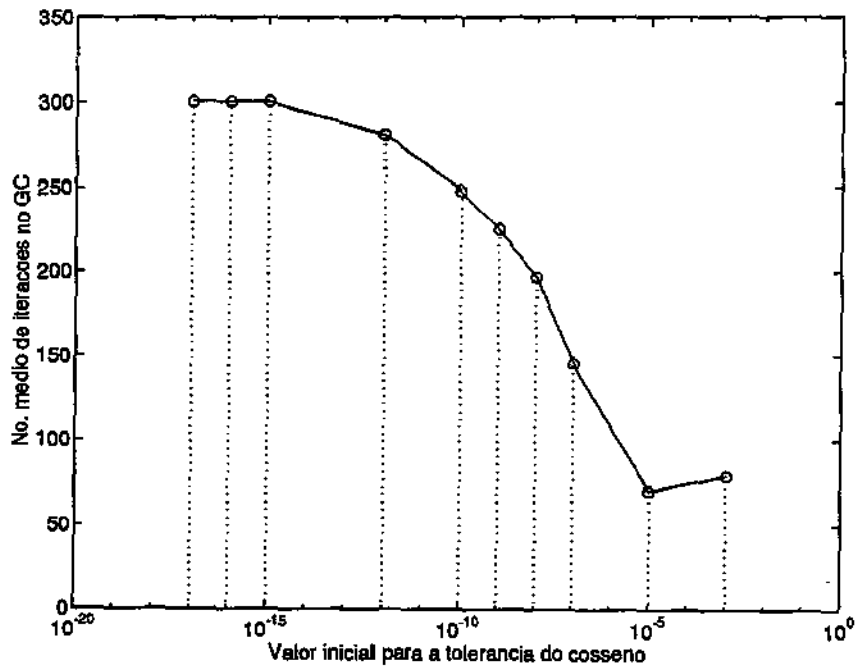


Figura 6.16: *problema* 200 x 400 x 10 : *número médio de iterações do GC*

Finalmente, a Figura 6.17 mostra o número de iterações efetuadas pelo método primal-dual para cada valor inicial de $\varepsilon_{\cos}$. O valor 23 permanece para vários valores iniciais de $\varepsilon_{\cos}$, chegando a 50 para $\varepsilon_{\cos} = 10^{-3}$.

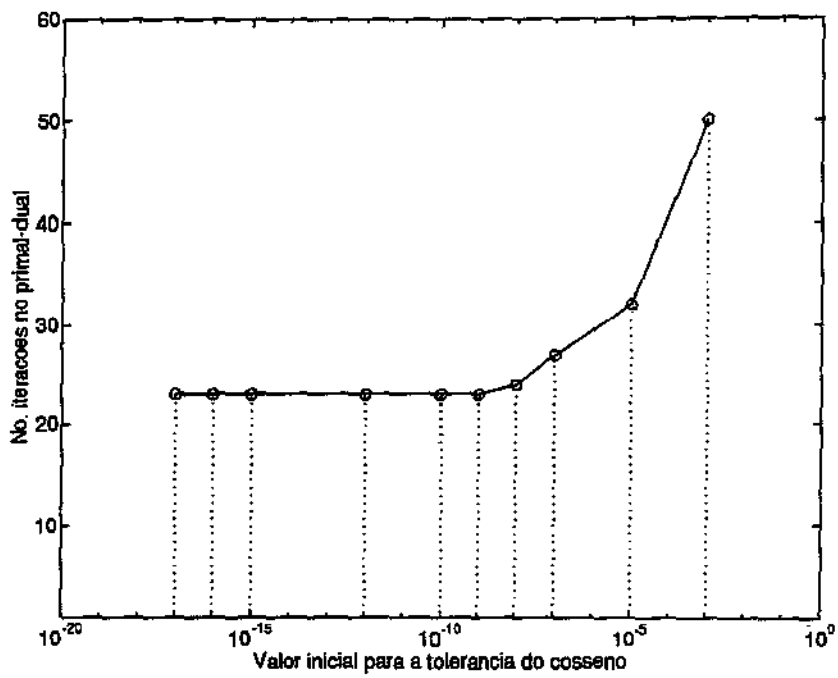


Figura 6.17: *problema* 200 x 400 x 10 : *número de iterações do primal-dual*

Até aqui, todas as comparações com o critério de parada pelo *cosseno* foram feitas tomando por base o critério do *resíduo absoluto*. Podemos ainda fazer comparações tomando o critério do *resíduo relativo*. De fato, se compararmos os valores das três subcolunas de "$\varepsilon_{\cos} = 10^{-7}$" da Tabela 6.9 com os valores das três subcolunas de "resíduo relativo" da Tabela 6.7, vemos que eles são muito parecidos.

No gráfico dado pela Figura 6.18, estamos colocando o tempo de cpu destes dois casos. Isto nos leva a concluir que, colocando o critério do *resíduo relativo*, o critério do *cosseno* não é tão necessário.

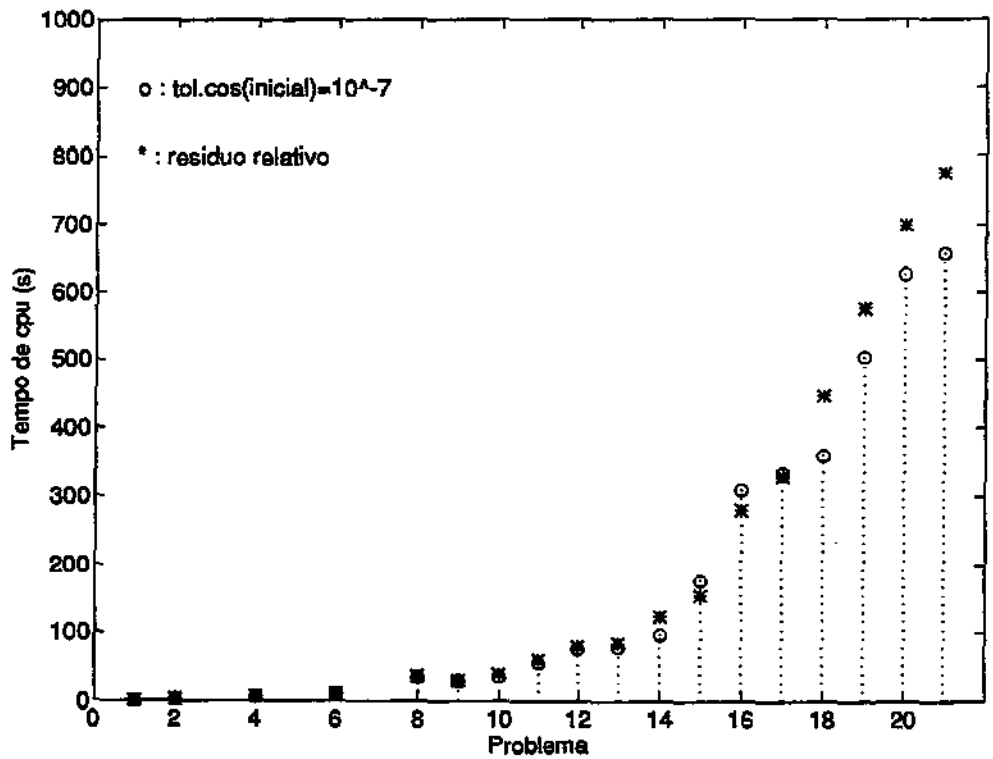


Figura 6.18: *resíduo relativo* $\times$ ε_{cos} *inicial* 10^{-7}

Ainda assim, podemos considerar os dois critérios, e o método dos gradientes conjugados termina quando o primeiro deles for alcançado.

Nas próximas tabelas, mostramos os resultados de quatro problemas para quatro combinações do critério de parada do gradiente conjugado. Estas combinações estão descritas na Tabela 6.10:

versão	resíduo	cosseno
I	absoluto	variável
II	relativo	variável
III	absoluto	fixo
IV	relativo	fixo

Tabela 6.10: *parada no gradiente conjugado*

As colunas $\varepsilon_{cos} = 10^{-5}$ e $\varepsilon_{cos} = 10^{-7}$ indicam, nas versões I e II, o valor inicial de ε_{cos} e, nas versões III e IV, o valor fixo para ε_{cos} .

	$\varepsilon_{cos} = 10^{-5}$			$\varepsilon_{cos} = 10^{-7}$		
versão	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
I	218.3	33	139	263.7	25	230
II	224.9	33	139	269.9	25	230
III	178.6	32	104	238.2	25	202
IV	165.6	32	104	235.7	25	202

Tabela 6.11: *problema* $300 \times 400 \times 10$

Nas quatro tabelas, podemos notar uma grande semelhança entre as versões I e II e também entre as versões III e IV (de fato, os valores das colunas *it.pd* e *mit.gc* são idênticos para I e II e depois para III e IV).

As versões III e IV geralmente apresentam um menor número de iterações do gradiente conjugado e, consequentemente, um menor tempo de cpu. No entanto, os valores de $r'r$ ao final do gradiente conjugado são sempre piores para estas versões.

versão	$\varepsilon_{cos} = 10^{-5}$			$\varepsilon_{cos} = 10^{-7}$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
I	56.2	38	39	41.6	23	51
II	55.9	38	39	41.6	23	51
III	27.4	27	28	50.9	25	50
IV	27.8	27	28	47.8	25	50

Tabela 6.12: *problema* $100 \times 200 \times 20$

versão	$\varepsilon_{cos} = 10^{-5}$			$\varepsilon_{cos} = 10^{-7}$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
I	894.8	36	277	413.9	27	171
II	908.9	36	277	411.4	27	171
III	422.7	50	89	327.6	25	146
IV	417.2	50	89	329.9	25	146

Tabela 6.13: *problema* $300 \times 400 \times 20$

versão	$\varepsilon_{cos} = 10^{-5}$			$\varepsilon_{cos} = 10^{-7}$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
I	576.9	49	118	492.9	29	173
II	578.2	49	118	494.7	29	173
III	542.8	48	114	395.3	30	134
IV	547.2	48	114	397.2	30	134

Tabela 6.14: *problema* $300 \times 500 \times 20$

Para $\varepsilon_{cos} = 10^{-5}$, as quatro tabelas apresentam valores grandes para $r'r$: da ordem de 10^{-1} e 10^{-2} para as versões I e II e da ordem de 10^{-1} e até valores maiores que 1 para as versões III e IV. De fato, não houve convergência para o problema da Tabela 6.13, isto é, foi atingido o número máximo de iterações do primal-dual (50). O problema da Tabela 6.14 quase atingiu este limite. Isto mostra que um valor "grande" e fixo de ε_{cos} pode ser arriscado: por um lado, pode haver convergência com um tempo de cpu menor, como nas duas primeiras tabelas; por outro lado, pode não haver convergência. Neste sentido, o critério do cosseno variável é mais confiável (a menos que o valor fixo de ε_{cos} seja pequeno).

Para $\varepsilon_{cos} = 10^{-7}$, estes resíduos prevalecem da ordem de 10^{-3} para as versões I e II e da ordem de 10^{-2} para as versões III e IV. Notamos aqui que, para o critério do cosseno variável, não há necessidade de tomarmos como valor inicial $\varepsilon_{cos} = 10^{-7}$, pois, com isto, efetuamos um número desnecessário de iterações no gradiente conjugado nas primeiras iterações do método primal-dual, uma vez que esta tolerância será diminuída. A Tabela 6.13 mostra que é melhor começar com $\varepsilon_{cos} = 10^{-5}$ e depois diminuir este valor: há convergência para este problema em 36 iterações do método primal-dual, quando esta convergência não é alcançada com o critério do cosseno fixo.

Nestes exemplos (e em outros também), observando a ordem de $r'r$ ao final do gradiente conjugado, concluímos que a parada deste método acaba acontecendo pelo critério do cosseno. Assim, se queremos que o critério do resíduo prevaleça, podemos colocar um valor mais acurado para ε_{cos}. Por outro lado, se não queremos considerar o critério de parada pelo cosseno, achamos que o critério do resíduo relativo é melhor do que o absoluto, dada a ordem de grandeza do lado direito de nosso sistema.

Observamos também, dos vários experimentos efetuados, que o critério do cosseno variável é sempre melhor do que o critério do cosseno fixo. Por este motivo, este último acabou sendo desprezado.

Concluindo: se optamos pelo critério de parada do resíduo, é melhor o resíduo relativo; se optamos pelo critério do cosseno, é melhor o variável. Se queremos uma acuracidade maior, o critério do resíduo é melhor; se a acuracidade pode ser menor, é melhor usarmos o critério do cosseno. Mantendo esta ordem nas tolerâncias — $\varepsilon_{cos} = 10^{-7}$ (inicial) e $\varepsilon_{gc} = 10^{-8}$, temos geralmente um número médio de iterações do gradiente conjugado maior se paramos com o critério do resíduo (e, por consequência, um maior tempo de cpu), mas as

direções obtidas são melhores. Cabe dizer ainda que, futuramente, faremos outros testes aumentando a redução no valor de 0.95 no critério do cosseno variável.

6.4.5 Aproveitamento da direção anterior

Outro conjunto de testes realizados se referem ao aproveitamento da última direção obtida pelo método dos gradientes conjugados para inicializar a próxima iteração. Em outras palavras, se na iteração ℓ do método primal-dual a direção obtida pelo gradiente conjugado é $(\Delta\widehat{y})^{\ell}$, na iteração $\ell+1$ do primal-dual o método dos gradientes conjugados inicializa o Algoritmo 4.3 com $(\Delta\widehat{y})^{\ell}$.

Depois de vários experimentos concluímos que este procedimento não é vantajoso, quando comparado com a inicialização usual que é a direção nula. De fato, inicializando o método dos gradientes conjugados com qualquer outra direção diferente da direção nula, é necessário efetuar um produto matriz-vetor $(\widehat{Q}\ \Delta\widehat{y})$, que é o maior esforço computacional de cada iteração do método dos gradientes conjugados. Além disso, nas primeiras iterações do método primal-dual, as direções geradas pelo gradiente conjugado são bastante distintas entre si e, nas últimas iterações, estas direções tendem à direção nula. Isto justifica a inicialização usual do gradiente conjugado, ou seja, a direção nula.

Tentamos obter algum aumento na eficiência utilizando o critério de parada do cosseno e o aproveitamento da direção anterior a partir de uma certa iteração do método primal-dual, e não necessariamente a partir da primeira. Por exemplo, um critério que poderia ser considerado é a utilização da direção anterior quando fazemos a troca de precondicionador.

O parâmetro *it.d* foi criado para indicar a partir de qual iteração do método primal-dual vamos inicializar o método dos gradientes conjugados com a direção anterior. Este parâmetro é inteiro, com $0 \leq it.d \leq it.max$. Se $it.d = \ell$, o gradiente conjugado começa da direção nula nas primeiras iterações do primal-dual e utiliza a direção anterior a partir da iteração ℓ do primal-dual. Se $it.d = 0$, o gradiente conjugado começa da direção nula na primeira iteração do primal-dual e utiliza a direção anterior em todas as outras iterações. Se $it.d > it.max$, a direção anterior nunca é utilizada, e o gradiente conjugado sempre começa da direção nula.

Os experimentos que estão resumidos na Tabela 6.15 mostram alguns resultados quando aproveitamos a direção anterior. Nestes experimentos consideramos o primeiro ponto inicial, $it.dg = 1$, $it.fg = 7$, $\varepsilon_{\cos} = 0.95\ \varepsilon_{\cos}$, com $\varepsilon_{\cos}$ inicial 10^{-7} , critério do resíduo absoluto no gradiente conjugado e os seguintes valores para $it.d$: 0, 7, 12 . Note que $it.d = 7$ coincide com $it.fg = 7$, isto é, utilizamos a direção anterior quando o precondicionador *floresta geradora* é introduzido. O fato é que não foi observada nenhuma diferença significativa entre vários valores de $it.d$, nem tampouco algum ganho pelo fato de aproveitar a direção anterior ao invés de iniciar o gradiente conjugado sempre com a direção nula. Podemos observar que as três colunas da Tabela 6.15 praticamente se repetem. E ainda, observando a coluna $\varepsilon_{\cos} = 10^{-7}$ da Tabela 6.9, onde sempre iniciamos o método dos gradientes conjugados com a direção nula, constatamos que esta coluna apresenta melhores resultados do que qualquer uma das três colunas da Tabela 6.15: em outras palavras, seja no tempo de cpu, ou no número de iterações do primal-dual, ou no número médio de iterações do gradiente conjugado, parece melhor iniciar o gradiente conjugado da direção nula. Vários outros experimentos foram efetuados neste sentido, inclusive modificando o valor inicial de $\varepsilon_{\cos}$, mas sempre chegamos às mesmas conclusões.

Outra tentativa para melhorar este resultado foi tomar a direção obtida ao final do gradiente conjugado multiplicada por um fator de redução ao se iniciar a próxima execução do gradiente conjugado, isto é, se na iteração ℓ do método primal-dual a direção obtida pelo gradiente conjugado é $(\Delta\widehat{y})^{\ell}$, na iteração $\ell + 1$ do primal-dual o método dos gradientes conjugados inicializa o Algoritmo 4.3 com $\Lambda(\Delta\widehat{y})^{\ell}$, onde $\Lambda = \mu^{\ell+1}/\mu^{\ell}$, com $\mu^{\ell} = \eta^{\ell}\gamma^{\ell}\ /\ \widehat{n}$ (ver Capítulo 3). Os resultados anteriores se mantiveram.

A Tabela 6.16 mostra os resultados obtidos com esta redução para um dos cinco problemas de números 9 (100 x 200 x 10), 16 (300 x 400 x 10) e 19 (400 x 600 x 10), respectivamente. Na primeira coluna, iniciamos o gradiente conjugado sempre com a direção anterior ($it.d = 0$) e, na última, sempre utilizamos a direção nula $it.d = 51$. Tomamos ainda, como anteriormente, os valores intermediários $it.d = 7$ e $it.d = 12$. Os resultados da última coluna ($it.d = 51$) ainda são um pouco melhores do que das outras, só perdendo de uma ou duas iterações no total de iterações do método primal-dual. Por exemplo, o número médio de iterações do gradiente conjugado para os três problemas, para $it.d = 0,\ 12,\ 51$ pode ser observado pelo gráfico dado pela

problema	$it.d = 0$			$it.d = 7$			$it.d = 12$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
1	0.8	16	27	0.8	16	27	0.8	16	27
2	2.2	17	37	2.3	17	37	2.2	17	36
4	6.6	18	63	6.7	18	64	6.5	18	64
6	11.5	19	68	11.6	19	68	11.4	19	68
8	41.4	21	133	40.3	20	133	41.7	20	135
9	34.0	21	93	33.8	21	94	35.4	21	96
10	42.4	22	97	42.1	22	98	41.9	22	98
11	69.0	21	148	72.5	21	154	72.2	21	156
12	89.0	22	163	89.4	22	164	91.4	22	167
13	107.4	23	173	102.4	23	169	102.4	23	171
14	125.0	23	175	129.2	23	178	127.1	23	182
15	190.6	24	222	170.1	23	205	180.4	25	172
16	332.9	24	305	344.9	23	313	335.6	23	315
17	353.7	24	284	351.7	24	292	347.0	24	294
18	502.5	26	363	516.9	26	366	503.1	26	361
19	663.5	25	416	749.8	25	430	694.8	25	438
20	906.8	26	481	894.1	26	479	913.3	26	488
21	971.1	28	470	1124.7	30	503	1054.0	29	498

Tabela 6.15: *primeira utilização de direções anteriores*

Figura 6.19. Já aqui podemos observar que, à medida em que o tamanho do problema cresce, a diferença a favor da direção nula se acentua.

$it.d = 0$			$it.d = 7$			$it.d = 12$			$it.d = 51$		
cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
34.1	21	92	35.0	21	93	34.7	21	95	33.3	22	84
297.6	23	286	301.8	23	286	330.0	23	305	273.1	25	230
558.9	25	354	573.7	25	363	624.0	25	396	523.6	27	288

Tabela 6.16: *exemplos com redução das direções anteriores: problemas* 9, 16 e 19

Outros experimentos foram efetuados mantendo $it.dg = 1$ e $it.fg = 7$, mas considerando agora o critério do resíduo relativo como parada no método do gradiente conjugado e desprezando o critério de parada pelo cosseno. Os resultados não se modificaram muito, como pode ser observado na Tabela 6.17 (aqui, começamos a partir do problema 8).

Comparando as colunas $it.d = 0$ e $it.d = 7$ com a coluna "resíduo relativo" da Tabela 6.7, observamos não aparecer nenhum ganho no tempo de cpu e no número médio de iterações do gradiente conjugado; apenas reduzimos uma ou duas iterações do primal-dual em alguns problemas.

Se estes resultados são comparados com os da Tabela 6.15, podemos observar uma pequena melhora no tempo de cpu, enquanto que o número médio de iterações do gradiente conjugado fica um pouco pior e o número de iterações do primal-dual não se altera.

Concluindo, independentemente do critério de parada do gradiente conjugado — se pelo cosseno variável ou pelo resíduo relativo — não observamos vantagens na utilização das direções anteriores.

Utilizamos novamente o mesmo fator de redução da direção, Λ, descrito anteriormente (agora com o critério de parada do resíduo relativo no gradiente conjugado) e, mais uma vez, os resultados se mantiveram. Mostramos os resultados dos mesmos problemas 9, 16 e 19, para $it.d = 0,\ 7,\ 12,\ 51$, na Tabela 6.18. Os valores obtidos para $it.d = 51$ continuam um pouco melhores do que para $it.d = 0,\ 7,\ 12$, mas, se comparamos estes resultados com os

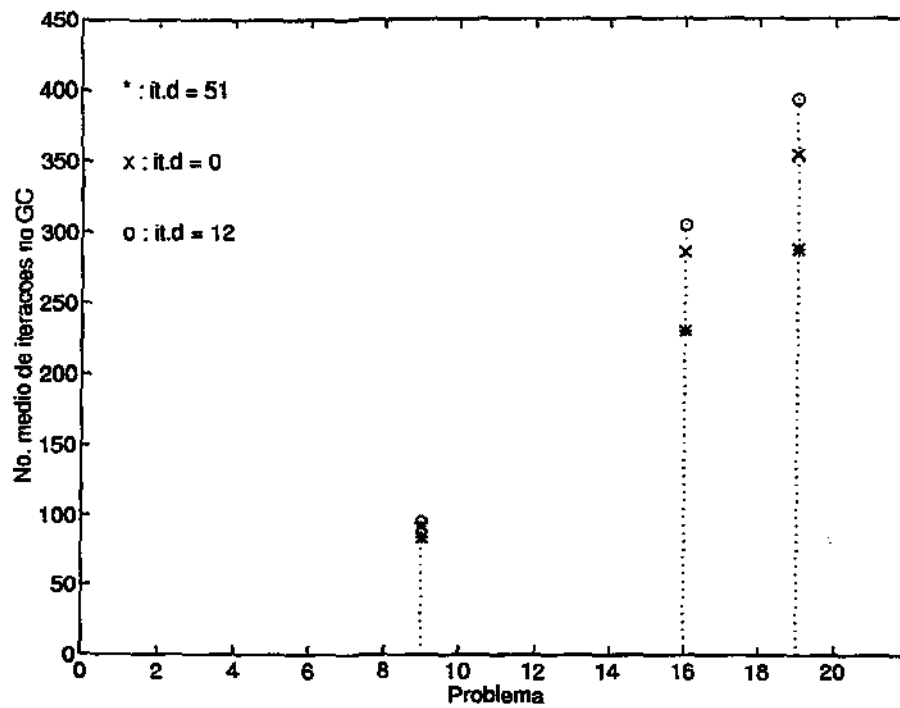


Figura 6.19: *problemas* 9, 16 *e* 19 : *número médio de iterações no gradiente conjugado variando it.d*

problema	$it.d = 0$			$it.d = 7$		
	cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*
8	39.6	20	153	33.9	20	129
9	33.4	21	108	32.3	21	103
10	40.5	22	109	39.8	22	105
11	68.2	21	176	69.4	21	180
12	88.7	22	189	81.8	22	178
13	106.4	23	201	103.7	23	200
14	122.9	22	207	127.1	23	210
15	192.7	24	265	193.1	24	264
16	322.3	23	349	321.1	23	358
17	342.7	24	329	341.9	24	332
18	475.1	25	407	481.3	25	412
19	679.7	25	493	683.2	25	492
20	863.8	26	545	903.9	26	565
21	1021.4	28	573	1018.6	27	584

Tabela 6.17: *segunda utilização de direções anteriores: parada do gradiente conjugado com resíduo relativo*

da Tabela 6.16, vemos que eles pioraram no número médio de iterações do gradiente conjugado e, como consequência, no tempo de cpu.

O gráfico da Figura 6.20 mostra o tempo de cpu (valores retirados das Tabelas 6.16 e 6.18) no problema 19, para os quatro valores de *it.d*: 0, 7, 12 e 51. Neste gráfico, podemos observar melhor o aumento do tempo de cpu quando o critério de parada do gradiente conjugado é o do resíduo relativo, ao invés do cosseno variável.

$it.d = 0$			$it.d = 7$			$it.d = 12$			$it.d = 51$		
cpu	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.gc*	*cpu*	*it.pd*	*mit.g*
35.9	21	116	36.7	21	116	36.9	21	118	34.9	21	117
295.7	23	332	294.6	23	334	305.4	23	343	290.1	23	328
695.0	25	508	641.7	24	506	639.1	24	488	628.3	25	476

Tabela 6.18: *exemplos com redução das direções anteriores, utilizando o critério do resíduo relativo no gradiente conjugado: problemas 9, 16 e 19*

Devido a estes e outros experimentos com resultados semelhantes, optamos por utilizar a direção nula sempre que iniciamos o método dos gradientes conjugados. No entanto, a inicialização do gradiente conjugado poderia ser ainda feita com o lado direito do sistema linear, ao invés da direção nula, o que resulta na economia de uma iteração deste método.

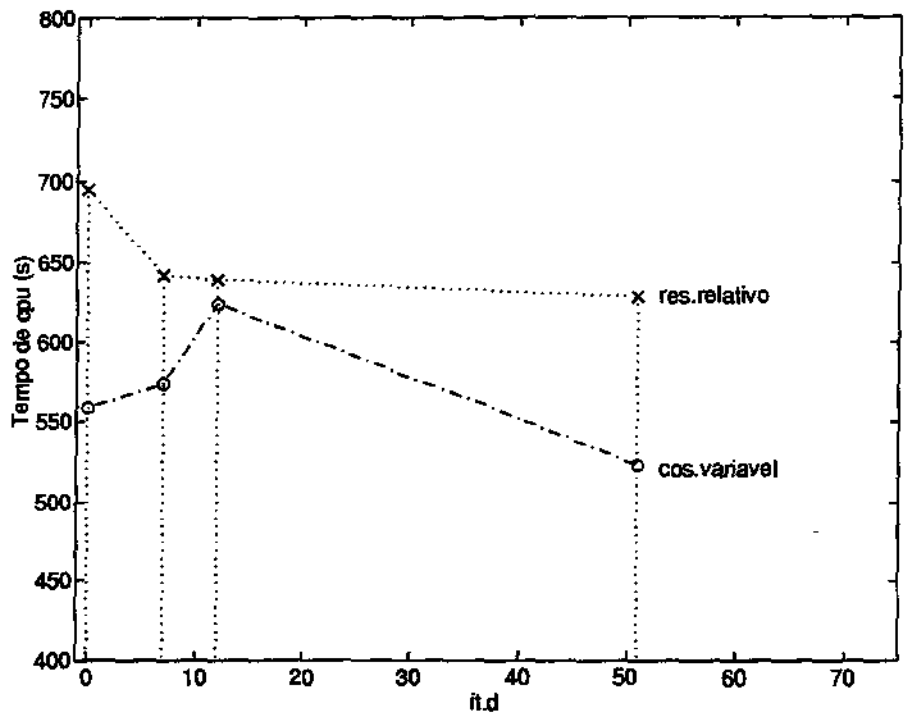


Figura 6.20: *problema* 19 : $400 \times 600 \times 10$

6.4.6 Informações das matrizes de scaling

Outros experimentos foram realizados com o intuito de observar o comportamento das matrizes de *scaling* Θ^k, $k = 1, \cdots, p+1$. Como já foi discutido anteriormente, os elementos destas matrizes tendem a zero ou infinito nas últimas iterações do método primal-dual. Esta informação pode ser utilizada, por exemplo, para decidirmos em qual iteração do primal-dual é conveniente introduzir o precondicionador *floresta geradora máxima*: se temos uma quantidade suficiente $(m-1)$ de elementos em Θ^k que são "grandes", em princípio convém introduzir a *floresta geradora máxima*. Se esta quantidade for ainda pequena, podemos continuar com o precondicionador *diagonal*.

Como exemplo, vamos tomar um problema com 2 produtos, 4 nós e 6 arcos, cuja rede é dada pela Figura 6.21. Os arcos são dados por $\{e_1, \cdots, e_6\}$ (já tomamos esta rede como exemplo anteriormente).

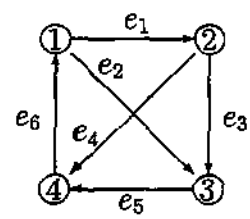


Figura 6.21: *rede exemplo para as matrizes de scaling*

Os demais dados são:

$$\text{produto 1:} \begin{cases} b^1 = (\ 1,\ \ 2,\ \ -4,\ \ 1\ )' \\ u^1 = (\ 7,\ \ 7,\ \ 7,\ \ 7,\ \ 7,\ \ 7\ )' \\ c^1 = (\ 2,\ \ 4,\ \ 1,\ \ 6,\ \ 4,\ \ 1\ )' \end{cases}$$

$$\text{produto 2:} \begin{cases} b^2 = (\ 5,\ \ 1,\ \ -1,\ \ -5\ )' \\ u^2 = (\ 6,\ \ 6,\ \ 6,\ \ 6,\ \ 6,\ \ 6\ )' \\ c^2 = (\ 7,\ \ 4,\ \ 5,\ \ 1,\ \ 3,\ \ 7\ )' \end{cases}$$

acoplamento: $d = (\ 9,\ \ 6,\ \ 9,\ \ 3,\ \ 7,\ \ 7)'$

Este problema é resolvido em 13 iterações do método primal-dual, independentemente se o precondicionador *floresta geradora máxima* é introduzido mais cedo ou mais tarde. Colocando este precondicionador desde a primeira iteração do primal-dual, podemos observar os resultados que estão colocados na Tabela 6.19. Nesta tabela, a coluna n_1 contém o número de elementos da matriz de *scaling* de cada produto ou do acoplamento que são menores do que 1; a coluna n_2 contém o número de elementos que são maiores do que 1. Como exemplo, na iteração 7 do primal-dual, os elementos que foram computados em n_1 são da ordem de 10^{-4} e, em n_2, da ordem de 10^3. Na iteração 10, estes valores já mudam para 10^{-6} e 10^5. Na última iteração, são da ordem de 10^{-8} e 10^7. Neste exemplo, os arcos selecionados para as árvores geradoras se repetem em todas as iterações do primal-dual: e_6, e_1, e_3 para o primeiro produto e e_2, e_4, e_5 para o segundo produto. Isto significa que não era necessário construir o precondicionador *floresta geradora máxima* em cada iteração do primal-dual: ele poderia ter sido construído uma única vez para cada produto, e seria usado em todas as outras iterações. Naturalmente, este é um exemplo muito pequeno. No entanto, em problemas maiores, as florestas geradoras tendem a se repetir a partir de uma determinada iteração do primal-dual, ou, pelo menos, boa parte de seus arcos coincidem nas últimas iterações do primal-dual. Esta informação pode ser aproveitada — assim evitamos construir as florestas desnecessariamente.

A informação dada pelas matrizes de *scaling* também pode ser utilizada na construção de uma **solução básica ótima**, principalmente se o problema não for degenerado.

No exemplo anterior, a dimensão da base é 12 e, neste caso, a base ótima é formada pelas árvores dos dois produtos e mais 6 variáveis de folga do acoplamento (de fato, este problema é desacoplável).

Num outro exemplo com 2 produtos, 7 nós e 12 arcos temos agora, na última iteração do método primal-dual:

produto 1 : $n_1 = 5$, $n_2 = 7$
produto 2 : $n_1 = 3$, $n_2 = 9$
acoplamento : $n_1 = 1$, $n_2 = 11$

Neste caso, como a dimensão da base é 24 (e não 27), temos uma **degenerescência dual** e, apenas pela ordem de grandeza dos elementos de Θ^k não conseguimos determinar uma solução básica ótima.

	produto 1		produto 2		acoplamento	
it.pd	n_1	n_2	n_1	n_2	n_1	n_2
1	6	0	6	0	1	5
2	5	1	3	3	1	5
3	3	3	3	3	1	5
4	3	3	2	4	0	6
5	3	3	3	3	0	6
6	3	3	3	3	0	6
7	3	3	3	3	0	6
8	3	3	3	3	0	6
9	3	3	3	3	0	6
10	3	3	3	3	0	6
11	3	3	3	3	0	6
12	3	3	3	3	0	6
13	3	3	3	3	0	6

Tabela 6.19: *quantidade de elementos maiores e menores que 1 em* Θ^k

Outro exemplo, agora com **degenerescência primal**: 3 produtos, 10 nós e 20 arcos, que resulta na última iteração do primal-dual:

produto 1 : $n_1 = 11$, $n_2 = 9$
produto 2 : $n_1 = 11$, $n_2 = 9$
produto 3 : $n_1 = 11$, $n_2 = 9$
acoplamento : $n_1 = 1$, $n_2 = 19$

Neste caso, falta informação sobre uma variável básica, já que a dimensão da base é 47.

Outros resultados podem ser observados (o precondicionador *floresta geradora máxima* foi introduzido na iteração 7 do primal-dual):

• Em um problema com $m = 50$, $n = 82$, $p = 5$ (Tabela 6.20): o total de iterações do primal-dual é 16 e a dimensão da base é 327.

Os produtos 1 e 3 devem possuir dois arcos em ciclo com suas respectivas árvores geradoras, ao passo que os produtos 2 e 5 possuem apenas um arco em ciclo. O produto 4 não apresenta ciclos. Dos 82 arcos, 76 estão "folgados". Este problema é **não degenerado**.

produto	$it.pd = 7$		$it.pd = 16$	
	n_1	n_2	n_1	n_2
1	40	42	31	51
2	43	39	32	50
3	38	44	31	51
4	40	42	33	49
5	38	44	32	50
acopl.	6	76	6	76

Tabela 6.20: *problema* $50 \times 82 \times 5$

• Outro problema, com $m = 200$, $n = 318$, $p = 3$ (Tabela 6.21): o total de iterações do primal-dual é 22 e a dimensão da base é 915. Neste caso, temos uma **degenerescência primal**: são 914 elementos "grandes" nas matrizes de *scaling* (entre 10^2 e 10^9).

produto	$it.pd = 7$		$it.pd = 15$		$it.pd = 22$	
	n_1	n_2	n_1	n_2	n_1	n_2
1	142	176	106	212	105	213
2	151	167	115	203	114	204
3	136	182	110	208	109	209
acopl.	37	281	31	287	30	288

Tabela 6.21: *problema* $200 \times 318 \times 3$

• Outro problema, com $m = 300$, $n = 513$, $p = 3$ (Tabela 6.22): o total de iterações do primal-dual é 22 e a dimensão da base é 1410. Aqui também temos uma **degenerescência primal**: são 1408 elementos "grandes" nas matrizes de *scaling* (entre 10^3 e 10^8).

	$it.pd = 7$		$it.pd = 15$		$it.pd = 22$	
produto	n_1	n_2	n_1	n_2	n_1	n_2
1	242	271	200	313	200	313
2	246	267	201	312	200	313
3	228	285	203	310	201	312
acopl.	46	467	44	469	43	470

Tabela 6.22: *problema* $300 \times 513 \times 3$

• Por último, um problema com $m = 400$, $n = 627$, $p = 3$ (Tabela 6.23): o total de iterações do primal-dual é 24 e a dimensão da base é 1824. Ainda temos uma **degenerescência primal**.

	$it.pd = 7$		$it.pd = 15$		$it.pd = 24$	
produto	n_1	n_2	n_1	n_2	n_1	n_2
1	273	354	202	425	200	427
2	300	327	213	414	207	420
3	295	332	221	406	212	415
acopl.	77	550	67	560	66	561

Tabela 6.23: *problema* $400 \times 627 \times 3$

Note que, em todos estes exemplos, os resultados obtidos na iteração 7 do primal-dual, nas colunas n_1 e n_2, são muitos parecidos com os obtidos na última iteração. Isto nos leva a crer que as florestas geradoras poderiam ter sido repetidas algumas vezes, ao invés de serem construídas em toda iteração: certamente, a maioria dos arcos se repetiria. Observamos também que, na iteração 7, quando o precondicionador *floresta geradora máxima* foi introduzido, o número de elementos da coluna n_2 ainda era insuficiente para se formar árvores geradoras. Neste caso, foram utilizados arcos com pesos pequenos para que estas árvores (ou florestas) fossem completadas.

6.4.7 Heurística para solução básica ótima

Vários outros experimentos foram efetuados para testar a heurística apresentada no Capítulo 5 para a obtenção de uma solução básica ótima (Algoritmo 5.1).

Para problemas não degenerados, esta heurística se mostrou bastante robusta: de todos os testes efetuados, apenas em dois casos tivemos falha. No entanto, para problemas degenerados, ela quase nunca obteve sucesso.

De fato, esta heurística falhou em todos os problemas que apresentaram degeneração primal (inclusive os apresentados na seção anterior), e em vários casos de degeneração dual. Aliás, foi neste ponto que pudemos observar que nosso gerador (na versão de dados reais) apresenta com facilidade problemas com degeneração primal e, na versão de dados inteiros, apresenta com facilidade problemas com degeneração dual. E quanto maior é a dimensão do problema, maior é a chance de gerar problemas degenerados.

Na Tabela 6.24, apresentamos alguns resultados retirados dos vários testes realizados com a heurística. Nestes problemas, o valor de n_a (número de arcos verdadeiramente acoplados) é pequeno. Note que a coluna $\widehat{m}$ — dimensão do sistema linear das direções — também é a dimensão da base. A coluna "degenerado" indica se o problema tem degeneração primal ou dual ou se o problema não é degenerado. A coluna "sucesso" indica se a heurística obteve ou não sucesso.

Consideramos "sucesso" quando as quatro condições seguintes são satisfeitas:

(i) a solução básica primal encontrada é factível;
(ii) a solução básica dual encontrada é factível;
(iii) o valor das funções objetivo primal e dual são iguais, a menos de uma pequena tolerância;
(iv) este valor é igual (a menos de uma tolerância) ao valor encontrado para a função objetivo na última iteração do método primal-dual de pontos interiores.

A heurística fracassa quando:

(i) o vetor α não ficou completo;
(ii) a matriz ρ não se tornou vazia;
(iii) qualquer uma das condições de " sucesso" anteriores não é satisfeita.

As duas primeiras condições, que costumam acontecer simultaneamente, são

as causas mais comuns do "fracasso".

Convém observar que, escolhido e fixado um tamanho de problema, sempre foi possível gerar um problema não degenerado e outro com degeneração primal, apenas experimentando valores para o parâmetro *semente*. O que aconteceu em quase todas as vezes foi que a heurística obteve sucesso para o não degenerado e falha para o degenerado, independentemente do tamanho deste problema.

m	n	p	$\widehat{m}$	n_a	degenerado	sucesso
10	21	10	111	0	não	sim
20	40	10	230	2	não	sim
50	80	10	570	6	não	sim
50	100	10	590	4	não	não
50	102	10	592	7	não	sim
80	104	10	894	6	não	sim
100	200	5	695	29	não	sim
100	203	5	698	8	primal	não
100	204	5	699	44	dual	não
150	256	5	1001	27	primal	não
150	305	5	1050	28	primal	não
180	402	3	939	14	dual	não
180	402	3	939	43	não	sim
200	408	10	2398	15	dual	sim
200	404	10	2394	25	primal	não
300	425	5	1920	29	primal	não
300	420	5	1915	24	não	sim
300	512	3	1409	32	dual	não
300	607	5	2102	38	dual	sim
300	607	5	2102	40	primal	não
300	608	5	2103	39	não	sim

Tabela 6.24: *heurística para solução básica ótima*

6.5 Comparação com outro programa

Comparamos o programa **mult** com o programa **Lipsol** (Linear-programming point solvers), versão 0.3, de Yin Zhang, que utiliza o método primal-dual de pontos interiores [44, 51, 75]. O **Lipsol** é um programa que resolve problemas gerais de programação linear, relativamente grandes. Ele é baseado na versão 4.0 do MATLAB, usando suas rotinas de matrizes esparsas, possuindo códigos nas linguagens Fortran e C. É um programa "free", que pode ser obtido nos endereços: http://math.umbc.edu/~yzhang/lipsol/ ou ftp.math.umbc.edu:pub/zhang/lipsol/v03/ . Para a entrada de dados, além dos formatos MPS e LPP, Lipsol admite também o MAT-format, do MATLAB (transforma os formatos LPP e MPS em MAT, para resolver os problemas). Este programa possui ainda rotinas de preprocessamento.

Na Tabela 6.25, podemos observar alguns resultados desta comparação, quanto ao tempo de cpu e o número de iterações. Em **mult**, consideramos os seguintes valores para os parâmetros: $it.dg = 1$, $it.fg = 6$, $\varepsilon_{cos} = 10^{-7}$ (inicial) e $\varepsilon_{\mu} = 10^{-7}$. Utilizamos ainda o ponto inicial usual (primeiro ponto inicial) e o critério do resíduo relativo na parada do gradiente conjugado.

Lembramos que as colunas $\widehat{m}$ e $\widehat{n}$ dão as dimensões de cada problema, sendo que em $\widehat{m}$ já subtraímos as p equações linearmente dependentes (uma de cada produto). As colunas m_I e n_I dão, respectivamente, o número de linhas e de colunas com que o **Lipsol** efetivamente trabalha, depois que é feito o preprocessamento. Podemos notar que este programa obtém uma boa redução nas dimensões dos problemas: nos cinco últimos, aproximadamente entre 19% e 23% do número de linhas, a menos do nono problema, que apresenta uma redução de 6.4%, aproximadamente. Notamos, inclusive, que o tempo de cpu deste problema é relativamente maior que os demais.

O **Lipsol** sempre utiliza um número menor de iterações que **mult**: descartando o primeiro problema, temos uma média de 18 iterações para o **Lipsol**, contra uma média de 26 iterações para o **mult**. Além disso, as factibilidades na solução final são da ordem de 10^{-12}, contra 10^{-5} de **mult**. No entanto, o tempo de cpu de **mult** é menor, a menos do segundo problema. A razão entre os tempos de cpu (cpu de **Lipsol** / cpu de **mult**) pode ser observada no gráfico dado pela Figura 6.22. Neste gráfico, os problemas da Tabela 6.25 foram numerados de 1 a 11. Apenas no segundo problema esta razão é menor que 1; no primeiro problema ela chega a 6.4 e, no nono, é de 4.5.

			Lipsol				mult			
m	n	p	m_L	n_L	cpu	$it.pd$	$\widehat{m}$	$\widehat{n}$	cpu	$it.pd$
10	20	3	47	77	1.73	10	47	80	0.27	14
100	204	10	1094	2134	38.4	16	1194	2244	45.60	23
200	405	10	2195	4245	168.48	18	2395	4455	113.67	26
300	512	10	3002	5122	222.47	17	3502	5632	205.77	23
300	606	10	3286	6346	430.59	17	3596	6666	202.32	25
440	809	10	4690	8390	1060.30	18	5199	8899	580.18	25
200	318	20	3458	5818	432.11	17	4298	6678	260.50	28
200	318	30	5028	8568	999.40	18	6288	9858	324.55	26
200	401	20	4101	8121	1059.90	18	4381	8421	238.37	26
300	418	20	4938	7298	678.64	18	6398	8778	485.40	25
300	418	30	7198	10738	1085.97	18	9388	12958	923.75	28

Tabela 6.25: *comparação com o Lipsol*

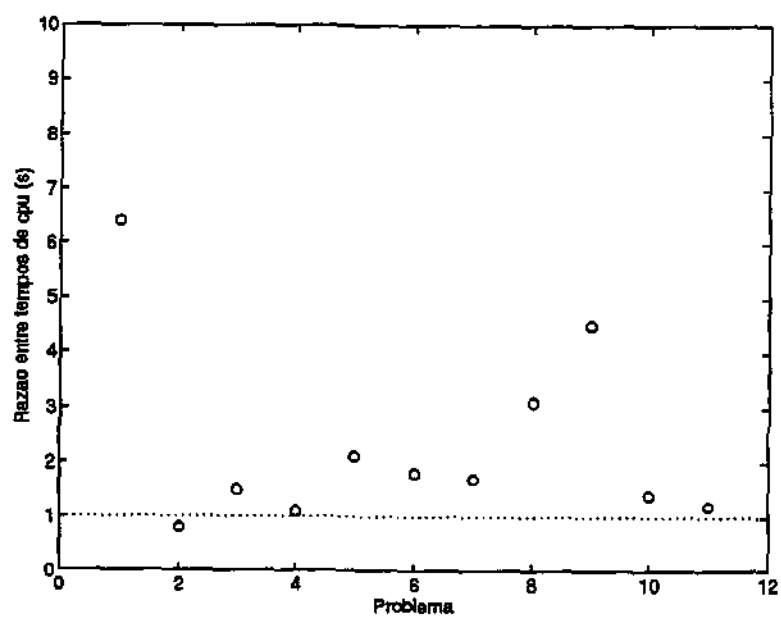


Figura 6.22: *razão entre os tempos de cpu de* **Lipsol e mult**

6.6 Outros problemas

Outros problemas foram retirados da Netlib. Eles podem ser obtidos na Internet, no site http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html. Estes problemas apresentam características bem particulares: as variáveis não são canalizadas, os custos são grandes ou negativos, nem todos os n arcos são acoplados, ou ainda, nem todos os arcos da rede são considerados para todos os produtos (ou seja, não necessariamente temos a mesma matriz A em todos os produtos). Muitos destes problemas pertencem a classes especiais, como foi descrito no Capítulo 1: problemas de *origem-destino* (cada produto vai de uma origem específica para um destino específico) ou problemas de *destino específico* (cada produto vai de uma origem específica para vários destinos ou vice-versa). Isto significa que vários elementos b_i^k são nulos.

Como o programa **mult** foi desenvolvido para problemas genéricos (como na descrição 5.1), e não para problemas de classes específicas, todas estas particularidades foram "camufladas" no programa. De fato, certas adaptações deveriam ter sido feitas neste programa, para que seu código tirasse proveito destas situações. Foi necessário introduzir alguns artifícios do tipo considerar explicitamente arcos com custo infinito, acoplamento infinito, ou a ausência de arcos, para resolver eficientemente estes problemas com o programa **mult**, uma vez que não é conveniente utilizar artifícios do tipo "M grande".

Em todos estes problemas foi utilizado, no método dos gradientes conjugados, o critério de parada pelo resíduo absoluto combinado com o critério do cosseno variável.

6.6.1 Problemas Assad

Nesta classe de problemas, nenhuma variável é canalizada, mas todos os n arcos são acoplados. Todos os custos são positivos, e o lado direito das restrições de acoplamento também.

Os valores dos parâmetros utilizados foram $it.dg = 1$, $it.fg = 7$, $\varepsilon_{cos} = 10^{-7}$ (inicial), $\varepsilon_\mu = 10^{-7}$. Utilizamos o ponto inicial usual (primeiro ponto inicial). A seguir, temos os problemas e seus resultados.

Devido ao fato das variáveis não serem canalizadas, também colocamos como ponto inicial, para as variáveis primais, $(x_j^k)^0 = 1$, $\forall k$, $\forall j$ (incluindo o acoplamento). Os resultados foram satisfatórios, e muito parecidos aos obtidos anteriormente. Também alteramos ε_{cos} inicial para 10^{-10} — neste caso,

problema	m	n	p	$\widehat{m}$	cpu	$it.pd$	$mit.gc$
assad1.5k	47	98	3	236	5.9	16	113
assad1.6k	47	98	3	236	5.0	20	79
assad3.4k	85	204	6	708	54.7	34	145
assad3.7k	85	204	6	708	51.5	31	153

Tabela 6.26: *problemas Assad*

temos uma diminuição do número de iterações do método primal-dual e um aumento no tempo de cpu e no número de iterações do gradiente conjugado.

6.6.2 Problemas Chen

Também nesta classe de problemas, todas as variáveis são canalizadas. No entanto, nem todos os arcos são acoplados, e os que são, ficam completamente utilizados na solução ótima ($x^{p+1} = 0$) . Vários destes arcos apresentam valores nulos no lado direito do acoplamento ($d_j = 0$). Estes problemas apresentam ainda muitos arcos paralelos, vários arcos com custo negativo e valores grandes para a função objetivo na solução ótima. São problemas degenerados.

Devido a estas particularidades, uma melhor solução ótima é encontrada se as tolerâncias utilizadas são mais acuradas. Além disso, os três pontos iniciais apresentados no Capítulo 3 não foram satisfatórios.

Iniciamos a solução primal com $(x_j^k)^0 = 1$, $\forall j$, $\forall k$ (incluindo o acoplamento) e a solução dual com $(y_j^{p+1})^0 = -300$, $\forall j$. As variáveis $(y_j^k)^0$, $\forall j$, $\forall k$ permanecem nulas, como anteriormente, e, para as variáveis $(z_j^k)^0$, temos a seguinte inicialização :

$(z_j^k)^0 = c_j^k - (y_j^{p+1})^0$, se $c_j^k - (y_j^{p+1})^0 > 0$

$(z_j^k)^0 = 100$, caso contrário.

O critério de parada pelo cosseno não se apresentou muito eficiente para estes problemas: relaxando ε_{cos}, os resíduos ao final do método dos gradientes conjugados foram bem grandes.

Nos resultados apresentados a seguir, utilizamos: $it.dg = 1$, $it.fg = 7$, $\varepsilon_{cos} = 10^{-10}$ (inicial), $\varepsilon_\mu = 10^{-10}$. A coluna n_a da Tabela 6.27 indica o número de arcos acoplados dentre os n arcos do problema.

problema	m	n	n_a	p	$\widehat{m}$	cpu	$it.pd$	$mit.gc$
chen0	26	117	43	4	143	9.5	26	133
chen1	36	174	65	5	240	18.8	24	164
chen2	41	358	155	7	435	116.7	32	341
chen3	31	149	56	15	506	52.6	28	163
chen4	55	420	176	15	986	400.3	36	404
chen5	65	569	242	10	882	477.2	37	513
chen6	41	409	177	9	537	248.7	40	387

Tabela 6.27: *problemas Chen*

Acreditamos que alguns destes resultados possam ser melhorados se alterarmos o ponto inicial, ou a tolerância do cosseno, ou algum outro parâmetro. Individualmente, constatamos que cada problema se comporta melhor se um ou outro parâmetro é alterado. Acreditamos também que o fato destes problemas serem bastante degenerados (pelo menos na solução ótima), faz com que o precondicionador *árvore geradora máxima* não seja muito eficiente, isto é, não faz muita diferença se introduzimos este precondicionador um pouco mais cedo ou um pouco mais tarde (tentamos $it.fg = 5,\ 7,\ 10$). No entanto, se ele não é utilizado, os resultados pioram.

6.6.3 Problemas Farvolden

Nestes problemas, todas as variáveis são canalizadas e nem todos os arcos são acoplados. A quase totalidade de seus arcos apresenta custos nulos ou muito grandes. A matriz A não é idêntica para todos os produtos porque os arcos considerados para cada um dos produtos não são os mesmos. Os valores da função objetivo na solução ótima são bem grandes.

Utilizamos o mesmo ponto inicial dos problemas Chen, mas, para o acoplamento, utilizamos $\left(y_j^{p+1}\right)^0 = -1,\ \forall j$. Mantivemos também os valores dos parâmetros: $it.dg = 1$, $it.fg = 7$, $\varepsilon_{cos} = 10^{-10}$ (inicial), $\varepsilon_\mu = 10^{-10}$. No entanto, foi necessário fazer uma maior redução no parâmetro de centragem — o redutor utilizado foi $\eta = 10^{-3}$ ao invés de $\eta = 10^{-1}$, como vinha sendo usado anteriormente. Esta necessidade se deu porque os valores iniciais do parâmetro de centragem μ eram da ordem de 10^6 , e a convergência era muito lenta. Os resultados obtidos com estes problemas podem ser observados na Tabela 6.28.

problema	m	n	n_a	p	$\widehat{m}$	cpu	$it.pd$	$mit.gc$
10term.0	190	507	143	10	2033	402.9	32	342
10term	190	510	146	10	2036	519.0	32	442
10term.50	190	498	134	10	2024	519.8	33	422
10term.100	190	491	127	10	2017	671.8	33	578
15term.0	285	745	202	15	2017	2538.7	36	882
15term	285	796	253	15	4513	4252.6	39	1354

Tabela 6.28: *problemas Farvolden*

Novamente, não fez muita diferença se introduzimos o precondicionador *árvore geradora máxima* um pouco mais cedo ou um pouco mais tarde ($it.fg = 5,\ 7,\ 10,\ 11$), mas, $it.fg = 13$ já é ruim. Se este precondicionador não for utilizado, os resultados pioram bastante. Experimentamos também $(y_j^{p+1})^0 = -300,\ \forall j$, ao invés de $(y_j^{p+1})^0 = -1,\ \forall j$, mas os resultados foram muito parecidos. Novamente, as particularidades de cada problema podem ser observadas — por exemplo, o problema 10term.100 apresenta uma melhora em seus resultados se $\varepsilon_\mu = 10^{-8}$. Já os problemas 15term.0 e 15term apresentam resultados melhores se $\varepsilon_\mu = 10^{-9}$ e $it.fg = 10$.

Capítulo 7

Degenerescência

O objetivo deste estudo é caracterizar a degenerescência e a falta de pontos interiores no problema de multifluxo. Para isso, iniciaremos com esta caracterização num problema geral de programação linear, passando a um problema de fluxo de custo mínimo com um único produto e, finalmente, chegando ao problema de multifluxo.

7.1 Programação linear

Seja A uma matriz $m \times n$, b um vetor de dimensão m e c um vetor de dimensão n. Considere o par primal-dual:

$$min\{c'x \ : \ Ax = b, \ x \geq 0\} \tag{7.1}$$

$$max\{b'y \ : \ A'y + z = c, \ z \geq 0, \ y \text{ irrestrito}\} \tag{7.2}$$

onde y tem dimensão m e z tem dimensão n.

Vamos considerar que $posto(A) = m$. Dadas as soluções x e $(y, \ z)$ de (7.1) e (7.2), respectivamente, definimos os dois subconjuntos seguintes, de colunas e linhas, com cardinalidades:

$$r_x = |\{a^j \ : \ x_j > 0\}|$$

$$r_z = |\{a_i \ : \ y_i \neq 0\} \cup \{e_j \ : z_j > 0\}|$$

onde a^j é a coluna j de A, a_i é a linha i de A e $e_j = (0 \cdots 1 \cdots 0)'$, com n componentes, é o j-ésimo vetor canônico.

Uma *solução primal factível* x (satisfaz $Ax = b$ e $x \geq 0$) é chamada de *básica* (ou *extrema*) se o conjunto de colunas associado a r_x é linearmente independente (isto implica $r_x \leq m$). Esta solução é *degenerada* se $r_x < m$. O problema (7.1) é *primal degenerado* se existe uma solução básica primal factível degenerada.

Uma *solução dual factível* $(y,\ z)$ (satisfaz $z = c - A'y \geq 0$) é chamada de *básica* se o conjunto de vetores associado a r_z é linearmente independente (isto implica $r_z \leq n$). Esta solução é *degenerada* se $r_z < n$. O problema (7.1) é *dual degenerado* se existe uma solução básica dual factível degenerada.

Uma solução primal factível x é *interior* se $x > 0$. Uma solução dual factível (y, z) é *interior* se $z > 0$.

Se existe uma variável primal (dual) *trivial* x_k (z_k), isto é, $x_k = 0$ $(z_k = 0)$ para toda solução primal (dual) factível x $(y,\ z)$, então não existe solução factível primal (dual) interior.

Uma *solução homogênea* de $\{Ax = b,\ \ x \geq 0\}$ é uma solução factível de $\{Ax = 0,\ \ x \geq 0\}$. Uma *solução homogênea extrema* é uma solução básica factível de $\{Ax = 0,\ \ x \geq 0,\ \ \sum_j x_j = 1\}$.

Teorema da representação [7]:
Toda solução primal factível x de (7.1) pode ser expressa pela soma de uma combinação linear convexa de soluções básicas factíveis x_e^i com uma combinação linear não negativa de soluções homogêneas extremas x_h^j, isto é, $x = \sum_i \alpha_i\ x_e^i + \sum_j \beta_j\ x_h^j$, com $\alpha_i \geq 0,\ \forall i,\ \sum_i \alpha_i = 1,\ \beta_j \geq 0,\ \forall j$. Similarmente, toda solução dual factível y de (7.2) pode ser expressa pela soma de uma combinação linear convexa de soluções básicas factíveis y_e^i com uma combinação linear não negativa de soluções homogêneas extremas y_h^j, isto é, $y = \sum_i \alpha_i\ y_e^i + \sum_j \beta_j\ y_h^j$, com $\alpha_i \geq 0,\ \forall i,\ \sum_i \alpha_i = 1,\ \beta_j \geq 0,\ \forall j$.

Teorema fundamental da dualidade [7]:
Se um dos problemas (7.1) ou (7.2) tem uma solução ótima, então o outro também tem e ambas soluções possuem o mesmo valor da função objetivo: $c'x = b'y$.

Lema de Farkas [7]:
Exatamente um dos seguintes sistemas tem uma solução factível e o outro é

inconsistente:

$$\{Ax = b\ ,\ x \geq 0\}$$

$$\{A'y \geq 0\ ,\ b'y < 0\}$$

Assim, um problema é **primal infactível** *se e somente se existe um vetor y que satisfaz $A'y \geq 0$, com $b'y < 0$.*

Teorema de Gale [46]:
Exatamente um dos seguintes sistemas tem uma solução factível e o outro é inconsistente:

$$\{A'y \leq c\}$$

$$\{Ax = 0\ ,\ x \geq 0\ ,\ c'x = -1\}$$

Assim, um problema é dual infactível se e somente se existe um vetor x que satisfaz $Ax = 0$, $x \geq 0$, com $c'x < 0$.

Teorema (inexistência de solução primal factível interior):
Considere um problema primal factível. Existe uma variável primal trivial x_k se e somente se existe um vetor y que satisfaz $A'y \geq e_k$, com $b'y = x_k = 0$.

prova: aplicando o teorema fundamental da dualidade sobre o par primal-dual abaixo, com soluções ótimas x e y, onde $b'y = x_k = 0$, temos:

$$max\{x_k\ :\ Ax = b,\ x \geq 0\}$$

$$min\{b'y\ :\ A'y \geq e_k\}$$

Note que $b'y \geq 0$ para toda solução factível y, pelo teorema fraco da dualidade. $\diamondsuit$

Teorema (degenerescência primal):
Considere uma solução básica factível associada à uma base B de $A = [B \mid N]$. Existe uma variável primal factível básica degenerada x_{Bk} (isto é, a k-ésima variável básica) se e somente se existe um vetor y satisfazendo $B'y \geq e_{Bk}$, com $b'y = x_{Bk} = 0$.

prova: aplicando o teorema fundamental da dualidade sobre o par primal-dual abaixo, com soluções ótimas x e y, onde $b'y = x_{Bk} = 0$, temos:

$$max\{x_{Bk} \ : \ Bx_B = b, \ x \geq 0\}$$

$$min\{b'y \ : \ B'y \geq e_{Bk}\}$$

Note que $b'y \geq 0$ para todo vetor factível y . $\Diamond$

Teorema (inexistência de solução dual factível interior):
Considere um problema dual factível. Existe uma variável dual trivial z_k se e somente se existe um vetor x que satisfaz $Ax = 0$, $x \geq e_k$, com $c'x = z_k = 0$.

prova: novamente, da aplicação do teorema fundamental da dualidade sobre o par primal-dual abaixo, com soluções ótimas x e $(y, \ z)$ e com $c'x = z_k = 0$, temos:

$$min\{c'x \ : \ Ax = 0, \ x \geq e_k\}$$

$$max\{z_k \ : \ z = c - A'y \geq 0\}$$

Note que $c'x \geq 0$ para toda solução factível x . $\Diamond$

Teorema (degenerescência dual):
Considere uma solução básica factível associada à uma base B de $A = [B \mid N]$. Existe uma variável de folga dual factível não básica degenerada z_{Nk} (isto é, a k-ésima variável não básica) se e somente se existe um vetor x satisfazendo $Ax = 0$, $x_N \geq e_{Nk}$, com $c'x = z_{Nk} = 0$.

prova: novamente, da aplicação do teorema fundamental da dualidade sobre o par primal-dual abaixo, com soluções ótimas x e z, onde $c'x = z_{Nk} = 0$, temos:

$$min\{c'x \ : \ Ax = 0 \ , \ x_N \geq e_{Nk}\}$$

$$max\{z_{Nk} \ : \ B'y = c'_B, \ N'y + z_N = c'_N, \ z_B = 0, \ z_N \geq 0\}$$

Note que $c'x \geq 0$ para todo vetor factível x . $\Diamond$

7.2 Fluxo de custo mínimo com um único produto

Nesta seção vamos identificar como conjuntos de nós e arcos as condições apresentadas sa seção anterior.

Seja agora A uma matriz de incidência de uma rede conectada com $m+1$ nós e n arcos, onde supomos já retirada uma de suas linhas, ou seja, $posto(A) = m$. Seja B uma árvore geradora de A, com $A = [B \mid N]$
(Nesta seção, estamos denotando um arco por j ou por e_j, indiferentemente).

A inversa da base, $B^{-1} = (\beta_{ij})$, onde $\beta_{ij} \in \{0, -1, +1\}$, é tal que, considerando a cadeia na árvore geradora, do nó raiz ao nó j, temos a seguinte representação para cada linha i:
$\beta_{ij} = 0$ se esta cadeia não utiliza o i-ésimo arco básico;
$\beta_{ij} = -1$ se esta cadeia utiliza o i-ésimo arco básico no seu sentido;
$\beta_{ij} = +1$ se esta cadeia utiliza o i-ésimo arco básico no seu sentido contrário.

Seja $\bar{N} = B^{-1}N$, com $\bar{N} = (N_{ij})$. Sabemos que uma coluna de $\bar{N} \in \{0, -1, +1\}$, e cada coluna de $\bar{N}$ pode ser obtida através de uma cadeia em B que liga o nó origem ao nó destino do arco j . Esta cadeia, mais o arco j, formam um único ciclo em B cujo sentido é definido por j, de modo que:
$\bar{N}_{ij} = 0$ se o i-ésimo arco de B não está no ciclo;
$\bar{N}_{ij} = -1$ se o i-ésimo arco de B está no ciclo e concorda com sua orientação;
$\bar{N}_{ij} = +1$ se o i-ésimo arco de B está no ciclo e tem orientação contrária ao ciclo.

Cada linha i de B^{-1} define uma partição no conjunto de nós:
$\mathcal{M}_0 = \{j \ / \ \beta_{ij} = 0\}$ e $\mathcal{M}_1 = \{j \ / \ \beta_{ij} \neq 0\}$.
Neste caso, a linha i de B^{-1}, denotada por β_i, possui todos os seus elementos não negativos ou todos os seus elementos não positivos, dependendo da direção do i-ésimo arco básico no caminho da raiz para as folhas. Considere $y_i = |\beta_{ij}|$, $\forall i$ e $z = -A'y$.

Seja

$$\sigma = \begin{cases} +1 & \text{se } \beta_i \geq 0 \\ -1 & \text{se } \beta_i < 0 \end{cases}$$

Cada linha i de $\bar{N}$ define uma partição no conjunto de arcos:
$\mathcal{N}_0 = \{j \notin B : \bar{N}_{ij} = 0\} \cup \{j \in B : j \neq i\} = \{j : z_j = 0\}$: arcos que não

pertencem ao corte gerado por $\mathcal{M}_1$;
$\mathcal{N}_+ = \{j \notin B : \bar{N}_{ij} = \sigma\} \cup \{i : \sigma = +1\} = \{j : z_j = -1\}$: conjunto dos arcos que saem de $\mathcal{M}_1$;
$\mathcal{N}_- = \{j \notin B : \bar{N}_{ij} = -\sigma\} \cup \{i : \sigma = -1\} = \{j : z_j = +1\}$: conjunto dos arcos que chegam em $\mathcal{M}_1$.

Para $y \in \{0,\ 1\}^m$, a demanda total associada ao conjunto $\mathcal{M}_1$ é dada por $b'y = \sum_{i \in \mathcal{M}_1} b_i$.

Para $x \in \{0,\ +1,\ -1\}^n$, o custo líquido associado à partição de arcos é dado por $c'x = \sum_{j \in \mathcal{N}_-} c_j - \sum_{j \in \mathcal{N}_+} c_j$.

Teorema:
Toda solução homogênea extrema de $\{Ax = b\ ,\ \ x \geq 0\}$ *é um circuito.*

prova: considere os pivoteamentos feitos para obter uma solução básica factível do sistema $\{Ax = 0\ ,\ \ \mathbf{1}'x = \mathbf{1}\}$, com $x \geq 0$:

x	rhs
A	$\mathbf{0}$
$\mathbf{1}$	1

$\Rightarrow$

x_B	x_N	rhs
B	N	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{1}$	1

$\Rightarrow$

x_B	x_N	rhs
I	$\bar{N}$	$\mathbf{0}$
$\mathbf{0}$	$\mathbf{1} - \mathbf{1}'\bar{N}$	1

Um pivô positivo de uma coluna não básica s exige que $1 - \sum_i \bar{N}_{is} > 0$. Para um pivoteamento em uma coluna s não básica que produza um novo rhs não negativo é necessário que toda a coluna s de $\bar{N}$ seja não positiva. Isto define um *circuito* cujo comprimento é dado pelo pivô positivo. $\Diamond$

Teorema:
Toda solução homogênea extrema de $\{A'y + z = c\ ,\ \ z \geq 0\}$ *é um subconjunto de nós sem arcos que saiam, juntamente com os arcos que chegam.*

prova: considere os pivoteamentos feitos para obter uma solução básica factível do sistema $\{A'y + z = 0\ ,\ \ \mathbf{1}'y + \mathbf{1}'z = 1\}$, com $z \geq 0$:

y	z	rhs
A'	I	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{1}$	1

$\Rightarrow$

y	z_N	z_B	rhs
B'		I	$\mathbf{0}$
N'	I		$\mathbf{0}$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	1

$\Rightarrow$

y	z_N	z_B	rhs
I		$(B^{-1})'$	$\mathbf{0}$
	I	$-(\bar{N})'$	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	1

Um pivô positivo de uma coluna não básica r exige que $1 + \sum_j \bar{N}_{rj} - \sum_j \beta_{rj} > 0$. Para um pivoteamento em uma coluna r não básica que produza um novo

rhs não negativo é necessário que toda a linha r de B^{-1} seja não positiva e toda a linha r de $\bar{N}$ seja não negativa. Isto define um subconjunto de nós sem arcos que saiam, juntamente com os arcos que chegam. $\Diamond$

Infactibilidade primal:
Neste caso, um problema de fluxo de custo mínimo é primal infactível se e somente se existe um vetor y que satisfaz $A'y \geq 0$, com $b'y < 0$. Isto significa que existe um subconjunto de nós $\mathcal{M}_1$ definido por um vetor $y \in \{1,\ 0\}^m$, com demanda total $b'y < 0$, com um conjunto vazio de arcos que chegam em $\mathcal{M}_1$, definido por $z = -A'y \in \{0,\ 1\}^n$. Em outras palavras, não existe nenhum arco para satisfazer a demanda dos nós de $\mathcal{M}_1$.

Inexistência de solução primal factível interior:
Neste caso, existe um vetor y que satisfaz $A'y \geq e_k$, com $b'y = 0$. Isto significa que existe um subconjunto de nós $\mathcal{M}_1$ definido por um vetor $y \in \{1,\ 0\}^m$, com demanda total $b'y = 0$, com um conjunto vazio de arcos que chegam em $\mathcal{M}_1$, definido por $z = -A'y \in \{0,\ 1\}^m$. Observe que todo arco de $\mathcal{N}_-$ que sai de $\mathcal{M}_1$ é trivial.

Degenerescência primal:
Neste caso, existe um vetor y e uma base B de $A = [B \mid N]$ tal que $B'y \geq e_{Bk}$, com $b'y = 0$. Isto significa que existe um subconjunto de nós $\mathcal{M}_1$ definido por um vetor $y \in \{1,\ 0\}^m$, com demanda $b'y = 0$. Observe que todo arco na partição $\mathcal{N}_- \cup \mathcal{N}_+$ definido por $\mathcal{M}_1$ com o vetor $z = -A'y$ pode ser considerado degenerado.

Infactibilidade dual:
Neste caso, existe um vetor x que satisfaz $Ax = 0,\ x \geq 0$, com $c'x < 0$. Isto significa que existe um circuito negativo definido por um vetor $x \in \{1,\ 0\}^n$, com custo líquido $c'x < 0$.

Inexistência de solução dual factível interior:
Neste caso, existe um vetor x que satisfaz $Ax = 0,\ x \geq e_k$, com $c'x = 0$. Isto significa que existe um circuito nulo definido por um vetor $x \in \{1,\ 0\}^n$, com custo líquido $c'x = 0$.

Degenerescência dual:
Neste caso, existe um vetor x que satisfaz $Ax = 0,\ x_N \geq e_{Nk}$, com $c'x = 0$.

Isto significa que existe um ciclo nulo definido por um vetor $x \in \{0, +1, -1\}^n$, com custo líquido $c'x = 0$. Note que, qualquer um dos arcos do ciclo pode ser considerado básico na árvore geradora; se ele não tem o sinal correto é necessário multiplicar este vetor x por -1, alterando o sentido do ciclo.

7.3 Multifluxo

Nesta seção vamos estender os conceitos anteriores para o problema de multifluxo.

Considere agora o problema de multifluxo, mas com variáveis não negativas, já na forma padrão (x^{p+1} é o vetor das variáveis de folga relativas às restrições de acoplamento), e o problema dual correspondente:

$$min\{\sum_{k}^{p}(c^k)'x^k \ : \ Ax^k = b^k, \ \forall k, \ \sum_{k}^{p+1} x^k = d, \ x^k \geq 0, \ \forall k\}$$

$$max\{\sum_{k}^{p}(b^k)'y^k - d'y^{p+1} \ : A'y^k - y^{p+1} + z^k = c^k, \ y^{p+1}, \ z^k \geq 0, \ \forall k\}$$

Soluções homogêneas:

O sistema $\{Ax^k = 0, \ \sum_{k=1}^{p+1} x^k = 0, \ x^k \geq 0, \ \forall k\}$ não possui solução não trivial. Assim, o problema primal não apresenta solução homogênea extrema.

As soluções homogêneas extremas de $\{A'y^k + z^k - y^{p+1} = c^k, \ y^{p+1}, \ z^k \geq 0, \ \forall k\}$ não possuem caracterização simples, pois uma base de multifluxo não é totalmente unimodular. Os pivoteamentos feitos no caso de fluxo com um único produto poderiam ser feitos aqui, mas eles se mostram bastante complicados, uma vez que pivoteamentos efetuados nas restrições de acoplamento interferem nos vários produtos e de diversas maneiras, dependendo da coluna escolhida. No entanto, alguns casos mais simples podem ser caracterizados:

(i) Para um produto k, um conjunto de nós definido por $y^k \in \{0, \ 1\}^m$, associado a um corte unidirecional (isto é, composto apenas por arcos que chegam), definidos por $z^k \in \{0, \ 1\}^n$, satisfazendo $A'y^k + z^k = 0$ e $b'y^k = 0$ (similar ao problema de fluxo com um único produto).

(ii) Um conjunto de arcos Y, definidos por $y^{p+1} \in \{0,\ 1\}^n$ associado a cada produto k com:
• ou um conjunto de arcos Y, definidos por $z^k \in \{0,\ 1\}^n$ tal que $z^k - y^{p+1} = 0$ com $y^k = 0$,
• ou um conjunto de nós definido por $y^k \in \{0,\ 1\}^m$, associado a um corte com arcos que saem, definidos por Y, e arcos que entram, definidos por $z^k \in \{0,\ 1\}^n$, tal que $A'y^k + z^k - y^{p+1} = 0$ satisfazendo $\sum_k (b^k)'y^k - d'y^{p+1} = 0$.

Como exemplo, considere um problema com 2 produtos, cuja rede com 4 nós e 6 arcos é dada por:

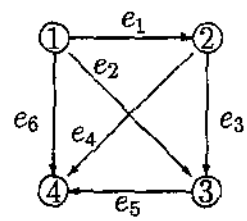


Figura 7.1: *rede exemplo para solução homogênea*

A matriz A associada é

$$A = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 1 \\ \hline -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \end{array}\right] \end{array}$$

onde a primeira linha será retirada (escolhemos como raiz o nó 1).

Exemplo de (i):

Sejam $b^1 = (2,\ -2,\ 3,\ -3)'$, $b^2 = (0,\ 1,\ 2,\ -3)'$ e $d = (5,\ 5,\ 5,\ 5,\ 5,\ 5)'$.
Se $y^1 = (0,\ 0,\ 1,\ 1)'$, $y^2 = (0,\ 0,\ 0,\ 0)'$,
$z^1 = (0,\ 1,\ 1,\ 1,\ 0,\ 1)'$, $z^2 = (0,\ 0,\ 0,\ 0,\ 0,\ 0)'$,
$y^3 = (0,\ 0,\ 0,\ 0,\ 0,\ 0)'$,

realmente temos que $A'y^1 + z^1 = 0, \quad A'y^2 + z^2 = 0, \quad z^1,\ z^2 \geq 0$ e $(b^1)'y^1 + (b^2)'y^2 - d'y^3 = 0$.

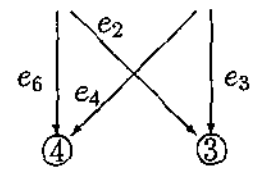


Figura 7.2: *exemplo(i) – produto 1*

Exemplo de (ii):

Sejam $b^1 = b^2 = (2,\ 6,\ -1,\ -7)'$ e $d = (2,\ 2,\ 2,\ 2,\ 3,\ 3)'$.
Se $y^3 = (0,\ 0,\ 0,\ 1,\ 1,\ 0)'$, então $Y = \{e_4,\ e_5\}$.

Para o **produto 2**, consideramos:
$y^2 = (0,\ 0,\ 0,\ 0)', \quad z^2 = (0,\ 0,\ 0,\ 1,\ 1,\ 0)'$,
de modo que $z^2 - y^3 = 0, \quad z^2,\ y^3 \geq 0$.

Para o **produto 1**, consideramos:
$y^1 = (0,\ 0,\ 1,\ 1)', \quad z^1 = (1,\ 1,\ 0,\ 0,\ 0,\ 0)'$,
de modo que $A'y^1 + z^1 - y^3 = 0, \quad z^1,\ y^3 \geq 0$ e $(b^1)'y^1 + (b^2)'y^2 - d'y^3 = 0$.

Figura 7.3: *exemplo(ii) – produtos 1 e 2*

A caracterização da **infactibilidade primal** e da **inexistência de solução primal factível interior** ficarão restritas aos casos discutidos nas soluções homogêneas anteriores.

Infactibilidade primal:
Existe uma solução dual homogênea extrema de valor negativo.

Inexistência de solução primal factível interior:
Existe uma solução dual homogênea de valor nulo.

Infactibilidade dual:
Nenhum problema de multifluxo pode ter seu dual infactível.

Inexistência de solução dual infactível interior:
Nenhum problema de multifluxo apresenta seu dual sem solução dual factível interior.

Capítulo 8

Conclusões

Neste trabalho, apresentamos um estudo computacional do método primal-dual de pontos interiores aplicado ao problema de multifluxo.

Iniciamos pela formulação do problema de multifluxo de uma maneira geral, indicando algumas técnicas especiais para a resolução deste problema, assim como apresentamos a caracterização de sua base.

Dado que os métodos mais eficientes para a resolução deste problema são os de pontos interiores, aplicamos um deles — o método primal-dual — ao problema de multifluxo. Nesta aplicação, destacamos o passo da resolução do sistema linear das direções. Nossa primeira implementação considerou um método direto — a fatoração de Cholesky. Posteriormente, implementamos um método iterativo — o gradiente conjugado — apresentando um precondicionador adequado (*floresta geradora máxima*).

A implementação considerando o gradiente conjugado precondicionado resultou num programa — **mult** — sobre o qual fizemos muitos testes computacionais, através de um gerador de problemas de multifluxo. Este programa também foi testado com alguns problemas da Netlib, além de ser sido comparado com o programa **Lipsol**. Todos os testes foram cuidadosamente discutidos no Capítulo 6.

Discutimos como calcular de maneira eficiente várias operações envolvendo a matriz de incidência A e a própria estrutura do problema de multifluxo.

Apresentamos ainda uma heurística para a obtenção de uma base ótima a partir de uma solução interior "quase-ótima" fornecida pelo método primal-dual, que obteve resultados bons para problemas não degenerados. Temos ainda uma caracterização parcial de ausência de ponto interior para multi-

fluxo, que ainda não encontramos na literatura.

8.1 Conclusões dos experimentos computacionais

Dos experimentos computacionais realizados, podemos concluir:

• **Parâmetro** μ: as duas expressões comumente utilizadas na literatura não apresentam resultados significantemente diferentes.

• **Precondicionadores:** o precondicionador *diagonal* é mais simples e é eficiente nas iterações iniciais do primal-dual; o *floresta geradora máxima* é mais caro, mas é mais eficiente. Usar somente o *diagonal* causa um grande número de iterações no método dos gradiente conjugados; usar somente o *floresta geradora máxima* pode levar a um aumento desnecessário do tempo de cpu. O mais adequado é combinar os dois precondicionadores, começando pelo *diagonal* e, em alguma iteração do método primal-dual, substituir este precondicionador pelo *floresta geradora máxima*. Os experimentos mostraram que sempre existe um momento, isto é, uma iteração ideal do método primal-dual onde é mais conveniente fazer esta mudança; esta iteração ficou em torno de 7 nos experimentos efetuados.

• **Pontos iniciais:** três pontos iniciais foram testados: um, que construímos para problemas de multifluxo (primeiro), outro, que inicialmente foi apresentado por Portugal e outros [58] para problemas de fluxo de custo mínimo com um único produto, sobre o qual fizemos uma extensão para multifluxo (segundo) e um outro ainda, apresentado por Mehrotra [51] para problemas gerais de programação linear (terceiro). Comparados o número médio de iterações do gradiente conjugado, o número de iterações do primal-dual e o tempo de cpu, pouca diferença se deu entre eles. Como o primeiro ponto apresentou um desempenho um pouco melhor e ele é mais simples de ser construído, concluímos por ele.

• **Parada do gradiente conjugado:** aqui, testamos separadamente e também combinamos quatro critérios (além do número máximo de iterações permitidas): resíduo absoluto, resíduo relativo, cosseno fixo e cosseno variável. Naturalmente, nossas conclusões são consequência das tolerâncias ε_{gc} e ε_{cos}, que devemos fixar. Trabalhamos com $\varepsilon_{gc} = 10^{-8}$, mas ε_{cos} tomou vários

valores em nossos experimentos. Quanto mais relaxada esta tolerância, mais ganhamos no tempo de cpu e no número médio de iterações do gradiente conjugado. No entanto, corremos o risco de gerar direções ruins, que nos levam a um aumento no número de iterações do método primal-dual.

Testando isoladamente os quatro critérios, concluímos que o resíduo relativo é melhor do que o resíduo absoluto, assim como o cosseno variável é melhor do que o fixo. Comparando os dois critérios — resíduo e cosseno — podemos indicar que, se precisamos de uma precisão mais acurada, é melhor usarmos o resíduo (relativo); se podemos trabalhar com uma precisão mais relaxada, podemos usar o cosseno (variável). Geralmente, neste último caso, o número médio de iterações do gradiente conjugado é menor, o que também costuma levar a um menor tempo de cpu.

• **Direções anteriores:** a tentativa de iniciar o gradiente conjugado com a direção $\Delta\hat{y}$ que foi obtida no final do gradiente conjugado da iteração anterior do método primal-dual, em vez da direção nula (que é a usual), não se mostrou vantajosa para os diversos testes realizados. Criamos inclusive um parâmetro extra (*it.d*) para indicar o número da iteração do método primal-dual onde queremos iniciar o gradiente conjugado com a direção anterior. Experimentamos também uma redução na própria direção, antes de iniciar a próxima execução do gradiente conjugado — em nenhum destes casos o uso da direção anterior, ao invés da direção nula, provocou alguma melhora nos resultados. Experimentamos ainda as alternativas anteriores usando como critério de parada do gradiente conjugado o resíduo relativo, mas os resultados não se alteraram. Além disso, começar o gradiente conjugado com a direção nula é sempre mais fácil e mais barato. Assim, concluímos por iniciar o método do gradiente conjugado pela direção nula.

• **Informações das matrizes de scaling:** uma contagem do número de elementos "pequenos" (tendendo a zero) e do número de elementos "grandes" (tendendo a infinito) nas matrizes de *scaling* obtidas nas iterações do método primal-dual nos permite concluir, por exemplo, se já temos informações suficientes para substituir o precondicionador *diagonal* pelo precondicionador *floresta geradora máxima*. Destes experimentos também podemos observar, a respeito da heurística para obtenção de uma base ótima, numa certa iteração do método primal-dual, se o número de elementos "grandes" em cada matriz de *scaling* Θ^k é suficiente para construir uma árvore (ou floresta) geradora para o produto k, ou ainda, se este número é mais do que o suficiente, in-

dicando que o produto k vai apresentar vários arcos em ciclo na árvore (ou floresta) geradora do produto k. Concluindo, estas informações poderão ser utilizadas como um indicador na troca do precondicionador *diagonal* para o *floresta geradora máxima* e também como indicador do número da iteração do método primal-dual onde já podemos introduzir a heurística que tentará obter uma solução básica ótima.

• **Heurística para obter uma solução básica ótima:** para problemas não degenerados, concluímos que a heurística dada pelo Algoritmo 5.1 apresenta sucesso em quase todos os problemas. No entanto, para problemas com degeneração dual ou primal (principalmente), ela falha na maioria das vezes.

• Também foram testados com sucesso problemas sem ponto interior factível, problemas infactíveis e problemas com redes desconectadas.

8.2 Trabalhos futuros

Como estudos futuros, podemos destacar:

Melhorias no programa mult

Quando executamos o programa **mult** com os problemas da Netlib e também quando o comparamos com o **Lipsol** notamos que, embora o tempo de cpu possa ser considerado satisfatório, o número de iterações do primal-dual apresentado por **mult** sempre foi maior. Várias melhorias podem ser feitas neste programa. Algumas, citamos a seguir. Várias destas modificações sugerem uma nova formulação para o problema, no sentido de que ele possa vir a ter várias partições nas variáveis e nas restrições:

• dos n arcos existentes, supor que n_a deles são de fato acoplados, com $n_a \leq n$. Nos diversos problemas da Netlib, por exemplo, verificamos que poucos arcos são acoplados. Assim, a dimensão do sistema linear das direções será $p(m-1)+n_a$. Na verdade, todas as informações referentes ao acoplamento terão dimensão n_a em vez de n.

• supor a existência de três tipos de variáveis: $x_j^k \geq 0$, $0 \leq x_j^k \leq u_j^k$ e x_j^k livre. Isto acarreta, por exemplo, modificações no teste da razão. Além disso, variáveis x_j^k não canalizadas não possuem s_j^k e w_j^k correspondentes.

• dependendo do tipo de cada variável (canalizada, não negativa ou livre), o ponto inicial deverá agora ser alterado.

• admitir matrizes diferentes para os produtos. Podemos supor que existe uma matriz completa A, com n arcos e m nós mas, para os diversos produtos, vários arcos podem ser desconsiderados.

• **preprocessamento:** observamos uma grande necessidade de incluir este item. De fato, ele precisa primeiramente ser estudado. No entanto, podemos fazer algumas implementações mais simples, de imediato:
(i) checar se existe $d_j = 0$. Caso exista, se $x_j^k \geq 0$, $\forall k$, a restrição j do acoplamento $\sum_{k=1}^{p+1} x_j^k = 0$ indica que $x_j^k = 0$, $\forall k$. Isto significa que o arco j pode ser eliminado de todos os produtos e também do acoplamento.
(ii) verificar a existência de nós *folha*, isto é, aqueles para os quais só existe um arco incidente. Estes nós podem ser eliminados da rede. Devemos verificar se cada nó *folha* é um nó de oferta (demanda) e alocar o fluxo no arco que sai (chega) dele, bem como atualizar a oferta/demanda do nó destino (origem) deste único arco. Infactibilidades podem ser verificadas neste ponto. Este tipo de procedimento deve ser feito várias vezes, até que não exista mais nós *folha*, pois, após eliminarmos um nó deste tipo, algum outro que estava conectado a ele e que não era *folha* pode vir a ser.

• manter o mesmo precondicionador *árvore geradora máxima* em algumas iterações consecutivas para o mesmo produto, dependendo de algum critério pre-estabelecido.

• utilizar a contagem do número de elementos "grandes" nas matrizes de *scaling* para introduzir o precondicionador *floresta geradora máxima*. Desta maneira, o número da iteração do método primal-dual onde é feita a troca de precondicionador (parâmetro $it.fg$) é determinada durante o processo, ao invés de ser decidida a priori.

• utilizar alocação dinâmica de memória.

• utilizar como direção inicial para o método dos gradientes conjugados o lado direito do sistema linear.

• ainda neste método, alterar a redução de 0.95 no critério de parada do cosseno (variável). Por exemplo, considerar $\varepsilon_{cos} \leftarrow 0.90\varepsilon_{cos}$ ou ainda alguns valores menores do que 0.90.

Outros estudos

Temos outros estudos a fazer e, depois disso, sua implementação. Por exemplo:

• melhorar a heurística para obtenção de uma base ótima, para que ela obtenha sucesso também com problemas degenerados.

• nas últimas iterações do método primal-dual, introduzir como precondicionador a base fornecida por esta heurística.

• outra tentativa é melhorar o precondicionador combinado, aproveitando as "considerações finais" do final do Capítulo 4 e introduzindo a fatoração de Cholesky incompleta neste precondicionador.

• testar a variante preditor-corretor na resolução do sistema linear das direções.

• estudar a possibilidade de trabalhar com o sistema linear das direções *aumentado*, isto é, considerar $\Delta\widehat{y}$ e $\Delta\widehat{x}$ como variáveis deste sistema. Isto poderá trazer maior estabilidade e precisão ao método primal-dual.

• obter novas fórmulas ou algoritmos para calcular o parâmetro de centragem μ . Por exemplo, considerar $\eta^\ell \in [0,\ 1)$ em função da dimensão do problema, e não necessariamente fixo em todas as iterações do método primal-dual. Outra sugestão é utilizar $(\gamma^\ell)^2$ ao invés de γ^ℓ sempre que $\gamma^\ell < 1$.

• estender este estudo para grafos generalizados.

• continuar o estudo sobre a degenerescência, com o objetivo de completar a caracterização das soluções duais homogêneas extremas.

Bibliografia

[1] Adler, I.; Karmarkar, N.; Resende, M. e Veiga, G., "Data Structures and Programming Techniques for the Implementation of Karmarkar's Algorithms", *ORSA Journal on Computing*, 1, 84-106 (1989).

[2] Adler, I.; Karmarkar, N.; Resende, M. e Veiga, G., "An Implementation of Karmarkar's Algorithm for Linear Programming", *Mathematical Programming*, 44, 297-335 (1989).

[3] Ahuja, R.K.; Magnanti, T.L. e Orlin, J.B., *Network Flows — Theory, Algorithms and Applications*, Prentice Hall, New Jersey, 1993.

[4] Ali, A.; Helgason, R.; Kennington, J.L. e Lall, H., "Computational Comparison among Three Multicommodity Network Flow Algorithms", *Operations Research*, 28, 4, 995-1000 (1980).

[5] Assad, A.A., "Multicommodity Network Flows - A Survey", *Networks*, 8, 1, 37-91 (1978).

[6] Axelsson, O. e Barker, V.A., *Finite Element Solution of Boundary Value Problems* , Academic Press, Inc., 1984.

[7] Bazaraa, M.S.; Jarvis, J.J. e Sherali, H.D., *Linear Programming and Network Flows*, John Wiley & Sons, New York, 1990.

[8] Bennett, J.M., "An Approach to Some Structured Linear Programming Problems", *Operations Research*, 14, 636-645 (1966).

[9] Brassard, G. e Bratley, P., *Algorithms — Theory and Practice*, Prentice Hall, New Jersey, 1990.

[10] Castro, J. "A Specialized Interior Point Algorithm for Multicommodity Network Flows", To appear in *SIAM Journal on Optimization.*

[11] Castro, J. e Nabona, N. "An Implementation of Linear and Nonlinear Multicommodity Network Flows", *European Journal of Operational Research*, 92, 37-53 (1996).

[12] Chen, H. e Dewald, C.G., "A Generalized Chain Labelling Algorithm for Solving Multicommodity Flow Problems", *Computers and Operations Research*, 1, 437-465 (1974).

[13] Cremeans, J.E.; Smith, R.A. e Tyndall, G.R., "Optimal Multicommodity Network Flows with Resource Allocation", *Naval Research Logistics Quarterly*, 17, 269-280 (1970).

[14] Dantzig, G.B. e Wolfe, P. ,"Decomposition Principle for Linear Programs", *Operations Research*, 8, 101-111 (1960).

[15] Ford, L.R. e Fulkerson, D.R., "A Suggested Computation for Maximal Multicommodity Network Flows", *Management Science*, 5, 97-101 (1958).

[16] Frangioni, A. "Dual Ascent Methods ans Multicommodity Flows", Ph.D. Thesis TD 5/97, Dip. di Informatica, Univ. di Pisa (1997).

[17] Geoffrion, A.M., "Primal Resource-Directive Approaches for Optimizing Nonlinear Decomposable Systems", *Operations Research*, 18, 3, 375-403 (1970).

[18] Geoffrion, A.M. e Graves, G.W., "Multicommodity Distribuition System Design By Benders Decomposition", *Management Science*, 20, 5, 822-844 (1974).

[19] Golub, G.H. e Van Loan, F., *Matrix Computations*, 2a. edition, The Johns Hopkins University Press, Baltimore, 1989.

[20] Gonzaga, C.C., "Path-Following Methods for Linear Programming", *SIAM Review*, 34, 167-224 (1992).

[21] Grigoriadis, M.D. e White, W.W., "A Partitioning Algorithm for the Multicommodity Network Flow Problem", *Mathematical Programming*, 3, 157-177 (1972).

[22] Hartman, J.K. e Lasdon, L.S., "A Generalized Upper Bounding Method for Doubly Coupled Linear Programs", *Naval Research Logistics Quarterly*, 17, 4, 411-429 (1970).

[23] Hartman, J.K. e Lasdon, L.S., "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems", *Networks*, 1, 333-354 (1972).

[24] Held, M.; Wolfe, P. e Crowder, H., "Validation of Subgradient Optimization", *Mathematical Programming*, 6, 62-88 (1974).

[25] Helgason, R.V., "A Lagrangean Relaxation Approach to the Generalized Fixed Charge Multicommodity Minimal Cost Network Flow Problem", unpublished dissertation, Department of Operations Research, Southern Methodist University, Dallas, Tex. (1980).

[26] Helgason, R.V. e Kennington, J.L., "A Product Form Representation of the Inverse of a Multicommodity Cycle Matrix", *Networks*, 7, 297-322 (1977).

[27] Jarvis, J.J., "On the Equivalence Between the Node-Arc and Arc-Chain Formulation for the Multicommodity Maximal Flow Problem", *Naval Research Logistics Quarterly*, 15, 525-529 (1969).

[28] Jarvis, J.J. e Keith, P.D., "Multicommodity Flows with Upper and Lower Bounds", Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Ga. (1974).

[29] Jarvis, J.J. e Martinez, O.M., "A Sensitivity Analysis of Multicommodity Network Flows", *Transportation Science*, 11, 4, 299-306 (1977).

[30] Jones, K.L.; Lustig, I.J.; Farvolden, J.M.; Powell, W.B., "Multicommodity Network Flows : The Impact of Formulation on Decomposition", *Mathematical Programming*, 62, 95-117 (1993).

[31] Kamath, A. e Palmon, O., "Improved Interior Point Algorithms for Exact and Approximate Solution of Multicommodity Flow Problems", Technical Report, Department of Computer Science, Stanford University, (1994).

[32] Kapoor, S. e Vaidya, P.M., "Speeding up Karmarkar's Algorithm for Multicommodity Flows", *Mathematical Programming*, 73, 111-127 (1996).

[33] Karmarkar, N.K., "A New Polynomial Time Algorithm for Linear Programming", *Combinatorica*, 4, 373-395 (1984).

[34] Kaul, R.N., "An Extension of Generalized Upper Bounded Techniques for Linear Programming", ORC Report No. 65-27, Department of Operations Research, University of California at Berkeley (1965).

[35] Kennington, J.L., "A Survey of Linear Cost Multicommodity Network Flows", *Operations Research*, 26, 2, 209-236 (1978).

[36] Kennington, J.L., "Solving Multicommodity Transportation Problems Using a Primal Partitioning Simplex Technique", *Naval Research Logistics Quarterly*, 24, 2, 309-325 (1977).

[37] Kennington, J.L. e Helgason, R.V., *Algorithms for Network Programming*, John Wiley & Sons, New York, 1980.

[38] Kennington, J.L. e Shalaby, M., "An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems", *Management Science*, 23, 9, 994-1004 (1977).

[39] Kojima, M., Megiddo, N. e Mizuno, S.,"A Primal- Dual Infeasible-Interior-Point Algorithm Linear Programming", *Mathematical Programming*, 61, 263-280 (1993).

[40] Kojima, M., Mizuno, S. e Yoshise, A., "A Primal- Dual Interior-Point Method for Linear Programming", *Progress in Mathematical Programming, Interior-Point and Related Methods*, Spring-Verlag, New York, 29-47 (1989).

[41] Lasdon, L.S., *Optimization Theory for Large Systems*, MacMillan Co., New York, 1970.

[42] Li, G. e Lustig, I.J.,"An Implementation of a Parallel Primal-Dual Interior Point Method for Multicommodity Flow Problems", Technical Report TR92-2, Department of Mathematical Sciences, Rice University (1992).

[43] Lustig, I.J., Marsten, R.E. e Shanno, D.F., "Computational Experience with a Primal-Dual Interior-Point Method for Linear Programming", *Linear Algebra and its Applications*, 152, 191-222 (1991).

[44] Lustig, I.J., Marsten, R.E. e Shanno, D.F., "On Implementing Mehrotra's Predictor-Corretor Interior Point Method for Linear Programming", *SIAM Journal on Optimization*, 2, 435-449 (1992).

[45] Maier, S.F., "A Compact Inverse Scheme Applied to a Multicommodity Network with Resource Constraints", in R. Cottle and J. Krarup, Eds., *Optimization Methods for Resource Allocation*, The English University Press, London, England, 179-203 (1974).

[46] Mangasarian, O.L., *Nonlinear Programming*, McGraw-Hill Book Company, 1969.

[47] Masuzawa, K., Mizuno, S. e Mori, M., "A Polynomial Time Interior Point Algorithm for Minimum Cost Flow Problems", *Journal of the Operations Research, Society of Japan*, 33, 2, 157-167 (1990).

[48] McCallum, C.J., "A Generalized Upper Bounding Approach to a Communication Network Planning Problem", Working Paper, Bell Laboratories, Holmdel, New Jersey (1974).

[49] McShane, K.A., Monma, C.L. e Shanno, D.F., "An Implementation of a Primal-Dual Interior-Point Method for Linear Programming", *ORSA Journal on Computing*, 1, 70-83 (1989).

[50] Megiddo, D., "Pathways to the Optimal Set in Linear Programming", *Progress in Mathematical Programming, Interior- Point and Realated Methods*, Spring-Verlag, New York, 131-158 (1989).

[51] Mehrotra, S., "On the Implementation of a Primal- Dual Interior Point Method", *SIAM Journal on Optimization*, 2, 575-601 (1992).

[52] Mehrotra, S., "Implementations of Affine Scaling Methods: Approximate Solutions of Systems of Linear Equations Using Preconditioned Gradient Methods", *ORSA Journal on Comptuting*, 4, 103-118, 1992.

[53] Mehrotra, S. e Wang, J.-S., "Conjugate Gradient Based Implementation of Interior Point Methods for Network Flow Problems", Technical

Report - 95-70, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, USA (1995).

[54] Monteiro, R.C. e Adler, I., "Interior Path-Following Primal-Dual Algorithms, Part 1: Linear Programming", *Mathematical Programmnig*, 44, 27-41 (1989).

[55] Mizuno, S. e Masuzawa, K., "Polynomial Time Interior Point Algorithms for Transportation Problems", *Journal of the Operations Research, Society of Japan*, 32, 3, 371-382 (1989).

[56] Oliveira, A.R.L., "A New Class of Preconditioners for Large-Scale Linear Systems from Interior Point Methods for Linear Programming", PHD Thesis, Rice University, Houston, Texas (1997).

[57] Portugal, L.F., Bastos, F., Júdice, J.J., Paixão, J. e Terlaky, T., "An Investigation of Interior Point Algorithms for the Linear Transportation Problem", *SIAM Journal on Scientific Computing*, 17, 5, 1202-1223 (1996).

[58] Portugal, L.F., Resende, M.G.C., Veiga, G. e Júdice, J.J., "A Truncated Primal-Infeasible Dual-Feasible Network Interior Point Method", Technical Report (1994).

[59] Resende, M.G.C. e Pardalos, P.M., "Interior Point Algorithms for Network Flow Problems", To appear in *Advances in Linear and Integer Programming.*

[60] Resende, M.G.C. e Veiga, G., "An Implementation of the Dual Affine Scaling Algorithm for Minimum-Cost Flow on Bipartite Uncapacitated Networks", *SIAM J. Optimization*, 3, 3, 516-537 (1993).

[61] Resende, M.G.C. e Veiga, G., "Computing the Projection in an Interior Point Algorithm: An Experimental Comparison", *Investigatión Operativa*, 3, 81-92 (1993).

[62] Resende, M.G.C. e Veiga, G., "An Efficient Implementation of a Network Interior Point Method", *in Network Flows and Matching: First DIMACS Implementation Challenge*, Johnson, D.S. e McGeoch, C.C., eds., vol.12 of DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, 299-384, 1993.

[63] Resende, M.G.C. e Veiga, G., "A Dual Affine Scaling Algorithm for Minimum Cost Network Flow", Technical Report (1990).

[64] Robacker, J.T., "Notes on Linear Programming: Part XXXVII, Concerning Multicommodity Networks", Memo RM- 1799, The Rand Corporation, Santa Monica, California (1956).

[65] Rosen, J.B., "Primal Partition Programming for Block Diagonal Matrices", *Numerische Mathematik*, 6, 250-260 (1964).

[66] Saigal, R., "Multicommodity Flows in Directed Networks", ORC 67-38, Operations Research Center, University of California, Berkeley (1967).

[67] Swoveland, C., "Decomposition Algorithms for the Multicommodity Distribution Problem", Working Paper #184, Western Management Science Institute, University of California, Los Angeles (1971).

[68] Swoveland, C. , "A Two-Stage Decomposition Algorithm for a Generalized Multicommodity Flow Problem", *INFOR* 11, 3, 232-244 (1973).

[69] Tomlin, J.A., "Minimum-Cost Multicommodity Network Flows", *Operations Research*, 14, 45-51 (1966).

[70] Tomlin, J.A., "Mathematical Programming Models for Traffic Network Problems", unpublished dissertation, Department of Mathematics, University of Adelaide, Australia (1967).

[71] Wallacher, C. e Zimmermann, U. "A Combinatorial Interior Point Method for Network Flow Problems", *Mathematical Programming*, 56, 321-335 (1992).

[72] Weigel, H.S. e Cremeans, J. E., "The Multicommodity Network Flow Model Revised to Include Vehicle per Time Period and Mode Constraints", *Naval Research Logistics Quarterly*, 19, 77-89 (1972).

[73] Wollmer, R.D., "Multicommodity Networks with Resource Constraints: The Generalized Multicommodity Flow Problem", *Networks*, 1, 245-263 (1972).

[74] Wright, S.J., *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, Pennsylvania, 1997.

[75] Zhang, Y. e Zhang, D. "On Polynomiality of the Mehrotra- Type Predictor-Corrector Interior-Point Algorithms", *Mathematical Programming*, 68, 303-318 (1995).