SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 16.04.03

Assinatura:

Ana Paula Lampaio J. J. J.

**Desenvolvimento de um algoritmo paralelo
de fase I para o problema de multifluxo:
uma aplicação ao problema de roteamento
de dados**

Luciano Nascimento Moreira
luciano_nm@yahoo.com.br

Orientadora: Profa. Dra. Cassilda Maria Ribeiro
cassilda@feg.unesp.br

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Abril/2003

A Comissão Julgadora:

Profa. Dra. Cassilda Maria Ribeiro

Profa. Dra. Roseli Ap. Francelin Romero

Prof. Dr. Raul Vinhas Ribeiro

Dedicatória

*Aos meus pais,
Juracy e Edna.*

Agradecimentos

Agradeço a Deus pela vida, e por ter me dado forças nesta longa jornada.

Aos meus pais, Juracy e Edna, por terem me dado todo afeto, carinho e amor, e por terem se sacrificado para que eu chegasse aonde cheguei. A vocês, as palavras são pequenas para representar a minha eterna gratidão.

A Cassilda, pela orientação e atenção, por ter me incentivado e mostrado a luz em momentos difíceis, pela enorme paciência e compreensão, e acima de tudo, pela pessoa amiga que sempre foi.

Ao meu irmão Júlio, cujo apoio foi decisivo em alguns momentos.

Ao Sandro, pela pessoa singular que é. Por tudo que fez por mim nos meus dias de angústia e inquietação, e por sua constante tentativa de transformá-los em dias mais felizes.

Aos meus colegas, e em especial ao Otávio e Igor, pelos momentos de risos, entretenimento e papo-furado.

E a FAPESP, cujo apoio foi fundamental para o desenvolvimento deste trabalho.

Índice

LISTA DE FIGURAS IX

LISTA DE TABELAS..... XI

LISTA DE GRÁFICOS XIII

RESUMO XV

ABSTRACT XVII

CAP 1 INTRODUÇÃO 1

CAP 2 CONCEITOS GERAIS 5

2.1 INTRODUÇÃO..... 5

2.2 HISTÓRICO 5

2.3 DEFINIÇÕES 6

2.4 CONSIDERAÇÕES FINAIS..... 15

CAP 3 PROBLEMAS DE FLUXOS E MULTIFLUXOS 17

3.1 MODELO DE FLUXO EM REDES 17

3.2 PRINCIPAIS PROBLEMAS DE FLUXO 19

3.3 PROBLEMAS DE MULTIFLUXO 21

Formulação matemática do problema de multifluxo 21

Formulação nó-arco do problema de multifluxo 22

Formulação arco-caminho do problema de multifluxo 23

3.4 CONSIDERAÇÕES FINAIS..... 24

CAP 4 O PROBLEMA DE ROTEAMENTO DE DADOS 25

4.1 PROBLEMA DE ROTEAMENTO 26

4.2 FORMULAÇÃO DO PROBLEMA PARA UM ÚNICO PRODUTO 27

4.3 GENERALIZAÇÃO PARA VÁRIOS PRODUTOS 30

4.4 RESTRIÇÕES DO PROBLEMA 32

4.5 CONSIDERAÇÕES FINAIS..... 34

CAP 5 ALGORITMO PARALELO PARA ROTEAMENTO DE DADOS..... 35

5.1 MÉTODO DE RELAXAMENTO..... 35

Decomposição do problema..... 35

O Método..... 38

Algoritmo de relaxamento (multifluxo) 40

5.2 O MÉTODO SIMPLEX CONVEXO 41

5.3 CONSIDERAÇÕES FINAIS..... 48

CAP 6 O MÉTODO DE FASE I PARALELO.....	49
6.1 O MÉTODO PRIMAL SIMPLEX PARA FLUXO EM REDES LINEAR.....	49
<i>O Método Primal Simplex para Fluxo Linear</i>	51
6.2 O MÉTODO SIMPLEX DE FASE I.....	53
6.3 O PROBLEMA DE FASE I PARA O MULTIFLUXO	54
6.4 CONSIDERAÇÕES FINAIS	59
CAP 7 COMPUTAÇÃO PARALELA.....	61
7.1 MOTIVAÇÕES PARA O PARALELISMO	61
7.2 CONCEITOS BÁSICOS.....	62
<i>Paralelismo e Concorrência</i>	62
<i>Granulação</i>	64
<i>Speedup e Eficiência</i>	64
7.3 ARQUITETURAS PARALELAS	65
<i>Classificação de Flynn</i>	67
<i>Classificação de Duncan</i>	71
7.4 PROGRAMAÇÃO PARALELA.....	73
<i>Projeto de Algoritmos Paralelos</i>	74
<i>Estilos de Paralelismo</i>	75
<i>Comunicação e Sincronismo</i>	76
<i>Suporte para a Programação Paralela</i>	78
7.5 CONSIDERAÇÕES FINAIS	79
CAP 8 IMPLEMENTAÇÃO E RESULTADOS COMPUTACIONAIS.....	81
8.1 IMPLEMENTAÇÃO DO ALGORITMO.....	81
8.2 RESULTADOS OBTIDOS	86
CONCLUSÕES.....	99
REFERÊNCIAS BIBLIOGRÁFICAS.....	103
ANEXO A	107

Lista de Figuras

FIGURA 2.1 - PONTE DE KÖENIGSBERG.....	6
FIGURA 2.2 - EXEMPLO DE GRAFO G	7
FIGURA 2.3 - EXEMPLOS DE GRAFOS PRÓPRIOS	8
FIGURA 2.4 - EXEMPLOS DE GRAFOS QUE NÃO SÃO PRÓPRIOS	9
FIGURA 2.5 - SUBGRAFO DO GRAFO G DA FIGURA 2.2	9
FIGURA 2.6 - SUBGRAFO GERADOR DO GRAFO G DA FIGURA 2.2.....	9
FIGURA 2.7 - EXEMPLO DE UM CAMINHO NO GRAFO G DA FIGURA 2.2	10
FIGURA 2.8 - EXEMPLO DE UM CICLO NO GRAFO G DA FIGURA 2.2	10
FIGURA 2.9 - EXEMPLO DE GRAFO ACÍCLICO	11
FIGURA 2.10 - EXEMPLO DE GRAFO CONEXO	11
FIGURA 2.11 - EXEMPLO DE ÁRVORE.....	12
FIGURA 2.12 - EXEMPLO DE SEQUÊNCIA ORIENTADA	13
FIGURA 2.13 - EXEMPLO DE ÁRVORE GERADORA DE G.....	13
FIGURA 7.1 - PARALELISMO FÍSICO	63
FIGURA 7.2 - PARALELISMO LÓGICO	63
FIGURA 7.3 - MODELO COMPUTACIONAL SISD	68
FIGURA 7.4 - MODELO COMPUTACIONAL SIMD	69
FIGURA 7.5 - MODELO COMPUTACIONAL MISD	70
FIGURA 7.6 - MODELO COMPUTACIONAL MISD	70
FIGURA 7.7 - CLASSIFICAÇÃO DE DUNCAN.....	72
FIGURA 8.1- EXEMPLO DA ESTRUTURA DE DADOS UTILIZADA PARA REPRESENTAR A SOLUÇÃO	84

Lista de Tabelas

TABELA 8.1 - RESULTADOS DA REDE 01	90
TABELA 8.2 - RESULTADOS DA REDE 02.....	92
TABELA 8.3 - RESULTADOS DA REDE 03.....	94
TABELA 8.4 - RESULTADOS DA REDE 04.....	96

Lista de Gráficos

GRÁFICO 8.1 - SPEEDUP DA REDE 01	91
GRÁFICO 8.2 - EFICIÊNCIA DA REDE 01	91
GRÁFICO 8.3 - SPEEDUP DA REDE 02	92
GRÁFICO 8.4 - EFICIÊNCIA DA REDE 02	93
GRÁFICO 8.5 - SPEEDUP DA REDE 03	94
GRÁFICO 8.6 - EFICIÊNCIA DA REDE 03	95
GRÁFICO 8.7 - SPEEDUP DA REDE 04	96
GRÁFICO 8.8 - EFICIÊNCIA DA REDE 04	97

Resumo

O problema de roteamento de dados em rede de computadores consiste em minimizar o tempo médio de atraso na transmissão de mensagens, escolhendo para elas um caminho ótimo, através dos arcos da rede. Em seu trabalho, Luvezute propôs um algoritmo primal de relaxamento para otimizar o problema de roteamento de dados. O algoritmo proposto por Luvezute resolve iterativamente o problema de multifluxo, decompondo-o da forma mais independente possível, em subproblemas de simples fluxo, sendo um subproblema para cada mensagem. Esta independência entre os cálculos permite que a resolução dos subproblemas seja simultânea, admitindo-se assim uma implementação em paralelo. Nesta dissertação apresentamos um algoritmo paralelo, do tipo Fase I para encontrar uma solução inicial factível para o problema de multifluxo. Este algoritmo permite resolver de maneira mais rápida os problemas de grande porte que é o nosso objetivo inicial. O algoritmo de Fase I aqui desenvolvido pode ser utilizado para problemas de Multifluxo em geral, isto é, problemas com função objetivo linear ou não linear. O algoritmo desenvolvido foi escrito em linguagem C e implementado numa rede de microcomputadores, usando o sistema operacional UNIX. Além dos testes computacionais, apresentamos uma análise da eficiência do algoritmo e do seu *speedup*.

Palavras chaves: rede de computadores, algoritmo paralelo, problema de multifluxo, roteamento de dados.

Abstract

In this thesis a parallel algorithm is presented to find a feasible initial solution for the routing problem. The optimal routing in packet-switched networks consists of minimizing the medium delay time in the transmission of messages. This problem belongs to the class of multicommodity network flow problems.

The developed algorithm can be used to solve multicommodity network flow problems with linear or nonlinear objective function. It solves, in fast way, problems of great size.

The algorithm was written in C language and implemented in the computers network. The operating system UNIX was used. They are presented experimental results, and an analysis of the efficiency and the speedup.

Capítulo 1

Introdução

Os problemas de fluxos e multifluxos em redes têm numerosas aplicações práticas. Trata-se então de uma área de pesquisa bastante fecunda, principalmente devido ao interesse que representa o melhoramento da gestão das grandes redes de serviços tais como: distribuição de água ou gás, planejamento de redes telefônicas e elétricas, controle de tráfego aéreo, roteamento de dados, geração de energia elétrica, alguns problemas de finanças; dentre outros. Com o desenvolvimento de novos meios de comunicação, este problema teve seu interesse aumentado, sobretudo na gestão de redes telefônicas e redes de computadores (roteamento de dados). Em particular, nesta proposta de projeto trataremos apenas do problema de roteamento de dados.

A área dedicada ao desenvolvimento de algoritmos para resolução do problema de roteamento ótimo é intensa [Authie, 1987; Bertsekas, Gafni, 1983; Bertsekas, Gallager, 1987; Frata et al, 1973; Gallager, 1977; Goldfarb, 1997; Mcbride, 1998; Ribeiro, Baz, 1992; Rockfellar, 1984; Schwartz, Cheng, 1976; Sokkalingam et al, 1997; Stern, 1977; Tarjan, 1997; Tsai et al, 1989; Tsitsiklis, Bertsekas, 1986]. Schwartz e Cheng [Schwartz, Cheng, 1976], por exemplo, adaptaram o método do gradiente projetado para resolução deste problema. Stern [Stern, 1977] propôs um método dual de descentralização dos cálculos utilizando um procedimento de relaxamento para cada nó existente na rede. Mcbride [Mcbride, 1998] descreve os recentes progressos sobre a redução do tempo computacional para resolver o problema de multifluxo com grande número de restrições.

Como Mcbride [Mcbride, 1998] descreveu, o tempo computacional, em problemas de redes, é um dos principais fatores de estudo. Muitos pesquisadores trabalham para a redução

deste fator [Armstrong, Jin, 1997; Orlin, 1997], pois com esta redução o problema de roteamento de dados se tornará mais eficiente.

O problema de roteamento de dados em rede de computadores consiste em minimizar o tempo médio de atraso na transmissão de mensagens, escolhendo para elas um caminho ótimo, através dos arcos da rede. Em seu trabalho, Luvezute [Luvezute, 1995] propôs um algoritmo primal de relaxamento para otimizar o problema de roteamento de dados. O algoritmo proposto por Luvezute resolve iterativamente o problema de multifluxo, decompondo-o da forma mais independente possível, em subproblemas de simples fluxo, sendo um subproblema para cada mensagem. Esta independência entre os cálculos permite que a resolução dos subproblemas seja simultânea, admitindo-se assim uma implementação em paralelo.

O algoritmo proposto por Luvezute teve sua implementação em paralelo realizada com sucesso por Hernandes [Hernandes, 1999; Ribeiro et al, 2000a, 2000b]. A grande dificuldade na utilização do algoritmo paralelo de Luvezute é encontrar uma solução inicial factível. Esta dificuldade aumenta a medida em que aumenta o tamanho das redes que se deseja resolver.

O objetivo deste trabalho é desenvolver um algoritmo que seja capaz de encontrar uma solução inicial factível para o problema de roteamento de dados

Nesta dissertação apresentamos um algoritmo paralelo, do tipo Fase I para encontrar uma solução inicial factível para o problema de multifluxo. Este algoritmo permite resolver de maneira mais rápida os problemas de grande porte que é o nosso objetivo inicial. O algoritmo de Fase I aqui desenvolvido pode ser utilizado para problemas de Multifluxo em geral, isto é, problemas com função objetivo linear ou não linear.

A utilização da computação paralela permite que problemas grandes sejam resolvidos em tempo computacional razoável. Os problemas de grande porte, muitas vezes possuem um tempo de resolução muito grande tornando assim, inviável a sua solução utilizando máquinas seqüenciais. Para algumas classes de problemas, os algoritmos seqüenciais funcionam bem para problemas pequenos, mas perdem desempenho quando se aumenta o tamanho dos problemas. Os problemas de multifluxo pertencem a essa classe de problemas.

A programação paralela consiste na execução simultânea (concorrente) de vários programas visando à solução de um problema. O problema que tratamos neste projeto é de

grande porte, e por isto, ideal para ser tratado em paralelo. A computação paralela tem várias vantagens tais como:

- redução do tempo computacional;
- possibilidade de se resolver problemas maiores;
- relação custo/benefício mais barata;
- desempenho escalonável: o desempenho do programa melhora se ele for executado em mais processadores;
- tolerância à falhas.

A necessidade de se desenvolver programas paralelos advém do fato que não existem ferramentas de auxílio ao desenvolvimento de programas paralelos que identifiquem automaticamente o paralelismo das aplicações genéricas ou gerem código eficiente para as mesmas. Portanto, o programador deve encontrar o paralelismo na aplicação e implementar o código de forma eficiente para explorar esse paralelismo.

Esta dissertação está organizada da seguinte forma: no Capítulo 2, são apresentados alguns conceitos da teoria dos grafos, que são essenciais para a compreensão do trabalho. No Capítulo 3 são apresentados os principais problemas de fluxo e multifluxo. No Capítulos 4 e 5 são apresentados respectivamente o problema de roteamento de dados e o algoritmo paralelo de relaxamento para o Problema de Roteamento de Dados. No Capítulo 6, é apresentado o método de Fase I paralelo desenvolvido para encontrar uma solução factível inicial para o problema de multifluxo. No Capítulo 7 é feita uma introdução à computação paralela e distribuída. No Capítulo 8 são apresentados alguns aspectos relativos à implementação do algoritmo e os resultados computacionais obtidos. Por último são apresentadas a Conclusão e perspectivas de trabalhos futuros.

Capítulo 2

Conceitos Gerais

2.1 Introdução

O modelo de grafos é muito utilizado para representar problemas nas áreas de engenharia, física, química, etc. Sua grande utilização vem, sobretudo, do poder visual de sua representação e de suas propriedades específicas. Um dos exemplos mais clássicos da utilização de grafos é a representação do problema de fluxos em redes. A representação deste problema através de um grafo facilita enormemente sua resolução devido às propriedades especiais que os grafos possuem. Com base nestas propriedades foi possível desenvolver algoritmos específicos para os problemas de rede.

Apresentaremos a seguir alguns conceitos, definições e propriedades da teoria dos grafos que são essenciais para a compreensão dos algoritmos que apresentaremos nos capítulos subsequentes.

2.2 Histórico

Na antiga Prússia, tinha uma cidade chamada Königsberg. Nesta cidade existia um parque com sete pontes que ligavam duas ilhas ao continente (veja figura 2.1). Havia uma brincadeira entre os moradores dessa cidade que consistia em saber se era possível percorrer as setes pontes do parque, uma única vez.

Em 1736, Euler, então morador da cidade, criou um modelo matemático para a resolução deste problema, que passou a ser conhecido como “problema das pontes de Königsberg”. O modelo matemático de Euler foi publicado num artigo intitulado *Solutio problematis ad geometriam situs pertinentis* e consistiu no primeiro artigo sobre a teoria dos grafos. Um século mais tarde (1847), Kirchhoff utilizou grafos em seus estudos de circuitos elétricos. A partir daí, esta teoria passou a ser utilizada em outras áreas como a economia, a química, etc. Mas o desenvolvimento da teoria dos grafos só ocorreu a partir de 1946, devido a sua aplicação em problemas de pesquisa operacional. Em 1958, Claude Berge apresentou uma síntese de toda a teoria, até então desenvolvida, no seu livro *La théorie des graphes et ses applications*.

Com o desenvolvimento da informática, a partir dos anos 60, as pesquisas, bem como as aplicações na área da teoria dos grafos aumentaram enormemente.

Figura 2.1 - Ponte de Königsberg

2.3 Definições

Um grafo orientado $G=(N,L)$ é formado por um conjunto $N=\{1,2,\dots,NN\}$ de nós, onde NN é o número de nós, e um conjunto $L=\{1,2,\dots,NA\}$ de arcos, cujos elementos $l \in L$ são pares ordenados de nós e NA é o número de arcos do grafo.

Graficamente os nós podem ser representados por círculos e os arcos por segmentos de retas. A figura 2.2 abaixo apresenta um exemplo de grafo.

sendo:

$$N = \{1, 2, 3, 4, 5, 6\} \text{ e } L = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9, l_{10}\}$$

$$NN = 6 \text{ e } NA = 10$$

Figura 2.2 - Exemplo de grafo G

Como se está considerando grafos orientados, os arcos podem ser definidos da seguinte forma: chama-se nó origem, representado por $O(j)$, o nó do qual o arco l_j sai e chama-se nó destino, representado por $D(j)$, o nó no qual o arco l_j entra. Assim, pode-se representar o arco l_j pelo par ordenado $l_j = (O(j), D(j))$, é assumido que $O(j) \neq D(j)$.

Um grafo orientado também pode ser descrito pela sua matriz de incidência nó-arco. A matriz de incidência nó arco de um grafo $G=(N,L)$ é uma matriz $A(NN \times NA)$ cujos elementos A_{ij} são 0, +1, -1 e cada coluna de A corresponde a um arco de G , ou seja:

$$A_{ij} = \begin{cases} 1 & \text{se o arco } l_j \text{ sai do nó } i \\ -1 & \text{se o arco } l_j \text{ entra no nó } i \\ 0 & \text{caso contrário} \end{cases}$$

No exemplo da figura 2.2 a matriz de incidência nó-arco associado ao grafo é dada por:

$$A = \begin{matrix} & l_1 & l_2 & l_3 & l_4 & l_5 & l_6 & l_7 & l_8 & l_9 & l_{10} \\ \begin{matrix} 1 \\ 2 \\ A=3 \\ 4 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix} \end{matrix}$$

Observe que a j -ésima coluna da matriz A pode ser dada por: $A(j) = e^{O(j)} - e^{D(j)}$, onde e^k , $k \in \{O(j), D(j)\}$, é a k -ésima coluna da matriz identidade, de ordem NN . Desta forma, na matriz do exemplo anterior, temos que $A(1) = e^{O(l_1)} - e^{D(l_1)} = e^1 - e^2$. Então:

$$A(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

A seguir serão apresentadas algumas definições da teoria dos grafos:

Grafo próprio: Um grafo $G=(N,L)$ é chamado de próprio se ele possui no mínimo dois nós e um arco, isto é, se $NN \geq 2$ e $NA \geq 1$. As figuras 2.3 e 2.4 a seguir exemplificam esta definição.

Figura 2.3 - Exemplos de grafos próprios

Figura 2.4 - Exemplos de grafos que não são próprios

Neste estudo são utilizados somente grafos próprios, então para simplificar a notação, toda vez que se falar em grafo, estará implícito que se trata de um grafo próprio.

Subgrafo: Um grafo $\hat{G} = (\hat{N}, \hat{L})$ é chamado de subgrafo de G , quando $\hat{N} \subset N$ e $\hat{L} \subset L$.

A figura 2.5, mostrada a seguir, fornece um subgrafo do grafo da figura 2.2

Figura 2.5 - Subgrafo do Grafo G da Figura 2.2

Subgrafo Gerador: Um grafo $\hat{G}$ é um subgrafo gerador quando, $\hat{G}$ é um subgrafo de G e $\hat{N} = N$, $\hat{L} \subset L$.

Figura 2.6 - Subgrafo gerador do grafo G da Figura 2.2

Caminho: Seja $P = \{n_1, l_{j1}, n_2, l_{j2}, \dots, n_q, l_{jq}, n_{q+1}\}$ uma seqüência finita com pelo menos um arco. Esta seqüência é denominada um caminho no grafo G , se seus elementos ímpares são nós distintos de G , seus elementos pares são arcos de G e para qualquer arco l_{ji} de P , temos $l_{ji} \subset \{(n_i, n_{i+1}), (n_{i+1}, n_i)\}$.

Figura 2.7 - Exemplo de um caminho no grafo G da Figura 2.2

Ciclo: Uma seqüência finita $C = \{n_1, l_{j1}, n_2, l_{j2}, \dots, n_q, l_{jq}, n_{q+1}\}$ é um ciclo no grafo G , se a subsequência $\{n_1, l_{j1}, n_2, l_{j2}, \dots, n_q\}$ é um caminho no grafo G , $l_{jq} \in \{(n_q, n_{q+1}), (n_{q+1}, n_q)\}$, $n_1 = n_{q+1}$ e $l_{j1} = l_{jq}$. A figura 2.8 a seguir mostra um ciclo no grafo da figura 2.2.

Figura 2.8 - Exemplo de um ciclo no grafo G da Figura 2.2

Cociclo: Seja um conjunto de nós U , com $U \subset N$. Seja $w^+(U)$ o conjunto de arcos incidentes à U com origem em U e destino em $N-U$. Seja $w^-(U)$ o conjunto de arcos incidentes à U com destino em U e origem em $N-U$. Chamamos de cociclo de U o conjunto dos arcos γ , de modo que $\gamma = w^+(U) \cup w^-(U)$.

Grafo Acíclico: um grafo G é chamado acíclico se não é possível formar ciclos com qualquer subsequência composta com elementos de N e L .

Figura 2.9 - Exemplo de grafo acíclico

Grafo conexo: Um grafo G é conexo quando para todo par de nós, distintos, i e j de N , existe pelo menos um caminho, formado com elementos de N e L , ligando i à j .

Figura 2.10 - Exemplo de grafo conexo

Árvore(Γ): Árvore é um grafo conexo e sem ciclo, ou seja, para todo par de nós i e j distintos de N , existe um único caminho ligando i à j .

Figura 2.11 - Exemplo de árvore

Co-Árvore: Dados um grafo G e uma árvore Γ de G , co-árvore é um subgrafo de G , cujos nós são os mesmos de G e os arcos são todos os arcos de G que não pertencem à árvore Γ .

Comprimento de um caminho ou ciclo: O comprimento de um caminho ou ciclo é determinado pela quantidade de arcos que pertencem a este caminho ou ciclo.

Seqüência orientada ($S(P)$): Uma seqüência orientada, para qualquer caminho (P) com comprimento q é definida da seguinte forma:

$$S_i(P) = \begin{cases} 1 & \text{se o arco } l_{ji} = (n_{i+1}, n_i) \\ -1 & \text{se o arco } l_{ji} = (n_i, n_{i+1}) \end{cases}$$

para $i = \{1, 2, \dots, q\}$.

Esta definição vale também para ciclo (C). A figura 2.12, a seguir, nos fornece um exemplo de seqüência orientada, no caso de um ciclo.

Para o ciclo $C = \{1, a_1, 2, a_5, \dots\}$ da figura 2.12, escolhendo o sentido horário de orientação, a seqüência orientada é dada por: $S(C) = \{1, 1, 1, -1, -1, -1\}$.

$$C = \{1, l_1, 2, l_3, 4, l_9, 6, l_{10}, 5, l_6, 3, l_2, 1\}$$

$$S(C) = \{1, 1, 1, -1, -1, -1\}$$

Figura 2.12 - Exemplo de sequência orientada

Grau($g(i)$): O grau de um nó i é o número de arcos incidentes no nó (i), isto é, o número de arcos que entram ou saem de um nó.

Nó final: Um nó de uma árvore é chamado nó final quando este nó possui somente um arco incidente, isto é, o nó cujo grau $g(i)$ é igual a 1.

Árvore geradora de g : Árvore geradora de G é a árvore que é um subgrafo gerador do grafo G .

Figura 2.13 - Exemplo de árvore geradora de G

A seguir apresentamos algumas das proposições da teoria dos grafos que são necessárias ao desenvolvimento do método simplex:

Proposição 2.1: Seja a sequência finita $P = \{n_1, l_{j_1}, n_2, l_{j_2}, \dots, n_q, l_{j_q}, n_{q+1}\}$ um caminho no grafo G e A sua matriz de incidência nó-arco. Então:

$$\sum_{i=1}^q S_i(P)A(j) = e^{n_1} - e^{n_{q+1}}$$

onde:

$A(j_i)$ é a coluna da matriz A relativa ao arco l_{ji} ;

e^{n_i} é um vetor unitário que possui 1 na posição n_i e zero nas demais posições.

Proposição 2.2: Seja $C = \{n_1, l_{j1}, n_2, l_{j2}, \dots, n_q, l_{jp}, n_{q+1}\}$ um ciclo em G e A sua matriz de incidência nó-arco. Então:

$$\sum_{i=1}^q S_i(C)A(j_i) = 0$$

Proposição 2.3: Para um grafo $\Gamma=(N,L)$, com no mínimo um nó, as seguintes afirmações são equivalentes:

1. Γ é uma árvore;
2. para todo par de nós distintos (p, t) de N , existe um único caminho em Γ , unindo p a t ;
3. Γ é conexo e possui o número de arcos igual ao número de nós menos um;
4. Γ é acíclico e possui o número de arcos igual ao número de nós menos um.

Proposição 2.4: Toda árvore tem no mínimo dois nós finais.

Proposição 2.5: Seja $\Gamma=(N,L)$ uma árvore com pelo menos dois nós, cujo nó i é um nó final de Γ e l_j o arco incidente em i . Retirando de Γ o nó final i e o arco l_j , isto é, $\hat{N} = N - \{i\}$ e $\hat{L} = L - \{l_j\}$, então $\hat{\Gamma}=(\hat{N}, \hat{L})$ é também uma árvore.

Proposição 2.6: Para todo grafo G , com a correspondente matriz de incidência nó-arco A , existe uma árvore Γ geradora de G , cujas colunas de A , referentes aos arcos de Γ , são linearmente independentes.

Proposição 2.7: Seja A ; a matriz de incidência nó-arco de um grafo G , conexo, com NN nós. Então o posto de A é igual a $(NN - 1)$.

2.4 Considerações Finais

Neste capítulo foram apresentadas algumas definições e proposições da teoria dos grafos. Apesar de se tratar de conceitos clássicos que podem ser encontrados em diversos livros da área, tais como: [Bertsekas, Gafni, 1983; Gondran, Minoux, 1984; Sakarovitch, 1894; Szwarcfiter, 1986], esta apresentação é necessária para se ter uma melhor compreensão dos tópicos que serão apresentados nos demais capítulos.

Capítulo 3

Problemas de Fluxos e Multifluxos

Os problemas de fluxos em redes têm várias aplicações nas áreas de engenharia, física e matemática. Pode se fazer uma ligação direta entre os problemas de fluxos e os problemas de escoamento de gás, água, esgoto e energia. Atualmente problemas tais como: transporte, controle de estoque, redes de comunicação e tráfego de veículos também são modelados como problemas de fluxos em redes. Neste capítulo fazemos uma breve apresentação do modelo de fluxo em redes, bem como de alguns tipos de problemas de redes encontrados na literatura. Abordaremos também os problemas de multifluxos, pois são os que iremos tratar neste projeto.

3.1 Modelo de fluxo em redes

Uma rede pode ser modelada como um grafo direcionado, cujos arcos representam, por exemplo, estradas, fios, dutos etc, enfim um meio de transporte de produtos, e os nós representam os pontos terminais, tais como, cidades, armazéns, etc que são conectados entre si pelos arcos.

O problema de fluxo em redes consiste em determinar a quantidade de fluxo (produto) que deve circular pelos arcos da rede de modo a otimizar uma função objetivo. Esta função objetivo pode representar a obtenção de um fluxo máximo, um fluxo compatível ou fluxo de custo mínimo. Podemos sempre fazer com que os problemas de fluxo, independentemente da função objetivo a ser utilizada, recaiam no caso mais clássico que é o Problema de Fluxo de Custo Mínimo (PFCM) veja, por exemplo, M. Gondran e M. Sakarovith [Gondran, Minoux, 1984; Sakarovitch, 1982; Sakarovitch, 1984]. O PFCM consiste em descobrir qual o caminho

entre dois pontos de uma rede, o fluxo deve percorrer, de modo que o custo de circulação seja mínimo. O problema de fluxo de custo mínimo pode ser escrito como mostrado a seguir:

$$\begin{cases} \text{Minimizar } z_l x_l \\ \text{sujeito a :} \\ Ax = b \\ 0 \leq x \leq C_l \end{cases}$$

onde:

z_l : custo unitário do fluxo que percorre o arco l ;

x_l : quantidade de fluxo que passa pelo arco l ;

C_l : fluxo máximo permitido ou capacidade do arco l ;

b_i : demanda do nó i ;

se

- $b_i > 0$, o nó i é denominado nó fonte;
- $b_i < 0$, o nó i é denominado nó sumidouro;
- $b_i = 0$, o nó i é denominado nó de passagem.

A : matriz de incidência nó-arco.

As restrições do tipo $Ax = b$ são chamadas de restrição de conservação de fluxo, pois a quantidade de fluxo que chega num nó deve ser igual a quantidade de fluxo que sai. Assim, para cada nó existe uma restrição que deve estar em equilíbrio. As variáveis do tipo x_l estão associadas aos arcos de modo que cada arco tem uma variável x_l .

A seguir descreveremos alguns tipos de problemas de fluxos.

3.2 Principais problemas de fluxo

Na literatura os principais tipos de problemas de fluxos encontrados são:

Problema do caminho mínimo

Dentre os problemas mais antigos da teoria de grafos, estão os problemas de caminho. O problema de travessia das pontes de Königsberg foi o primeiro problema de caminho modelado e resolvido como um grafo, por Euler em 1736. Existem outros problemas tão antigos quanto este, como por exemplo, o problema da travessia do lobo, do cabrito e do repolho num pequeno barco. Neste problema o barqueiro pode transportar apenas um de cada vez, e tem que evitar que, na sua ausência, o lobo coma o cabrito ou o cabrito coma o repolho.

O problema do caminho mínimo é um dos problemas mais típicos dentre os problemas de grafos, e consiste em:

Seja $G=(N,L)$ uma rede cujos arcos l possuem comprimento d_l . O problema do caminho mínimo consiste em encontrar qual o caminho P que liga o nó s_m ao nó t_m de modo que o comprimento total do caminho seja mínimo.

Problema do fluxo máximo:

Este problema consiste em encontrar o maior fluxo (fluxo máximo) que pode circular numa rede com entrada em um nó, chamado nó fonte, e com saída em outro nó, chamado nó sumidouro. Neste problema a capacidade do arco é o único parâmetro relevante. Todas as demandas são iguais a zero e todos os arcos têm custo zero, exceto o arco que liga o nó fonte ao nó sumidouro, que possui custo unitário.

Problema de transporte:

Neste problema a rede possui somente nós fonte e sumidouro, isto é, em todos os nós tem sempre um fluxo que entra ou sai do nó. No problema não existem nós de passagem, então os nós podem ser particionados em dois conjuntos distintos, N_1 e N_2 , representando respectivamente o conjunto de nós fonte e nós sumidouro. Deste modo todos os arcos têm origem em um nó que pertence ao conjunto N_1 e destino em um nó que pertence ao conjunto

N_2 . O grafo representativo da rede é um grafo bipartido, os arcos têm capacidade infinita e as demandas são todas diferentes de zero ($b_l \neq 0$, para $l \in N$) e $\sum_{l=1}^{NN} b_l = 0$.

Problema de transporte generalizado (*Transshipment Problem*):

No problema de transporte o grafo é bipartido e os nós do grafo são fonte ou são nós sumidouro. No problema do transporte generalizado o grafo não é bipartido e existem nós que são apenas nós de passagem, isto é, não são nem fonte nem sumidouro. Todos os demais parâmetros deste problema são idênticos ao problema de transporte, por esta razão ele é considerado como caso geral do problema de transporte.

Problema de designação:

O problema da designação é um caso particular do problema de transporte onde o número de nós fonte é igual ao número de nós sumidouro ($N_1 = N_2$) e todas as demandas b_l ($l \in N$) são iguais a 1 ou -1.

Este problema também é conhecido pelo nome de *matching* de cardinalidade N_1 , num grafo bipartido, veja por exemplo, em Gondran [Gondran, Minoux, 1984].

Um exemplo típico deste problema é a designação de n tarefas à n máquinas. É suposto conhecido o custo z_{ij} de se executar a tarefa t_i pela máquina m_j . Se a tarefa t_i não puder ser executada pela máquina m_j , o custo z_{ij} é infinito. Procura-se então a designação que conduz ao menor custo total.

Como já foi dito anteriormente, há várias décadas os problemas de fluxos em redes têm sido objeto de estudo devido a sua grande aplicação prática. Existem então, na literatura vários algoritmos específicos para resolvê-los. Estes algoritmos sempre procuram explorar as particularidades próprias de cada caso. Podemos citar como exemplo disso, o Problema de Fluxo de custo mínimo (PCFM) com função objetivo linear. O PCFM pode ser resolvido através de qualquer método de programação linear, mas, por se tratar de um modelo de grafo, a matriz de restrições do PCFM possui características especiais: em cada coluna ela apresenta apenas dois elementos não nulos (1, -1). Isto faz com que a matriz de restrição seja totalmente unimodular.

Estas particularidades permitiram o desenvolvimento de uma especialização do algoritmo simplex, para redes, que agiliza o processo de resolução e soluciona problemas relativamente grandes.

3.3 Problemas de multifluxo

O problema de multifluxo ou multi-produto é um modelo natural para as redes de comunicação, como por exemplo: redes telefônicas ou de computadores. Ele consiste em fazer circular numa rede, vários fluxos (produtos) independentes, que podem ser justapostos mas não misturáveis. Se a função objetivo a ser minimizada for linear, tem-se o problema de multifluxo linear, se a função objetivo for não linear, tem-se o problema de multifluxo não linear.

O algoritmo geral da programação linear pode ser utilizado para resolver o problema de multifluxo linear, mas o número de variáveis seria muito grande, tornando este procedimento pouco eficiente. Por esta razão procura-se utilizar as características particulares de cada problema para se desenvolver algoritmos mais eficientes, e de fácil implementação.

Existem dois métodos clássicos para a resolução do problema de multifluxo linear: o método do particionamento e o método da decomposição. Uma descrição detalhada destes métodos pode ser encontrada em: [Bazaraa, Jarvis, 1977; Gondran, Minoux, 1984; Hu, 1969; Rothschild, 1966; Sakarovitch, 1973].

Formulação matemática do problema de multifluxo

Considere um grafo orientado $G = (N, L)$ com $N = \{1, 2, \dots, NN\}$ e $L = \{1, 2, \dots, NA\}$, onde NN é o número de nós e NA é o número de arcos. Um arco l também pode ser escrito pelo par ordenado $l = (O(l), D(l))$ para $l = \{1, 2, \dots, NA\}$. Considere que circulem pelo grafo G M fluxos independentes $x^1, x^2, \dots, x^M$ com componentes não-negativas ($x^m \geq 0, \forall l, \forall m$).

Para $m = \{1, 2, \dots, M\}$, $x^m = (x_l^m)$ é um fluxo simples entre um certo nó fonte $s_m \in N$ e um nó sumidouro $t_m \in N$. Em outras palavras, $Ax^m = b^m$, onde A é a matriz de incidência nó-

arco de G e b^m é um N -vetor de demanda dos produtos cujos componentes relativos aos nós s_m e t_m , tomam valores b^m e $-b^m$ respectivamente. A soma de todos os fluxos simples (produtos) que se direcionem num mesmo arco, deve ser menor ou igual a capacidade do arco (C_l), ou seja:

$$X_l = \sum_{m=1}^M x_l^m \leq C_l .$$

Um multifluxo é um conjunto de fluxos nos arcos, dado pelo vetor $X = (X_l)$, que satisfaz as equações de conservação de fluxo e as restrições de não-negatividade.

Formulação nó-arco do problema de multifluxo

Existem dois tipos de formulação matemática para o problema de multifluxo: *formulação nó-arco* e *formulação arco-caminho*. Apresentaremos a seguir a *formulação nó-arco*. Este nome advém do fato de se utilizar a matriz de incidência nó-arco do grafo.

$$\left\{ \begin{array}{l} \text{Minimizar } \sum_{m=1}^M z^m x^m \\ \text{sujeito a :} \\ \quad Ax^m = b^m \quad m = 1, \dots, M \\ \quad \sum_{m=1}^M x_l^m \leq C_l \quad \forall l \in L \text{ e } m = 1, \dots, M \\ \quad 0 \leq x^m \leq u^m \quad m = 1, \dots, M \end{array} \right.$$

onde:

z_l^m : custo unitário do produto m no arco l ;

$$z^m = [z_1^m, z_2^m, \dots, z_{NA}^m];$$

x_l^m : fluxo do produto m no arco l ;

$$x^m = [x_1^m, x_2^m, \dots, x_{NA}^m];$$

A : matriz de incidência nó-arco;

b_l^m : demandado do nó l , relativo ao produto m ;

$$b^m = [b_1^m, b_2^m, \dots, b_{NN}^m]^T;$$

C_l : capacidade do arco l ;

u^m : fluxo máximo do produto m no arco l .

Gostaríamos de chamar a atenção para a diferença fundamental entre o problema de multifluxo e fluxo simples. No caso do simples fluxo, todos os fluxos que circulam na rede são da mesma natureza (por exemplo, corrente elétrica), logo, em cada arco, eles podem ser somados algebricamente. Já no caso do multifluxo, os fluxos (produtos) são de natureza diferente e, portanto não podem ser somados algebricamente.

Formulação arco-caminho do problema de multifluxo

A formulação arco-caminho utiliza a matriz de incidência arco-caminho, daí o nome da formulação.

Seja $p(m)$ o número de caminhos elementares entre s_m e t_m em G , e Seja Q_j^m , $j = 1, \dots, p(m)$, o j -ésimo caminho da lista e seja P_j^m o vetor característico, dado por:

$$(P_j^m)_l = \begin{cases} 1, & \text{se } l \in Q_j^m \\ 0, & \text{caso contrário} \end{cases}$$

Finalmente denotamos por $x_j^m \geq 0$ a quantidade de fluxo relativa ao produto m circulando no caminho Q_j^m .

Assim, a nova formulação do problema de multifluxo se escreve como a seguir:

$$\left\{ \begin{array}{l} \text{Minimizar} \quad \sum_{m=1}^M \sum_{j=1}^{p(m)} z_j^m x_j^m \\ \text{sujeito a :} \\ \quad \sum_{m=1}^M \sum_{j=1}^{p(m)} P_j^m x_j^m \leq C \\ \quad \sum_{j=1}^{p(m)} x_j^m = b^m \quad m = 1, \dots, M \\ \quad x_j^m \geq 0 \quad \forall m \text{ e } \forall j = 1, \dots, p(m) \end{array} \right.$$

onde:

$C = (c_1, c_2, \dots, c_L)^t$ é o vetor capacidade do arco;

z_j^m é o custo do caminho Q_j^m .

Embora os problemas de multifluxo não possuam uma estrutura ideal como a dos problemas de simples fluxo, tais como: matriz de incidência totalmente unimodular, validade do teorema do corte mínimo, dentre outros; eles possuem uma estrutura que permite a aplicação de técnicas de decomposição, ver: [Bazaraa, Jarvis, 1977; Gondran, Minoux, 1984; Hu, 1969; Rothschild, 1966; Sakarovitch, 1973].

Observe que as duas formulações apresentadas para os problemas de multifluxo (nó-arco e nó-caminho) recaem num problema de programação linear. É evidente, então, que estes problemas podem ser resolvido por programação linear. Contudo, a maior parte dos problemas de multifluxo é de grande porte, e conseqüentemente têm um grande número de variáveis, por isso fica difícil aplicar o método simplex.

3.4 Considerações Finais

Neste capítulo descrevemos um modelo de redes e apresentamos os principais problemas de fluxo encontrados na literatura. Para o problema de multifluxo, apresentamos os dois tipos de formulação matemática existentes: a formulação nó-arco e a formulação arco-caminho

Capítulo 4

O Problema de Roteamento de Dados

Em uma rede de comutação por pacotes, as mensagens que são enviadas de um computador a outro são segmentadas em pequenas quantidades de *bytes*, chamados pacotes (*packets*). Os pacotes são enviados individualmente pelos arcos da rede até seu destino. Quando um pacote chega no seu destino ele é remontado e as mensagens são reconstituídas. Geralmente o número de caminhos possíveis é muito grande. Um importante problema em redes de comutação por pacotes é o problema de roteamento de dados. Este problema consiste em se determinar o encaminhamento ótimo dos pacotes, através dos arcos da rede, de modo a otimizar um critério. Dos muitos critérios existentes na literatura o mais freqüentemente utilizado é o de minimizar o tempo médio de atraso na transmissão de mensagens.

Cada pacote que percorre a rede, indo de um nó origem a um nó destino, sofre um processo de espera (fila) quando passa pelos nós intermediários. Isto é, o pacote espera numa fila até que o arco possa ser liberado e ele seja enviado ao próximo nó da sua rota. O problema de roteamento é de natureza estocástica, pois as taxas de entrada e o tamanho das mensagens são variáveis aleatórias [Bertsekas, Gallager, 1987].

Kleinrock [Kleinrock, 1964] desenvolveu uma formulação matemática para o problema de roteamento de dados, através da qual, ele mostrou que sob algumas hipóteses estocásticas realistas, este pode ser formulado como um problema de multifluxo a custo convexo.

A área de pesquisa dedicada ao desenvolvimento de algoritmos para resolução do problema de roteamento ótimo é intensiva [Authie, 1987; Bertsekas, Gafni, 1983; Bertsekas,

Gallager, 1987; Frata et al, 1973; Ribeiro, Baz, 1992; Rockfellar, 1984; Schwartz, Cheng, 1976; Stern, 1977; Tsitsiklis, Bertsekas, 1986; Tsai et al, 1989]. Schwartz e Cheung, por exemplo, adaptaram o método do gradiente projetado para resolução deste problema.

Neste capítulo apresentaremos a formulação matemática do problema de roteamento de dados. Inicialmente mostraremos o problema considerando um único produto e, em seguida, faremos uma generalização para vários produtos.

4.1 Problema de Roteamento

Como já comentamos, em redes de comunicação, as mensagens que são transmitidas através dos arcos, são convertidas em pequenas quantidades de *bytes*, usualmente chamados de pacotes. Estes pacotes são enviados individualmente pelos arcos da rede até seu destino.

O problema de multifluxo clássico, em uma rede com N nós e L arcos, consiste em otimizar uma função dada, considerando as restrições de não negatividade, as restrições de conservação de fluxo em cada nó e as restrições de capacidade em cada arco, para produto que circula na rede.

Para o caso de minimização clássico temos:

$$\begin{cases} \text{Minimizar } \bar{T} \\ \text{sujeito a :} \\ \quad A^0 x = b \\ \quad x_l \geq 0, \quad l = 1, \dots, L \end{cases} \quad (4.1)$$

onde:

$T(X)$: função objetivo.

$x_l^{(m)}$: fluxo do produto m no arco l .

$x_l = (x_l^{(1)}, \dots, x_l^{(M)})'$: fluxo no arco l .

$$x = (x_1^{(1)}, x_1^{(2)}, \dots, x_1^{(M)}, x_2^{(1)}, x_2^{(2)}, \dots, x_2^{(M)}, \dots, x_L^{(1)}, x_L^{(2)}, \dots, x_L^{(M)})^T: \text{ vetor de fluxos,}$$

onde: M é o número de produtos.

$$X_l = \sum_{j=1}^M x_l^{(j)}: \text{ fluxo global no arco } l.$$

C_l : capacidade do arco l .

$A_{((N-1) \times L)}$: matriz de restrições associadas à conservação do fluxo em cada nó da rede.

$b_{(N-1) \times 1}$: vetor de demanda nos nós da rede.

Kleinrock [Kleinrock, 1964] desenvolveu um modelo matemático para o caso de roteamento dados, formulando-o como um problema de multifluxo não linear a custo convexo. Neste modelo cada par de nós origem e destino (O, D) de uma mensagem, ou conjuntos de mensagens, é considerado como um produto, ou seja, no caso de um único produto temos somente um par de nós origem-destino; no caso de dois produtos temos dois pares de nós origem-destino, e assim sucessivamente.

4.2 Formulação do problema para um único produto

Seja uma rede direcionada com N nós e L arcos com capacidade de C_l bits/segundo. Considere que as mensagens que estão na entrada são armazenadas num *buffer* e estas são transmitidas através dos arcos da rede.

Como já dissemos este problema é de natureza estocástica, pois as razões de entrada das mensagens (quantidade de mensagens que entram na rede) bem como seus respectivos comprimentos são aleatórios. Assim, se considerarmos que a chegada de mensagens possa ser aproximada pela distribuição de Poisson, com média de padrão de chegada de λ_l (mensagens/segundo) e se considerarmos também que o comprimento da mensagem possa ser aproximado por uma distribuição exponencial, cujo comprimento médio é de $1/\mu_l$ (bits/mensagem), então o tempo médio de atraso no arco l , usando o critério desenvolvido por Kleinrock, é:

$$T_l = \frac{1}{\mu_l C_l - \lambda_l} \text{ (segundos/mensagem)}$$

Por isso o tempo total de atraso em cada arco consiste no atraso de transmissão da mensagem (espera nos *buffers*) e no tempo gasto durante a própria transmissão.

Em uma rede com N nós e L arcos, temos a média total do tempo de atraso, que é a soma de atrasos em cada arco. Portanto:

$$\bar{T} = \frac{1}{\bar{b}} \sum_{l=1}^L [\lambda_l T_l + \lambda_l T'_l]$$

onde:

$\bar{b}$: número médio de mensagens por segundo, entrando na rede;

λ_l : média de chegada das mensagens (mensagens/segundo);

T_l : tempo médio de atraso na arco l ;

T'_l : tempo gasto para a transmissão de mensagens no arco l .

Deve-se encontrar um padrão médio de chegada de mensagens λ_l , para cada arco l , com um número $\bar{b}$ (mensagens/segundo) entrando no nó origem e saindo no nó destino, de maneira que $\bar{T}$ seja mínimo.

Introduzindo-se o conceito de fluxo em rede, tem-se: $x_l = \frac{\lambda_l}{\mu_l}$ representando fluxo médio de mensagens (bits/segundo) no arco l .

Para simplificar, foi assumido que todas as mensagens na rede, tem o comprimento médio igual a $1/\mu_l$. Assim, utilizando o conceito de fluxo tem-se:

$$\bar{T} = \frac{1}{\bar{b}} \sum_{l=1}^L [\lambda_l T_l + \lambda_l T'_l] = \frac{1}{\bar{b}} \sum_{l=1}^L [x_l \mu_l T_l + x_l \mu_l T'_l],$$

como:

$$T_l = \frac{1}{\mu_l C_l - \lambda_l} \text{ (segundos/mensagem),}$$

resulta:

$$\bar{T} = \frac{1}{b} \sum_{l=1}^L \left[\frac{x_l \mu_l}{\mu_l C_l - \lambda_l} + T_l x_l \mu_l \right] = \frac{1}{b} \sum_{l=1}^L \left[\frac{1}{(C_l - x_l)} + T_l \mu_l \right] x_l$$

Para uma maior simplicidade, tomou-se $\mu_l = 1$. Assim a função final a ser minimizada é:

$$\bar{T} = \frac{1}{b} \sum_{l=1}^L x_l \left[\frac{1}{(C_l - x_l)} + T_l \right],$$

onde: x_l é o fluxo no arco l .

No caso de um único produto, o problema de fluxo a ser resolvido é:

$$\begin{cases} \text{Minimizar } \bar{T} \\ \text{sujeito a :} \\ A^0 x = b \\ x_l \geq 0, \quad l = 1, \dots, L \end{cases} \quad (4.2)$$

onde:

$\bar{T}$: função objetivo definida anteriormente;

b : vetor demanda (mensagens) nos nós da rede;

A^0 : matriz de restrições (definida anteriormente) $((N-1) \times L)$;

com:

$$a_{nl}^0 = \begin{cases} 1, & \text{se o arco } l \text{ sai do nó } n; \\ -1, & \text{se o arco } l \text{ entra no nó } n; \\ 0, & \text{se o arco } l \text{ não incide no nó } n. \end{cases}$$

O conjunto dos arcos que saem (e respectivamente entram) do nó n é denotado por $A_+^0(n)$ (respectivamente $A_-^0(n)$).

4.3 Generalização para vários produtos

No caso de multifluxo (vários produtos), a função objetivo é determinada pelos fluxos globais nos arcos da rede, ou seja:

$$\bar{T}(x) = \frac{1}{b} \sum_{l=1}^L T_l(X_l),$$

onde:

$$T_l(X_l) = \begin{cases} \left[\frac{1}{(C_l - X_l)} + T'_l \right] X_l, & \text{se } 0 \leq X_l \leq C_l; \\ \infty, & \text{se } X_l \geq C_l; \end{cases}$$

X_l : fluxo global no arco l .

Sejam:

M : número de produtos;

$b_n^{(m)}$: demanda no nó n , relativo ao produto m ;

$x_l^{(m)}$: fluxo do produto m no arco l ;

$$X_l = \sum_{m=1}^M x_l^{(m)};$$

$$\bar{b} = \frac{\sum_{m=1}^M b^{(m)}}{M}.$$

(Todas as unidades podem ser tomadas como bits/segundo).

Para uma melhor simplificação, usar-se-á uma notação mais compacta:

$x^{(m)} = (x_1^{(m)}, \dots, x_L^{(m)})$: fluxo de cada arco, relativo ao produto m ;

$x = \left(x^{(1)'}, x^{(2)'}, \dots, x^{(M)'} \right)^t$: fluxos em relação a todos os produtos;

$x_l = \left(x_l^{(1)}, \dots, x_l^{(M)} \right)^t$: fluxo no arco l em relação a todos os produtos;

$X = (X_1, X_2, \dots, X_L)^t$: fluxo global em cada arco;

$b^{(m)} = \left(b_1^{(m)}, b_2^{(m)}, \dots, b_N^{(m)} \right)^t$: demanda de todos os nós relativo ao produto com $b_{D_m}^{(m)}$ excluído.

$b = \left(b^{(1)'}, b^{(2)'}, \dots, b^{(M)'} \right)^t$: demanda em todos os nós com relação a todos os produtos.

No decorrer do desenvolvimento, a quantidade de fluxo no arco l pode ser denotada de duas maneiras: x_l ou x_{nk} , quando o arco l sai do nó n e entra no nó k .

Um conjunto linearmente independente de restrições de fluxo, para cada produto m , pode ser escrito na forma:

$$A^{(m)} x^{(m)} - b^{(m)} = 0, \quad m = 1, \dots, M;$$

onde:

$A^{(m)}$: matriz de incidência nó-arco com a linha relativa ao nó destino D_m excluída.

Desse modo, o problema de multfluxo pode ser escrito como:

$$\left\{ \begin{array}{l} \text{Minimizar } \bar{T}(x) = \frac{1}{b} \sum_{l=1}^L T_l(X_l) \\ \text{sujeito a :} \\ \quad Ax = b \\ \text{com } x \geq 0, \text{ ou seja, cada componente } x_l^{(m)} \geq 0 \end{array} \right. \quad (4.3)$$

onde:

A : matriz diagonal $(A^{(1)}, A^{(2)}, \dots, A^{(M)})$;

$$X_l = \sum_{m=1}^M x_l^{(m)} : \text{fluxo global no arco } l.$$

4.4 Restrições do problema

As restrições do problema de roteamento de dados, formulado como um problema de fluxo, são de três tipos:

Restrições de não negatividade.

O fluxo não deve assumir valores negativos, portanto para cada arco da rede, cada componente $x_l^{(m)} \geq 0$, $l=1, \dots, L$; e $m=1, \dots, M$. Assim tem-se que $X_l \geq 0$.

Restrições de capacidade.

O fluxo global X_l deve ser menor que a capacidade do arco, ou seja, $X_l < C_l$, $l=1, \dots, L$;

Restrições de conservação de fluxo nos nós da rede

O fluxo que chega no nó deve ser igual ao fluxo que deixa o nó, assim, a equação de conservação de fluxo do nó i ; relativo ao produto m ; pode ser escrita do seguinte modo:

$$\sum_{n=1}^N x_{in}^{(m)} - \sum_{n=1}^N x_{ni}^{(m)} = \begin{cases} b^{(m)}, & i : \text{nó origem} \\ 0, & \text{caso contrário} \\ -b^{(m)}, & i : \text{nó destino} \end{cases}$$

Se a equação acima for generalizada para todos os nós da rede e para todos os produtos, obtém-se: $Ax = b$, onde:

A : matriz diagonal por blocos, onde cada bloco é a matriz de incidência nó-arco; relativa a um produto;

b : vetor representante de demanda em cada nó;

x : vetor de fluxo.

Em vista disso, pode-se escrever o problema de roteamento para uma rede com N nós, L arcos e M produtos, como:

$$\begin{cases} \text{minimizar} & \bar{T}(x) = \frac{1}{b} \sum_{l=1}^L \left[\frac{1}{(C_l - X_l)} + T_l' \right] X_l \\ \text{sujeito a:} & Ax = b \\ & x_l \geq 0, \quad l = 1, 2, \dots, L \end{cases} \quad (4.4)$$

onde:

$$X_l = \sum_{m=1}^M x_l^{(m)} : \text{fluxo global no arco } l;$$

$$A: \text{matriz diagonal } (A^{(1)}, A^{(2)}, \dots, A^{(M)}).$$

Na formulação do problema de roteamento proposta por Kleinrock, não é necessário considerar explicitamente as restrições de capacidade ($X_l \leq C_l$), como acontece no problema (3.4), porque elas estão presentes na função objetivo, isto é:

$$T_l(X_l) = \begin{cases} \left[\frac{1}{(C_l - X_l)} + T_l' \right] X_l, & \text{se } 0 \leq X_l \leq C_l \\ \infty, & \text{se } X_l \geq C_l \end{cases}$$

As restrições de capacidade ($X_l \leq C_l$), também conhecidas como restrições de acoplamento, dificultam a decomposição do problema de multifluxo em subproblemas menores, mas se elas não estiverem presentes, a resolução do problema fica facilitada, pois o problema pode ser facilmente decomposto em subproblemas de dimensões menores. Contudo, a não linearidade da função objetivo é um fator de dificuldade uma vez que, de modo geral, um

problema não linear é mais difícil de resolver que o problema linear. A maioria dos algoritmos existentes para se resolver problemas de multifluxo, tratam do caso linear [Gondran, Minoux, 1984; Kennington, Helgason, 1980, Sakarovitch, 1984).

4.5 Considerações Finais

Neste capítulo foi apresentado o problema de roteamento de dados. Na formulação proposta por Kleinrock [Kleinrock, 1964], viu-se que se trata de um problema de natureza estocástica com uma função não-linear. Primeiramente o problema foi formulado para um único produto e em seguida foi generalizado para vários produtos.

Capítulo 5

Algoritmo Paralelo para Roteamento de Dados

Neste capítulo será apresentado o método primal de relaxamento [Ribeiro et al, 1998; Luvezute, 1995] para a resolução do problema de roteamento de dados formulado no capítulo 4. Este método faz uma decomposição do problema de multifluxo em subproblemas de simples fluxo. Também estudaremos o método Simplex Convexo, que é utilizado para a resolução dos subproblemas.

5.1 Método de relaxamento

O método de relaxamento primal, para o problema de roteamento de dados, decompõe o problema de minimização global de multifluxo em vários subproblemas de simples fluxo. Cada subproblema de simples fluxo pode então ser resolvido de maneira independente, podendo assim ser executado em vários processadores independentes. Seja M o número de produtos que deve circular na rede de comunicação, a cada nova iteração o método resolve M subproblemas de simples fluxo, ou seja, com um só produto, que podem ser resolvidos separadamente.

Decomposição do problema

Como visto no capítulo 4, pode-se escrever o problema de roteamento de dados como mostrado a seguir:

$$\begin{cases} \text{Minimizar} & \bar{T}(x) = \frac{1}{b} \sum_{l=1}^L T_l(X_l) \\ \text{sujeito a :} & Ax = b \\ \text{com } x \geq 0, \text{ ou seja, cada componente } x_l^{(m)} \geq 0 \end{cases} \quad (5.1)$$

onde:

$$T_l(X_l) = \begin{cases} \left[\frac{1}{(C_l - X_l)} + T_l \right] X_l, & \text{se } 0 \leq X_l < C_l \\ \infty, & \text{se } X_l \geq C_l \end{cases} \quad ; \text{ tempo de atraso em cada arco;}$$

$$\bar{T}(x) = \frac{1}{b} \sum_{l=1}^L T_l(X_l) : \text{ tempo médio de atraso na transmissão das mensagens;}$$

A : matriz diagonal $(A^{(1)}, A^{(2)}, \dots, A^{(M)})$, onde:

$A^{(m)}$: matriz de incidência nó-arco $((N-1) \times L)$, com relação ao produto m , com:

$$a_{nl}^{(m)} = \begin{cases} 1, & \text{se o arco } l \text{ sai do nó } n; \\ -1, & \text{se o arco } l \text{ entra no nó } n; \\ 0, & \text{se o arco } l \text{ não incide no nó } n. \end{cases}$$

Nesta matriz, retira-se a linha relativa ao nó destino (D_m) do produto m , para que a matriz tenha *rank* completo.

$$x = (x^{(1)'}, x^{(2)'}, \dots, x^{(M)'})^t : \text{ vetor de fluxos;}$$

$$x^{(m)} = (x_1^{(m)}, x_2^{(m)}, \dots, x_L^{(m)})^t : \text{ vetor de fluxo em todos os arcos referentes ao produto } m;$$

$$b^{(m)} = (b_1^{(m)}, b_2^{(m)}, \dots, b_N^{(m)})^t : \text{ vetor de demanda de todos os nós relativos ao produto } m \text{ com a coordenada, referente ao nó destino do produto } m, \text{ excluída;}$$

$$b = (b^{(1)'}, b^{(2)'}, \dots, b^{(M)'})^t : \text{ vetor de demanda nos nós da rede, referentes a todos os produtos;}$$

$$\bar{b} = \frac{\sum_{m=1}^M b^{(m)}}{M} : \text{número médio de mensagens entrando na rede.}$$

Seja x uma solução factível inicial para o problema de roteamento de dados apresentado em (5.1).

Se fixarmos os valores de x para todos os produtos, com exceção de um produto p que se conserva relaxado, podemos reescrever o problema (5.1) em função do fluxo $x^{(p)}$, como mostrado a seguir:

$$\begin{cases} \text{Minimizar } \frac{1}{b} \sum_{l=1}^L \bar{T}_l (\bar{X}_l + x_l^{(p)}) & p=1, \dots, M, l=1, \dots, L \\ \text{sujeito a :} & \\ A^{(p)} x^{(p)} = b^{(p)} & \\ x_l^{(p)} \geq 0, & p=1, \dots, M, l=1, \dots, L \end{cases} \quad (5.2)$$

$$\text{onde: } \bar{X}_l = \sum_{m=1, m \neq p}^M x_l^{(m)}, \quad x^{(m)} = (x_1^{(m)}, \dots, x_L^{(m)})^T$$

$$T_l(\bar{X}_l + x_l^{(p)}) = \begin{cases} \left[\frac{1}{C_l - (\bar{X}_l + x_l^{(p)})} + T'_l \right] (\bar{X}_l + x_l^{(p)}), & \text{se } 0 \leq (\bar{X}_l + x_l^{(p)}) < C_l; \\ \infty, & \text{se } (\bar{X}_l + x_l^{(p)}) \geq C_l. \end{cases}$$

Como os valores de $\bar{X}_l$, $l=1, 2, \dots, L$ são pré-fixados, podemos escrever: $\bar{C}_l = C_l - \bar{X}_l$.

Então o problema (5.2) pode ser escrito somente em função de $x_l^{(p)}$, $l=1, 2, \dots, L$, $p=1, 2, \dots, M$ como mostrado a seguir:

$$\left\{ \begin{array}{l} \text{Minimizar } \frac{1}{b} \sum_{l=1}^L T_l(x_l^{(p)}), \quad p=1, \dots, M, \quad l=1, \dots, L \\ \text{sujeito a :} \\ \quad A^{(p)} x^{(p)} = b^{(p)} \\ \quad x_l^{(p)} \geq 0, \quad p=1, \dots, M, \quad l=1, \dots, L \end{array} \right. \quad (5.3)$$

onde:

$$T_l(x_l^{(p)}) = \begin{cases} \left[\frac{1}{\bar{C}_l - x_l^{(p)}} + T_l' \right] (\bar{X}_l + x_l^{(p)}), & \text{se } 0 \leq x_l^{(p)} < \bar{C}_l; \\ \infty, & \text{se } x_l^{(p)} \geq \bar{C}_l. \end{cases}$$

Observe que ao se fixar um valor factível para os demais fluxos, em todos os arcos da rede, com relação aos $(M-1)$ produtos restantes, tem-se um problema de simples fluxo cuja variável é $x^{(p)}$. Repetindo-se este processo para os $(M-1)$ demais produtos, tem-se um total de M subproblemas de um único produto, que podem ser resolvidos separadamente.

O Método

O método de relaxamento, desenvolvido por Luvezute [Luvezute, 1995], consiste em resolver o problema de roteamento de dados decomposto (problema 5.3), iterativamente, ou seja: a partir de uma solução factível inicial, resolvem-se M subproblemas de simples fluxo. O valor de x , encontrado para estes M subproblemas, servirá de solução inicial para se resolver os M subproblemas da iteração seguinte.

O processo iterativo de atualização do fluxo x pode ser realizado de duas maneiras:

(i) semelhante ao utilizado na resolução de sistemas pelo método iterativo de Jacobi, ou seja: a partir de uma solução inicial factível x , resolve-se o problema (5.3) com todos os valores de $(x_l^{(m)})$ fixos, com exceção do fluxo do produto 1, $(x_l^{(1)})$ que se conserva relaxado. A mesma coisa é feita para todos os produtos, que utiliza sempre a mesma solução inicial, isto é, resolve-se o problema (5.3) para x fixado e o produto 2 $(x_l^{(2)})$ relaxado e assim sucessivamente para os M produtos. Com os valores de x assim obtidos, inicia-se uma nova iteração. Então, para se calcular

$x^{(p)}$, na iteração $[k+1]$, usa-se todos os valores de $x^{(1)}$ até $x^{(M)}$ da iteração $[k]$. E o processo se repete até que $\|x_l^{[k+1]} - x_l^{[k]}\| \leq \varepsilon$ para o produto $m=1, \dots, M$ e para todo o arco $l=1, \dots, L$.

(ii) semelhante ao utilizado na resolução de sistemas pelo método iterativo de Gauss-Siedel, ou seja: primeiramente resolve-se o problema (5.3) relaxando-se o produto 1. A solução $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \dots, x_L^{(1)})^T$ encontrada aqui, juntamente com os valores dos demais fluxos que estão fixados, servirão como solução inicial para resolver novamente o problema (5.3) relaxando-se agora o produto 2. A seguir relaxa-se o produto 3 e resolve-se novamente o problema (5.3) utilizando os valores de $x^{(1)} = (x_1^{(1)}, \dots, x_L^{(1)})^T$ e $x^{(2)} = (x_1^{(2)}, \dots, x_L^{(2)})^T$ até agora obtidos, juntamente com os demais fluxos que estão fixados. Este processo é repetido sucessivamente para todos os produtos. Portanto, para se calcular $x^{(p)}$, na iteração $[k+1]$, são utilizados os valores de $x^{(1)}$ até $x^{(p-1)}$ calculados na iteração $[k+1]$ e os valores de $x^{(p+1)}$ até $x^{(M)}$ da iteração $[k]$. O processo iterativo é repetido até que o valor de x na iteração $[k]$ esteja suficientemente próximo do valor de x na iteração $[k+1]$, isto é, até que para todo produto $m=1, \dots, M$ e para todo o arco $l=1, \dots, L$, tenhamos $\|x_l^{[k+1]} - x_l^{[k]}\| \leq \varepsilon$.

Como em cada subproblema os arcos estão com a capacidade reduzida da soma dos fluxos que estão fixados, a factibilidade da nova solução é garantida.

Este método de relaxamento pode ser utilizado porque, como já dissemos, no problema de roteamento, as restrições de capacidade estão embutidas na função objetivo. Deste modo não é necessário trabalhar com as restrições de acoplamento $(x_l \leq C_l)$ que dificultam a decomposição do problema em subproblemas e interfere no grau de independência dos subproblemas.

Para maiores detalhes sobre problemas de multifluxo com restrições de acoplamento ver, por exemplo, em Gondran [Gondran, Minoux, 1984].

Algoritmo de relaxamento (multifluxo)

A seguir apresentaremos uma descrição sucinta do algoritmo de relaxamento para o problema de roteamento de dados.

Seja $x^{[0]}$ uma solução inicial factível.

Seja $\varepsilon > 0$ uma precisão para o critério de parada.

Solução ótima global := false.

K:=0.

Enquanto (*Solução ótima global* = false) faça:

 início

 k:=k+1.

 Para $m:=1, \dots, M$ faça:

 início

 Fixar todos os fluxos, exceto os valores de $x^{(m)}$, de acordo com os processos iterativos de atualização de fluxo descrito anteriormente.

 Minimizar o problema (5.3) para o produto m .

 fim (para).

 Se para todo $m=1, \dots, M$ e, para todo arco $l=1, \dots, L$:

$$\|x_l^{[k+1]} - x_l^{[k]}\| \leq \varepsilon, \text{ solução ótima global := true.}$$

Fim (enquanto).

Observe que este algoritmo pode ser implementado usando qualquer um dos processos iterativos de atualização de fluxo: Jacobi ou Gauss-Seidel.

O processo (ii) semelhante ao Gauss-Seidel tem a desvantagem de diminuir a independência entre os subproblemas, o que dificultaria uma implementação em paralelo do método, sobretudo no aspecto de comunicação de dados. Por outro lado, o referido processo poderia convergir mais rapidamente que o processo (i), uma vez que ele utiliza informações mais atuais. O processo (i) semelhante ao de Jacobi tem a vantagem de manter a total independência dos subproblemas. Este fato faz com que o método seja perfeito para execução em paralelo, numa máquina de memória distribuída, porque o número de comunicações entre processadores, a

cada iteração, será pequeno. Por este motivo, optou-se pela implementação paralela do método que utiliza o processo iterativo semelhante ao de Jacobi.

A etapa de minimização dos subproblemas (problema 5.3) pode ser feita por qualquer método de simples fluxo a critério convexo. Foi utilizado o método Simplex Convexo.

5.2 O método Simplex Convexo

O método Simplex Convexo [Kennington, Helgason, 1980], foi desenvolvido para resolução de problemas de fluxo em redes, com custo convexo, e se apresenta da seguinte forma:

$$\begin{cases} \text{Minimizar } \bar{T}(x) \\ \text{sujeito a:} \\ \quad Ax = b \\ \quad 0 \leq x \leq C \end{cases} \quad (5.4)$$

onde:

A : matriz de incidência nó-arco da rede ($N \times L$);

$1.b = 0$, onde: $1 = (1, \dots, 1)^T$, com dimensão ($L \times N$);

$\bar{T}(x)$: função convexa.

Este método, desenvolvido por Willard Zangwill, pode ser analisado como uma generalização do método simplex linear [Bazaraa, Jarvis, 1977].

Da programação linear temos que, para que o problema tenha solução, a matriz A de restrições, tem que ter o posto completo. A proposição 2.7 do capítulo 2, afirma que o posto de A , num problema de fluxo de custo mínimo é igual ao número de linhas menos 1. Então para que se possa aplicar o método simplex é necessário obter uma matriz A de posto completo. Uma maneira de se conseguir isto, é adicionar à matriz A uma coluna linearmente independente. Para

tanto, basta acrescentar a rede um arco l_r . Para que a solução do problema inicial não seja alterada, este arco deve ter limite superior e custo, iguais a zero. Neste caso, para se obter uma matriz de posto completo para o problema proposto (5.4) adicionamos o arco l_r e obtemos o seguinte problema:

$$\left\{ \begin{array}{l} \text{Minimizar } \bar{T}(x) \\ \text{sujeito a :} \\ \quad Ax + e^r x_r = b \\ \quad 0 \leq x \leq C \\ \quad 0 \leq x_r \leq 0 \end{array} \right. \quad (5.5)$$

onde: r : é um número inteiro, menor ou igual ao número de nós;

e^r : é o vetor unitário, com um na posição r e zero nas demais posições;

x_r : é a quantidade de fluxo que deve percorrer o arco artificial l_r .

Como x_r é restrita a zero, uma solução ótima para o problema (5.5) será também uma solução ótima para o problema (5.4).

O método

O método Simplex Convexo, do mesmo modo que o Simplex Linear, parte de uma solução inicial factível e depois procura uma direção factível de melhoria. Gostaríamos de relembrar que, uma direção d é dita factível para x , se $x + \alpha d$ é factível para algum $\alpha > 0$. Esta direção é chamada de direção de melhoria no ponto x , se d é factível para x e, a derivada direcional de $\bar{T}(\cdot)$ na direção d , sobre o ponto x , é negativa.

O Simplex Convexo, assim como no caso linear, particiona as variáveis em dois tipos: variáveis básicas (x^B) e não básicas (x^{NB}). A cada iteração, o método permite que as variáveis não básicas mudem de fluxo. Entretanto, aqui, as variáveis não básicas podem assumir outros valores que seus limitantes inferiores e superiores, ou seja, $0 \leq x^{NB} \leq C^{NB}$.

No desenvolvimento do método Simplex Convexo, é assumido que o gradiente de $\bar{T}(\cdot)$ existe sobre $\{x: 0 \leq x_l \leq C_l\}$, e que $\bar{T}(\cdot)$ é separável, ou seja:

$$\bar{T}_l(x) = \sum_l \bar{T}_l(x_l)$$

Suponha que as equações de conservação de fluxo $Ax=b$ sejam particionadas em $[B:NB]x=b$, onde B é uma base. Igualmente, seja x particionada em $[x^B:x^{NB}]$ e C em $[C^B:C^{NB}]$. Então, o problema pode ser escrito como:

$$\left\{ \begin{array}{l} \text{Minimizar } \bar{T}([x^B:x^{NB}]) \\ \text{sujeito a :} \\ x^B = B^{-1}b - B^{-1}NBx^{NB} \\ 0 \leq x^B \leq C^B \\ 0 \leq x^{NB} \leq C^{NB} \end{array} \right. \quad (5.6)$$

Substituindo x^B em (5.6), obtemos:

$$\left\{ \begin{array}{l} \text{Minimizar } f(x^{NB}) = \bar{T}[B^{-1}b - B^{-1}NBx^{NB}:x^{NB}] \\ \text{sujeito a :} \\ 0 \leq B^{-1}b - B^{-1}NBx^{NB} \leq C^B \\ 0 \leq x^{NB} \leq C^{NB} \end{array} \right. \quad (5.7)$$

Como o problema foi transformado num problema de minimização em função das variáveis não-básicas, toma-se o conjunto de direções no espaço não básico

$$D = \{e^1, e^2, \dots, e^{L-N}, -e^1, -e^2, \dots, -e^{L-N}\}$$

onde:

e^i : vetor unitário, com a i -ésima coordenada igual a um e as outras coordenadas são nulas.

Para garantir a factibilidade deve-se assegurar que $0 \leq x^{NB} \leq C^{NB}$. Assim o conjunto de direções factíveis que asseguram $0 \leq x^{NB} + \alpha d \leq C^{NB}$, é:

$$\begin{cases} e^i & \text{é direção factível se : } x_i^{NB} < C_i^{NB} \\ (-e^i) & \text{é direção factível se : } x_i^{NB} > 0 \end{cases}$$

Considerando o conceito de derivada direcional no espaço não básico, indicamos a seguir, como selecionar uma direção factível de melhoria.

A derivada direcional de $f(\cdot)$ no ponto x^{NB} , na direção d , desde que o limite exista, é definida por:

$$D_d f(x^{NB}) = \lim_{t \rightarrow 0^+} \frac{f\left(x^{NB} + t \frac{d}{\|d\|}\right) - f(x^{NB})}{t},$$

Pode-se mostrar que a derivada direcional no espaço não-básico é dada por:

$$D_d f(x^{NB}) = \frac{\nabla f(x^{NB})d}{\|d\|}$$

onde: $\nabla f(x^{NB})$ é o gradiente de $f(\cdot)$ avaliado no ponto x^{NB} .

Como $f(x^{NB}) = \bar{T}[B^{-1}b - B^{-1}NBx^{NB}; x^{NB}]$ pode-se particionar $\nabla \bar{T}$ em

$$\nabla \bar{T}(x) = [\nabla \bar{T}_B(x); \nabla \bar{T}_{NB}(x)].$$

Assim, obtém-se: $\nabla f(x^{NB}) = \nabla \bar{T}_{NB}(x) - \nabla \bar{T}_B(x)B^{-1}NB$.

Queremos que $\nabla f(x^{NB})d < 0$. Deste modo, para tomarmos uma direção de melhoria no espaço não básico, devemos escolher uma direção que satisfaça cada coordenada i :

$$\begin{cases} e^i, & \text{se } x_i^{NB} < C_i^{NB} \text{ e } \nabla \bar{T}_{NB}(x_i) - \nabla \bar{T}_B(x_i)B^{-1}NB < 0; \\ (-e^i), & \text{se } x_i^{NB} > 0 \text{ e } \nabla \bar{T}_{NB}(x_i) - \nabla \bar{T}_B(x_i)B^{-1}NB > 0. \end{cases}$$

Este conjunto representa o conjunto dos arcos que são candidatos a entrar na base. Para avaliá-lo, precisamos calcular o valor de $\nabla f(x^{NB})$ que indicará o crescimento da função objetivo.

Seja B uma base qualquer para (5.7), e seja Γ_B sua árvore correspondente.

Podemos calcular as componentes de $\nabla \bar{T}$, diretamente sobre a árvore Γ_B , como mostraremos a seguir:

Seja l_k um arco que não está na árvore Γ_B , e $P = \{n_1, l_{j_1}, n_2, l_{j_2}, \dots, n_t, l_{j_t}, n_{t+1}\}$, o caminho que liga os nós origem $O(l_k)$ e destino $D(l_k)$ na árvore Γ_B . Seja $C'(k)$ o ciclo formado por P , com adição de l_k à árvore Γ_B , e seja também, sua correspondente sequência de orientação $OR(k)$.

Pode-se determinar o conjunto dos arcos que são candidatos a entrar na base (direção de melhoria) de duas maneiras:

(i) Através da sequência de orientação $OR(k)$, isto é:

$$\bar{c}_k = \sum_{i=1}^t OR_i(k) \frac{\partial \bar{T}(x)}{\partial x_{j_i}}$$

onde:

t : número de nós no ciclo formado por $(P + \{l_k\})$;

x_{j_i} : fluxos nos arcos da árvore, correspondente aos arcos l_{j_i} .

Assim, tomando ε como uma precisão para o critério de parada, o conjunto dos arcos candidatos a entrar na base é definido como:

$$\begin{cases} l_k, & \text{se } x_k^{NB} < C_k^{NB} \text{ e } \bar{c}_k > \varepsilon; \\ l_k, & \text{se } x_k^{NB} > 0 \text{ e } \bar{c}_k < -\varepsilon. \end{cases}$$

(ii) Ou através das variáveis duais π_i (também chamadas de potenciais dos nós), isto é, calcular

$$\bar{c}_k = \pi_{O(l_k)} - \pi_{D(l_k)} - \frac{\partial \bar{T}(x)}{\partial x_{l_k}},$$

diretamente sobre a árvore, usando:

$$\begin{cases} \pi_w = 0, & \text{onde } r = \text{nó raiz de } \Gamma_B, \\ \pi_{O(l_j)} - \pi_{D(l_j)} = \frac{\partial \bar{T}(x)}{\partial x_j}, & \text{para } l_j \in \Gamma_B. \end{cases}$$

Deste modo, o conjunto dos arcos candidatos a entrar na base é definido como anteriormente.

Qualquer uma destas maneiras pode ser estendida para o método do algoritmo Simplex Convexo. Escolhemos o primeiro caso para esta representação.

Uma vez conhecida a direção de melhoria, uma busca unidimensional é requerida, para determinar o tamanho do passo a ser dado nesta direção.

A base é atualizada caso o resultado da alteração torne uma das variáveis básicas nula ou a faça atingir seu limite superior.

Agora, apresentaremos o algoritmo para o caso específico, onde $\bar{T}(X)$ é separável.

Algoritmo Simplex Convexo

(0) Inicialização

Seja $\bar{x}$ um fluxo factível e determine uma árvore Γ_B . Seja $\varepsilon > 0$ uma precisão para o critério de parada.

(1) Avaliação

Para $l_k \notin \Gamma_B$, seja $P = \{n_1, l_{j_1}, n_2, l_{j_2}, \dots, n_t, l_{j_t}, n_{t+1}\}$ o caminho em Γ_B que liga $O(l_k)$ ao $D(l_k)$.

Seja $C'(k)$ o ciclo formado por P com a adição de l_k , e seja $OR(k)$ a sequência de orientação correspondente a este ciclo.

Sejam:

- $\bar{c}_k = \sum_{i=1}^l OR_i(k) \frac{\partial \bar{T}(\bar{x})}{\partial x_{j_i}};$
- $\psi_1 = \{l_k : x_k < C_k \text{ e } \bar{c}_k > \varepsilon\};$
- $\psi_2 = \{l_k : x_k > 0 \text{ e } \bar{c}_k < -\varepsilon\}.$
- Se $\psi_1 \cup \psi_2 = \emptyset \Rightarrow \bar{x}$ é uma solução ótima para o problema.
- Senão seja: $l_k \in \psi_1 \cup \psi_2$ e $\delta = \begin{cases} +1 & \text{se } l_k \in \psi_1 \\ -1 & \text{se } l_k \in \psi_2 \end{cases}$

(2) Cálculo do bloqueio

Sejam:

- $\Delta_1 = \min_{O_i(l_k)=\delta} \{\bar{x}_{j_i}, \infty\};$
- $\Delta_2 = \min_{O_i(l_k)=\delta} \{C_{j_i} - \bar{x}_{j_i}, \infty\};$
- $\Delta = \min \{\Delta_1, \Delta_2\};$

Se $\Delta = 0 \Rightarrow$ vá para o passo 5;

(3) Atualização do fluxo

Neste passo, verificamos se a solução é factível e determinamos a magnitude do passo.

Definimos o vetor u como segue:

$$u_j = \begin{cases} OR_j(k), & \text{se } l_j \in c'(k); \\ 0, & \text{caso contrário.} \end{cases}$$

Seja:

$$\gamma = -\delta \sum_{i=1}^l OR_i(k) \left[\frac{\partial \bar{T}(\bar{x} - \Delta \delta u)}{\partial x_i} \right];$$

- Se $\gamma < 0 \Rightarrow \begin{cases} \bar{x} = \bar{x} - \Delta \delta u \\ \text{vá para o passo 5} \end{cases}$

(4) Procedimento de busca (bisseccção)

Este passo pode ser substituído por qualquer outro procedimento de Busca Unidimensional conhecido.

a) Seja: $\alpha_0 = 0$ e $\alpha_1 = \Lambda$;

b) Seja: $\alpha = \frac{(\alpha_0 + \alpha_1)}{2}$;

c) Seja: $\gamma = -\delta \sum_{i=1}^l OR_i(k) \left[\frac{\partial \bar{T}(\bar{x} - \alpha \delta u)}{\partial x_i} \right]$;

d) Se $|\gamma| < \varepsilon \Rightarrow \begin{cases} \text{Seja } \bar{x} = \bar{x} - \alpha \delta u \text{ e} \\ \text{retorne ao passo 1} \end{cases}$;

e) Se $\gamma > 0 \Rightarrow \begin{cases} \alpha_1 = \alpha \text{ e} \\ \text{retorne ao passo b} \end{cases}$;

Senão $\begin{cases} \alpha_0 = \alpha \text{ e} \\ \text{retorne ao passo b} \end{cases}$.

(5) Mudança de base

Verifique se algum fluxo x_i , no arco l_i de $C'(k)$, tornou-se nulo, ou atingiu seu limitante superior, caso isto tenha ocorrido remova o arco l_i de Γ_B e adicione o arco l_k , formando, assim, uma árvore. Retorne ao passo 1.

5.3 Considerações Finais

Neste capítulo foi apresentado o método paralelo de relaxamento, desenvolvido por Luvezute, para o problema de roteamento de dados. Viu-se que este problema pode ser decomposto em M problemas de simples fluxo, independentes, que são resolvidos em paralelo. Foi também apresentado o algoritmo simplex convexo para fluxo em redes. Este é o algoritmo utilizado para resolver os subproblemas de simples fluxo.

Capítulo 6

O Método de Fase I Paralelo

Neste capítulo será apresentado o método paralelo desenvolvido para encontrar uma solução inicial para o problema de multifluxo não linear. Deu-se o nome de Fase I paralelo a este método porque ele tem um procedimento semelhante ao método simplex de Fase I utilizado para encontrar uma solução inicial para o problema de simples fluxo linear. Para uma melhor compreensão do método paralelo, será apresentado o método simplex para simples fluxo linear e o método simplex de Fase I linear.

6.1 O Método Primal Simplex para Fluxo em Redes Linear

O método primal simplex é ainda hoje o mais eficiente para resolver os problemas de fluxos em redes com função objetivo linear. Ele é capaz de resolver em segundos problemas relativamente grandes e em minutos, problemas de dimensão considerada.

Como o algoritmo que desenvolvemos, para encontrar a solução inicial do problema de multifluxo não linear, faz uso do método primal simplex linear; mostraremos a seguir os principais passos deste método.

O método primal simplex para redes é uma especialização do método simplex, desenvolvido por Dantzig [Dantzig, 1951], para programação linear. O simplex para redes procura tirar proveito das características específicas do problema de redes, como por exemplo, a matriz básica que pode ser re-arranjada em uma matriz triangular, através da permutação de linhas e colunas. Esta transformação simplifica os cálculos dos custos relativos, das atualizações

das variáveis duais e da atualização da matriz básica, a cada iteração. Uma outra característica importante da matriz de restrições A , é que ela é totalmente unimodular, isto é, o determinante de todas as sub-matrizes quadradas tem valor 1, -1, ou 0. Esta propriedade garante que a solução do problema, se ela existir, será sempre inteira.

Formulação do Problema

Sabe-se que, no problema de fluxo em redes a matriz de restrições tem posto igual a $(N - 1)$. Então para que se possa aplicar o método simplex é necessário obter uma matriz A de posto completo. Vimos anteriormente no capítulo 5, que para se conseguir isto basta acrescentar à matriz A uma coluna linearmente independente. Então, acrescentando-se a rede um arco l_r , a formulação do problema de fluxo fica:

$$\text{NP} \begin{cases} \min cx \\ \text{s.a. } Ax + e^r x_r = b \\ 0 \leq x \leq u \\ 0 \leq x_r \leq 0 \end{cases}$$

onde:

r : é um número inteiro, menor ou igual ao número de nós;

e^r : é o vetor unitário, com um na posição r e zero nas demais posições;

x_r : é a quantidade de fluxo que deve percorrer o arco artificial l_r .

Uma vez que o fluxo em l_r será sempre zero, qualquer solução ótima do problema NP também será um ótimo do problema inicial.

O grafo associado ao problema NP é chamado de grafo enraizado, o arco l_r é chamado de arco raiz, e o nó r é chamado de nó raiz. Se $x_r \neq 0$, o problema inicial não possui solução factível. Graficamente indica-se o arco raiz como uma seta incidindo no nó raiz.

Do mesmo modo que no simplex não linear, apresentado no capítulo 5, o simplex linear faz uma partição das variáveis em básicas (x^B) e não básicas (x^{NB}) .

A proposição abaixo assegura que a matriz básica B pode ser transformada numa matriz triangular inferior.

Proposição 6.1: Seja A uma matriz de incidência nó-arco de um grafo conexo e enraizado G , com nó raiz r . Se B é uma base de $[A \mid e']$, então B é uma matriz triangular inferior.

Uma vez que a matriz básica pode ser transformada em uma matriz triangular inferior, então as variáveis duais podem ser calculadas diretamente na rede pela mesma relação apresentada no capítulo 5, isto é:

$$\begin{cases} \pi_w = 0, & \text{onde } r = \text{nó raiz de } \Gamma_B, \\ \pi_{O(l_j)} - \pi_{D(l_j)} = c_j, & \text{para } l_j \in \Gamma_B. \end{cases}$$

O conjunto dos arcos candidatos a entrar na base é calculado através dos custos relativos, fornecidos pela relação:

$$\bar{c}_k = \pi_{O(l_k)} - \pi_{D(l_k)} - c_k$$

O Método Primal Simplex para Fluxo Linear

1. Inicialização: Seja $[x^B \mid x^{NB}]$ uma solução básica factível e Γ_B uma árvore de G com nó raiz r .

Cálculo dos valores de π :

$$\begin{cases} \pi_r = 0 \\ \pi_{O(l_j)} - \pi_{D(l_j)} = c_j & \text{para } l_j \in \Gamma_B \end{cases}$$

2. Cálculo dos custos relativos: Considerar:

$$\psi_1 = \{l_j : x_j^{NB} = 0 \text{ e } (\pi_{O(j)} - \pi_{D(j)} - c_j) > 0\}$$

ou

$$\psi_2 = \{l_j : x_j^{NB} = u_j \text{ e } (\pi_{O(j)} - \pi_{D(j)} - c_j) < 0\}$$

Se $\{\psi_1 \cup \psi_2\} = \emptyset$, terminar com $x = [x^B \mid x^{NB}]$ ótimo -- FIM. Senão, seleccionar um $l_k \in \{\psi_1 \cup \psi_2\}$ e definir:

$$\delta \leftarrow \begin{cases} 1 & \text{se } l_k \in \psi_1 \\ -1 & \text{se } l_k \in \psi_2 \end{cases}$$

3. Cálculo do bloqueio: Seja $P = \{n_1, l_{j_1}, n_2, l_{j_2}, \dots, n_q, l_{j_q}, n_{q+1}\}$, representando o caminho em Γ_B , unindo $O(k)$ a $D(k)$. Calcular a sequência de orientação $S(P)$ e o valor do bloqueio Δ usando:

$$\Delta_1 \leftarrow \min_{S_i(P)=\delta} \{x_{ji}, \infty\}$$

$$\Delta_2 \leftarrow \min_{S_i(P)=\delta} \{u_{ji} - x_{ji}, \infty\}$$

$$\Delta_1 \leftarrow \min \{\Delta_1, \Delta_2, u_k\}$$

4. Atualização do fluxo: Calcular $x_k \leftarrow x_k + \Lambda\delta$ e $x_{ji} \leftarrow x_{ji} - \Lambda\delta S_i(P)$, para todo $l_{ji} \in P$.

Se $\Delta = u_k$, retornar ao passo 2.

5. Atualização da árvore: Considerar:

$$\psi_3 = \{l_{ji} : x_{ji} = 0, \text{ onde } S_i(P) = \delta\}$$

ou

$$\psi_4 = \{l_{ji} : x_{ji} = u_{ji}, \text{ onde } -S_i(P) = \delta\}$$

Selecionar um $l_m \in \{\psi_3 \cup \psi_4\}$. Trocar l_m por l_k na árvore. Atualizar as variáveis duais (π) e retornar ao passo 2.

6.2 O método simplex de Fase I

Encontrar uma solução básica inicial nem sempre é uma tarefa trivial, devido às restrições de capacidade. Existem várias técnicas na literatura para solucionar este problema, [Bazaraa, 1977; Jensen, Barnes, 1980]. A seguir apresentamos uma técnica equivalente ao método do M-grande da Programação Linear, também conhecida como Fase I.

Esta técnica consiste em acrescentar à rede um nó artificial (0) e vários arcos artificiais de acordo com as seguintes regras:

- se $b_i > 0$ então, acrescenta-se o arco $(i,0)$;
- se $b_i < 0$ então, acrescenta-se o arco $(0,i)$;
- se $b_i = 0$ não há necessidade de acrescentar um arco artificial.

O fluxo inicial factível é: $x_{(i,0)} = b_i$ e $x_{(0,i)} = -b_i$. Observe que a solução inicial possui fluxo percorrendo somente os arcos artificiais.

A função objetivo desse novo problema tem como finalidade diminuir a soma dos fluxos nos arcos artificiais. Matematicamente:

$$\min \varnothing = \sum x_{(i,0)} + \sum x_{(0,i)}$$

Observando a nova função objetivo, nota-se que todos os arcos artificiais têm custo igual a 1 e todos os arcos do grafo original têm custo zero. Como o objetivo consiste em reduzir o valor de $\varnothing$, os arcos do grafo original são candidatos a entrar na base e os arcos artificiais não retornarão à base, uma vez que isto aumentaria o valor de $\varnothing$.

Aplica-se então o algoritmo simplex até obter $\emptyset = 0$. A função objetivo $\emptyset$ é conhecida pelo nome de multa. Então, a cada iteração do método a multa vai sendo reduzida, até atingir o valor zero. Se isto não for possível, o problema inicial (NP) é infactível.

6.3 O Problema de Fase I para o multifluxo

A maioria dos métodos existentes; para a resolução de problemas de multifluxo são do tipo primal e, portanto, para serem executados necessitam de uma solução inicial factível.

O problema de encontrar uma solução factível consiste em encontrar uma solução que respeite o conjunto de restrições do problema do multifluxo, sem levar em consideração a função objetivo do problema original. Ou seja, o problema consiste em encontrar um x que satisfaça o conjunto de restrições (6.1) abaixo:

$$\begin{cases} Ax = b \\ x_l \leq C_l \\ x_l^m \geq 0 \end{cases} \quad (6.1)$$

onde:

$$x_l = \sum_{m=1}^M x_l^m \quad x = (x_1^1, x_1^2, \dots, x_1^m, \dots, x_2^1, x_2^2, \dots, x_2^m)$$

Deste modo, um algoritmo que seja capaz de encontrar uma solução factível para um multifluxo, pode ser utilizado tanto para problemas de multifluxo lineares, com para problemas de multifluxo não lineares.

Foi a partir deste raciocínio que desenvolvemos o algoritmo para encontrar uma solução inicial para o problema de roteamento de dados, que aqui chamamos de Fase I. O método que desenvolvemos pode então ser utilizado para qualquer problema de multifluxo em geral. A seguir faremos uma apresentação detalhada do método:

Seja o problema de encontrar um conjunto de soluções x , tal que:

$$\begin{cases} Ax = b \\ x_l \leq C_l \\ x_l^m \geq 0 \end{cases} \quad (6.2)$$

Inicialmente fazemos uma decomposição no conjunto de restrições (6.2) de modo a se obter conjuntos de M restrições, sendo um conjunto para cada produto.

Assim passamos a ter M subproblemas de Fase I que consistem em:

Encontrar uma solução x^p , $p = 1, 2, 3, \dots, M$ que satisfaça

$$A^{(p)} x^{(p)} = b^{(p)} \quad 0 \leq x_l^p \leq C_l$$

Como não possuímos um limitante superior de capacidade, específico para cada produto; inicialmente consideramos a capacidade global como limitante superior. Observe que se tivéssemos um limitante de capacidade específico para cada produto, teríamos M subproblemas de simples fluxo, de Fase I, totalmente independentes.

A seguir aplicamos o algoritmo Simplex linear de Fase I em cada subproblema. Observe também que a aplicação do método simplex, em cada um desses problemas, pode ser feita de modo independente. Então, se tivermos disponíveis M processadores, cada um desses subproblemas pode ser resolvido em um processador.

Com os resultados obtidos pelo método de Fase I linear, fazemos uma verificação, em todos os arcos da rede para saber quais deles tiveram seus fluxos globais (x_l) maiores a capacidade global, isto é:

$$x_l > C_l \quad x_l = \sum_{m=1}^M x_l^m$$

A estes arcos, chamamos de arcos violados. Senão existir nenhum arco violado, então a solução encontrada já é uma solução factível e o algoritmo termina.

Se existirem arcos violados, procedemos em cada subproblema um ajuste nas capacidades (C_l^m) dos arcos que estão violados e que tem fluxo maior que zero, isto é, $x_l^m > 0$. Chamamos de redes locais as redes dos subproblemas. Do mesmo modo, chamados x_l^m e C_l^m , respectivamente de fluxos locais e capacidades locais.

O ajuste dos C_l^m é feito do seguinte modo:

- Para os arcos violados com fluxos locais $x_l^m > 0$, reduzimos a capacidade local C_l^m de uma quantidade igual ao maior inteiro de $\lceil (x_l - C_l) / M \rceil$, onde x_l é o fluxo global, C_l é a capacidade global no arco l e M é o número de produtos.

Desta forma as capacidades locais para os arcos violados cujo $x_l^m > 0$ é dada por

$$C_l^m = C_l^m - \lceil (x_l - C_l) / M \rceil \quad (6.3)$$

Observe que $(x_l - C_l)$ é a quantidade de fluxo que fez com que o arco l se tornasse violado na rede global. Quando reduzimos as capacidades C_l^m das redes locais, de acordo com a equação (6.3) acima, estamos distribuindo o fluxo excedente entre todas as sub-redes, pois obrigamos os fluxos desses arcos na próxima iteração, a ficarem reduzidos.

A seguir calculamos a soma das capacidades locais $\bar{C}_l = \sum_{m=1}^M C_l^m$.

Para os arcos onde $\bar{C}_l < C_l$, fazemos uma redistribuição das capacidades que foram reduzidas anteriormente. Esta redistribuição é feita entre os arcos da rede local cuja a capacidade local é menor que a capacidade global, isto é, $C_l^m < C_l$.

O objetivo desta redistribuição é conservar a capacidade total (C_l) da rede global, ou seja, é garantir que a soma das capacidades locais não seja inferior a capacidade global, isto é, garantir que $\bar{C} = \sum_{m=1}^M C_l^m \geq C_l$.

Em seguida, para cada subproblema, executa-se novamente um simplex linear de Fase I para simples fluxo. Se todas as soluções das redes locais forem factíveis, procede-se uma nova iteração.

Se uma das redes locais se tornar inactivável devido a redução de capacidade, então aumentamos em uma unidade a capacidade dos arcos, da rede local inactivável, cujos fluxos x_l^m estão no limite superior e cuja capacidade local C_l^m seja menor que a capacidade global, isto é, para os arcos da rede inactivável onde $x_l^m = C_l^m$, com $C_l^m < C_l$; fazemos $C_l^m = C_l^m + 1$. Para manter o equilíbrio das capacidades globais C_l , no mesmo arco l que sofreu o aumento de capacidade C_l^m ; para os demais produtos, isto é, nas demais redes locais, fazemos uma redução de capacidade na mesma proporção do aumento.

A seguir executa-se novamente o simplex linear de Fase I de simples fluxo. Este procedimento se repete até que a rede local que estava inactivável se torne factível.

Depois de factibilizar a rede local, inicia-se uma nova iteração do método.

A seguir, apresentamos o método descrito acima em linguagem algorítmica (pseudocódigo).

```
{ALGORITMO MULTIFLUXO PARALELO DE FASE I}
início
  {---- PASSO 1 ----}
  para m:= 1 até M faça em paralelo
    simplexFaseI( $R^m, S^m$ ) { $R^m$  = rede local m,  $S^m$  = solução local}
  fim-para
  {---- PASSO 2 ----}
  para m:= 1 até M faça
    para l:= 1 até L faça
       $x_l \leftarrow x_l + x_l^m$ 
    fim-para
  fim-para
  {---- PASSO 3 ----}
  para l:=1 até L faça
    se  $x_l > C_l$ 
      então
        SoluçãoInactivável  $\leftarrow$  1
    fim-se
```

```

fim-para
  enquanto SoluçãoInfactível = 1 faça
    {---- PASSO 4 ----}
    se redefactível( $R^m$ ) {-Verifica se a rede local é factível-}
    então
      para  $l := 1$  até  $L$  faça
        para  $m := 1$  até  $M$  faça
          se ( $x_l > C_l$ ) E ( $x_l^m > 0$ )
             $\bar{C}_l^m \leftarrow C_l^m - \left\lceil \frac{(X_l - C_l)}{M} \right\rceil$ 
             $\bar{C}_l \leftarrow \bar{C}_l^m$ 
          fim-se
        fim-para
        se ( $\bar{C}_l < C_l$ ) E ( $\bar{C}_l > 0$ ) então
          resto  $\leftarrow C_l - \bar{C}_l$ 
          enquanto resto > 0 faça
            se ( $x_l > C_l$ ) E ( $C_l^m < C_l$ ) então
               $C_l^m \leftarrow C_l^m + 1$ 
               $\bar{C}_l \leftarrow \bar{C}_l + 1$ 
              resto  $\leftarrow$  resto - 1
            fim-se
             $m \leftarrow m + 1$ 
          fim-enquanto resto
        fim-se
      fim-para
    senão {---tem uma rede local infactível---}
      para  $l := 1$  até  $L$  faça
        se ( $x_l^m = C_l^m$ ) E ( $C_l^m < C_l$ ) então
          cont = 0
          enquanto  $C_l^m < C_l$  E cont < (M-1) faça
             $C_l^m \leftarrow C_l^m + 1$ 
            cont  $\leftarrow$  cont + 1
          fim-enquanto
        fim-se
      fim-para
    fim-se
  {---- PASSO 5 ----}
  SoluçãoInfactível  $\leftarrow$  0
  para  $m := 1$  até  $M$  faça em paralelo
    simplexFaseI( $R^m, S^m$ )
  fim-para

```

```

{----- PASSO 6 -----}
para  $m$  de 1 até  $M$  faça
  para  $l$  de 1 até  $L$  faça
     $x_l = x_l \leftarrow x_l^m$ 
  fim-para
fim-para
para  $l$  de 1 até  $L$  faça
  se  $x_l > C_l$ 
    então
       $SoluçãoInfactível \leftarrow 1$ 
    fim-se
fim-para
fim-enquanto  $SoluçãoInfactível$ 
fim

```

6.4 Considerações Finais

Neste capítulo foi apresentado o método de Fase I para encontrar uma solução factível para o problema de multifluxo. O método aqui desenvolvido pode ser usado para encontrar a solução inicial de qualquer problema de multifluxo, seja ele linear ou não linear. Isto é possível porque no problema de Fase I, trabalhamos somente com as capacidades dos arcos e elas são lineares.

Capítulo 7

Computação Paralela

A computação paralela consiste basicamente em um conjunto de elementos de processamento, que cooperam e comunicam entre si para solucionar grandes problemas, de maneira mais rápida do que se estivessem sendo solucionadas sequencialmente [Almasi, Gottlieb, 1994; Sousa, 1996].

A capacidade de aumentar o processamento em uma única máquina foi a principal razão para o surgimento da computação paralela. Suas principais vantagens são:

- Alto desempenho para programas lentos;
- Solução mais natural para problemas intrinsecamente paralelos;
- Maior tolerância à falhas.

Neste capítulo apresentamos uma visão geral da computação paralela, falando de algumas características fundamentais e abordando tópicos mais relevantes.

7.1 Motivações para o Paralelismo

Vários fatores explicam a necessidade do processamento paralelo. O principal deles trata-se da busca por maior desempenho [Almasi, Gottlieb, 1994; Zaluska, 1991]. As diversas áreas nas quais a computação se aplica, sejam científicas, industriais ou militares, requerem cada vez

mais poder computacional, em virtude dos algoritmos complexos que são utilizados e do tamanho do conjunto de dados a ser processado.

Além da busca de maior desempenho, outros fatores motivam o desenvolvimento da computação paralela [Almasi, Gottlieb, 1994]. Entre eles:

- desenvolvimento tecnológico (principalmente a tecnologia VLSI de projeto de microcomputadores), permitiu a construção de microcomputadores de alto desempenho que, agrupados, possibilitam um ganho significativo de poder computacional. Além disso, tais configurações possibilitam uma melhor relação custo/de desempenho, quando comparadas aos caros supercomputadores [Zaluska, 1991];
- Restrições físicas, como a velocidade finita da luz, tornam difícil o aumento de velocidade em um único processador;
- Vários processadores fornecem uma configuração modular, o que permite o agrupamento desses processadores em módulos, de acordo com a natureza da aplicação. Além disso, tem-se um meio de extensão através da inclusão de novos módulos;
- Tolerância a falhas (redundância de *hardware*).

Além disso, várias aplicações são inerentemente paralelas, e ao executá-las seqüencialmente, perde-se desempenho.

7.2 Conceitos Básicos

Esta seção apresenta alguns conceitos fundamentais sobre computação paralela.

Paralelismo e Concorrência

Concorrência existe quando, em um determinado instante, dois ou mais processos começaram a sua execução, mas não terminaram. Por essa definição, concorrência pode ocorrer tanto em sistemas com um único processador, quanto em sistemas com múltiplos processadores.

Afirmar que processos estão sendo executados em paralelo implica na existência de mais de um processador, ou seja, o paralelismo (ou paralelismo físico) ocorre quando há mais de um processo sendo executado no mesmo intervalo de tempo. Esse tipo de concorrência é demonstrado na Figura 1 que mostra a execução de três processos (e1, e2 e e3) em função do tempo.

Figura 7.1 - Paralelismo Físico

Quando vários processos são executados em um único processador, sendo que somente um deles é executado a cada vez, tem-se um pseudo-paralelismo (paralelismo lógico). O usuário tem a falsa impressão de que suas tarefas são executadas em paralelo, mas, na realidade, o processador é compartilhado entre os processos. Isso significa que, em um determinado instante, somente um processo é executado, enquanto que os outros que já foram iniciados aguardam a liberação do processador para continuarem, vide Figura 7.2.

Figura 7.2 - Paralelismo Lógico

Baseado nas definições apresentadas acima é possível definir três tipos de estilo de programação dentro da computação, que são:

Programação seqüencial: caracteriza-se pela execução de várias tarefas uma após a outra;

Programação concorrente: caracteriza-se pela iniciação de várias tarefas, sem que as anteriores tenham necessariamente terminado (sistemas multi ou uniprocessadores);

Programação paralela: caracteriza-se pela iniciação e execução das tarefas em paralelo (sistemas multiprocessadores).

Granulação

Granulação, ou nível de paralelismo relaciona o tamanho das unidades de trabalho submetidas aos processadores. Esta é uma definição muito importante na computação paralela, visto que está intimamente ligada ao tipo de plataforma (o porte e a quantidade de processadores) à qual se aplica o paralelismo [Kimer, 1991].

Diversas definições de granulação podem ser encontradas na literatura [Almasi, Gottlieb, 1994; Hwang, 1984; Kimer, 1991, Navaux, 1989]. Mas, de maneira simples, a granulação pode ser dividida em três níveis: fina, média e grossa.

Granulação grossa relaciona o paralelismo ao nível de processos e programas, e geralmente se aplica a plataformas com poucos processadores grandes e complexos [Kimer, 1991; Navaux, 1989].

Granulação fina, por outro lado, relaciona paralelismo ao nível de instruções ou operações e implica em grande número de processadores pequenos e simples. A granulação média situa-se em um patamar entre as duas anteriores, implicando em procedimentos sendo executados em paralelo [Kimer, 1991; Navaux, 1989].

Speedup e Eficiência

Uma característica fundamental da computação paralela está relacionada ao aumento de velocidade de processamento através da utilização do paralelismo. Neste contexto, existem duas

medidas importantes que são utilizadas na verificação da qualidade dos algoritmos paralelos. Essas medidas são o *speedup* e eficiência.

Uma definição largamente aceita para *speedup* é: aumento de velocidade observado quando se executa um determinado processo em p processadores em relação à execução deste processo em 1 processador. Então, tem-se:

$$speedup = \frac{T_1}{T_p},$$

onde: T_1 = tempo de execução em 1 processador

T_p = tempo de execução em p processadores

No caso ideal, o ganho de *speedup* deveria tender a p . Porém, podemos citar três fatores que influenciam essa relação, gerando sobrecargas e diminuindo assim o valor de *speedup* ideal: sobrecarga da comunicação entre os processadores, partes do código que é executado de modo estritamente seqüencial (*Amdahl's Law*) e o nível de paralelismo utilizado (em virtude do uso de granulação inadequada à arquitetura) [Almasi, 1994; Quinn, 1987].

A eficiência trata da relação entre o *speedup* e o número de processadores. Ela representa a taxa média de utilização de p processadores e é dada por:

$$eficiência = \frac{speedup}{p}$$

No caso ideal ($speedup = p$), a eficiência seria máxima e teria valor 1 (100%).

7.3 Arquiteturas Paralelas

O conjunto de Arquiteturas computacionais pode ser dividido em três grupos básicos:

- Arquiteturas seqüências;
- Arquiteturas com paralelismo de baixo nível [Duncan, 1990];
- Arquiteturas paralelas.

A utilização de arquiteturas paralelas possui pontos positivos e negativos. Entre os pontos positivos, têm-se todas as vantagens já citadas anteriormente no início do capítulo.

Como pontos negativos, podemos citar:

- Programação mais difícil;
- A necessidade de se usar técnicas de balanceamento de carga para se obter uma melhor distribuição dos processos nos vários processadores;
- A necessidade de se usar sincronismo e comunicação entre os processos, o que gera certa complexidade;
- A sobrecarga gerada no sistema (por exemplo, na comunicação entre processos), que impede que se tenha um *speedup* ideal.

Existem muitas maneiras de se organizar computadores paralelos. Para que se possa visualizar melhor todo o conjunto de possíveis opções de arquiteturas paralelas, é interessante classificá-las. Segundo Ben-Dyke [Ben-Dyke, 1993], uma classificação ideal deve ser:

Hierárquica: iniciando em um nível mais abstrato. À medida que as arquiteturas vão se diferenciando, a classificação vai sendo refinada em sub-níveis;

Universal: um computador único deve ter uma classificação única;

Extensível: futuras máquinas que surjam, devem ser incluídas sem que sejam necessárias modificações na classificação;

Concisa: os nomes que representam cada uma das classes devem ser pequenos para que a classificação seja de uso prático;

Abrangente: a classificação deve incluir todos os tipos de arquiteturas existentes.

Muito já foi desenvolvido em termos de *hardware* paralelo, e várias classificações foram propostas [Almasi, Gottlieb, 1994; Ben-Dyke, 1993; Duncan, 1990; Hwang, Briggs, 1984]. A mais conhecida pela comunidade computacional é a classificação de Flynn [Flynn, 1972].

Classificação de Flynn

Segundo Flynn, o processo computacional deve ser visto como uma relação entre fluxos de instruções e fluxos de dados. Um fluxo de instruções equivale a uma sequência de instruções executadas (em um processador) sobre um fluxo de dados aos quais estas instruções estão relacionadas [Almasi, Gottlieb, 1994; Ben-Dyke, 1993; Duncan, 1990; Hwang, Briggs, 1984].

Nesta classificação as arquiteturas foram divididas em quatro categorias de máquinas, conforme o fluxo de dados e de instruções.

Os fluxos podem ser:

I- fluxo de instrução (*instruction stream*);

D- fluxo de dados (*data stream*);

Sendo que cada um destes podem ser:

S- único;

M- múltiplo.

As classes assim obtidas são as seguintes:

SISD: *Single Instruction / Single Data*. - Computadores sequenciais.

SIMD: *Single Instruction / Multiple Data*. - Supercomputadores vetoriais.

MISD: *Multiple Instruction / Single Data*. - Computadores que operam com pipeline.

MIMD: *Multiple Instruction / Multiple Data*. - Computadores paralelos intrínsecos.

Em seguida será feito um breve comentário sobre as classes de arquitetura comentada acima:

SISD (Single Instruction/Single Data)

Esta é a arquitetura dos computadores clássicos (seqüenciais). Ela possui apenas um fluxo de instruções e um fluxo de dados, as instruções são executadas seqüencialmente sobre um único fluxo de dados: correspondendo ao modelo tradicional de Von Neumann. A unidade de controle é única, mas pode dispor de várias unidades funcionais. A figura abaixo mostra um esquema desta arquitetura.

Figura 7.3 - Modelo Computacional SISD

SIMD (Multiple Instruction/Single Data)

Envolve múltiplos processadores (escravos) sob o controle de uma única unidade de controle (mestre), executando simultaneamente a mesma instrução em diversos conjuntos de dados. Possuem funcionamento síncrono.

Figura 7. 4 - Modelo Computacional SIMD

As arquiteturas SIMD são típicas de máquinas que dispõem de um grande número de processadores simples e de baixo desempenho. Às vezes não se podem utilizar estes processadores isoladamente. Alguns só trabalham sobre um bit e são dotados de uma memória de milhares de bits (exemplo: *connexion machine*). Como resultado, os programas não podem ser armazenados na memória de cada processador.

MISD (Multiple Instruction/Single Data)

Este tipo de arquitetura envolve múltiplos processadores executando diferentes instruções em um único conjunto de dados. Geralmente, nenhuma arquitetura é classificada como MISD, isto é, não existem representantes desta categoria. Alguns autores consideram arquiteturas *pipeline* como exemplo deste tipo de organização.

Figura 7. 5 - Modelo Computacional MISD

MIMD (Multiple Instruction/Multiple Data)

Existe uma unidade de controle e uma memória de tamanho pequeno ou grande (segundo a arquitetura adotada: a memória distribuída ou global) para cada processador, os quais executam tarefas diferentes com dados diferentes. O funcionamento é assíncrono. A figura abaixo mostra um esquema da arquitetura MIMD.

Figura 7. 6 - Modelo Computacional MISD

As arquiteturas MIMD são constituídas tipicamente por máquinas que dispõem de um pequeno número de processadores que são independentes e relativamente potentes. As transferências de informações são feitas de maneira desordenada, e são guiadas pelos dados.

Classificação de Duncan

Duncan [Duncan, 1990] propôs uma classificação de arquiteturas paralelas mais moderna e abrangente, com o objetivo de encaixar as inovações arquiteturais dos últimos anos e uma taxonomia mais coerente. Para o desenvolvimento dessa classificação, três aspectos básicos são considerados:

- Excluir arquiteturas que possuem mecanismos de paralelismo de baixo nível, os quais são características comuns nos computadores atuais. Um exemplo desses mecanismos é a utilização de *pipeline*;
- Manter os elementos da classificação de Flynn devido sua ampla utilização;
- Incluir arquiteturas que foram criadas após a classificação de Flynn e não se encaixam nela.

A classificação de Duncan divide as arquiteturas em dois grupos principais: arquiteturas síncronas e assíncronas.

Figura 7.7 - Classificação de Duncan

Arquiteturas Síncronas

Arquiteturas paralelas síncronas coordenam suas operações concorrentes de modo síncrono, em todos os processadores, através de relógios globais, unidades de controle únicas ou controladores de unidades vetoriais [Duncan, 1990]. Tais arquiteturas apresentam pouca flexibilidade para a expressão de algoritmos paralelos [Blech, 1994].

Processadores Vetoriais: possuem um *hardware* específico para a otimização de operações sobre vetores. A organização básica do processador de instrução possui uma unidade de processamento vetorial e um processador escalar;

Arquiteturas SIMD: as máquinas SIMD são compostas por um único fluxo de instruções atuando sobre diferentes dados, como já descrito anteriormente. Os processadores matriciais são exemplos dessa categoria síncrona, onde a unidade de controle supervisiona e sincroniza as unidades de processamento.

Arquiteturas Sistólicas: são arquiteturas projetadas para solucionar problemas específicos, que demandem computação intensa e grandes quantidades de operações de E/S. Os processadores das pontas possuem comunicação com a memória.

Arquiteturas Assíncronas

Estas arquiteturas caracterizam-se pelo controle descentralizado do *hardware*, de maneira que os processadores sejam independentes entre si. Esta classe é formada basicamente pelas arquiteturas MIMD, convencionais ou não [Duncan, 1990].

Arquiteturas MIMD convencionais: correspondem as arquiteturas MIMD, discutidas na classificação de Flynn. A organização da memória é que determina como serão feitas as comunicações e o sincronismo entre os processadores. A memória pode ser compartilhada (comum a todos os processadores) ou distribuída (cada processador possui sua própria memória).

Arquiteturas MIMD não convencionais: essa classe engloba as arquiteturas, que apesar de possuírem múltiplos fluxos de dados e instruções, possuem também características próprias que dificultam a classificação com puramente MIMD. Nesta classe encontram-se as máquinas híbridas (MIMD/SIMD), arquiteturas a fluxo de dados (*dataflow*), arquiteturas de redução e arquiteturas de frente de onda. Elas apresentam aspectos MIMD, pois são assíncronas e com múltiplos fluxos de instruções e de dados, mas possuem características próprias, impedindo que sejam classificadas apenas como MIMD [Duncan, 1990].

7.4 Programação Paralela

A programação paralela e concorrente existe para que se forneçam ferramentas para a construção de programas paralelos, de modo que se consiga um melhor desempenho e uma melhor utilização do *hardware* paralelo disponível. Como já foi dito anteriormente, a computação paralela apresenta muitas vantagens em relação à computação seqüencial, e todas essas vantagens podem ser citadas como pontos de incentivo para o uso da programação concorrente.

Um programa seqüencial é constituído basicamente de um conjunto de construções já bem dominadas pelo programador em geral, como por exemplo, atribuições, comandos de decisão (if... then... else), laços (for... do), entre outras. Um programa concorrente, além dessas

primitivas básicas, necessita de novas construções que permitam tratar todos os aspectos decorrentes da execução paralela de vários processos.

Segundo Almasi [Almasi, 1994], para a execução de programas paralelos, deve haver meios de:

- Definir um conjunto de tarefas a serem executadas paralelamente;
- Ativar e encerrar a execução dessas tarefas;
- Coordenar e especificar a interação entre essas tarefas.

Projeto de Algoritmos Paralelos

A chegada das máquinas paralelas impôs uma mudança na área de métodos numéricos, para poder tratar, de maneira concorrente, as partes independentes de um mesmo programa. A solução numérica de um problema; no contexto seqüencial, é composto de três etapas:

1. escolha do algoritmo;
 2. implementação do algoritmo em uma máquina;
 3. estudo do desempenho.
4. No contexto do cálculo paralelo, a solução numérica deste mesmo problema é essencialmente composta de seis etapas:
5. escolha do algoritmo;
 6. divisão das tarefas;
 7. atribuição das tarefas aos processadores;
 8. sincronização das tarefas;
 9. implementação do algoritmo paralelo em uma máquina;
 10. estudo do desempenho.

Em consequência do exposto, a problemática do paralelismo é constituída, em parte, por problemas de matemática aplicada e, em parte, por problemas de natureza computacional.

Estilos de Paralelismo

Um algoritmo paralelo pode ser organizado de várias maneiras, de acordo com o tipo de plataforma onde será executado. Genericamente, os modelos computacionais paralelos são divididos em dois tipos: memória compartilhada e troca de mensagens.

O modelo paralelo baseado em memória compartilhada divide-se de acordo com o tipo de acesso à memória. No modo síncrono (ou *PRAM - Parallel Random Access Memory - Acesso Aleatório Paralelo à Memória*), os processadores atuam de modo síncrono no acesso à memória, enquanto no modo assíncrono esse sincronismo de acesso não existe.

O modelo paralelo, baseado em troca de mensagens, é dividido em três estilos, que são: paralelismo geométrico, paralelismo algorítmico ou *processor farming*. Como nosso modelo é baseado em troca de mensagens iremos detalhá-lo.

Paralelismo Geométrico

Também conhecido como paralelismo de resultados ou paralelismo de dados (*Data Parallelism*), caracteriza-se pela divisão do conjunto de dados a serem trabalhados igualmente entre todos os processadores. Dessa maneira, cada processador executa uma cópia do programa completo, porém em um subconjunto de dados. Este é considerado o modo mais fácil de desenvolvimento de algoritmos paralelos e é intensamente utilizado em computadores massivamente paralelos.

Paralelismo Processor Farm

Caracteriza-se pela existência de um processador “mestre” que supervisiona um grupo de processadores “escravos”. Cada escravo processa, de modo síncrono, as tarefas submetidas a ele pelo processador “mestre”. Este modelo também é conhecido como paralelismo pela pauta, visto que é definido um conjunto de tarefas (pauta) e a partir daí essas tarefas são distribuídas pelo processador mestre.

O estilo *processor farm* possui várias vantagens, como por exemplo: facilidade de ampliação do sistema, o que pode ser conseguido através do aumento de trabalhadores;

facilidade de programação; balanceamento de carga mais natural, visto que as tarefas vão sendo submetidas aos processadores de acordo com a disponibilidade. Como desvantagens, podem ser citadas: a sobrecarga de comunicação e a possibilidade de gargalo no processador “mestre”.

Paralelismo Pipeline

Também conhecido por paralelismo especialista ou algorítmico, caracteriza-se pela divisão de uma aplicação em várias tarefas específicas, que são distribuídas aos processadores de forma *pipeline*. Nesse caso, quando apenas uma unidade de dados atravessa o *pipeline* não há paralelismo.

Muitas vezes, este é o estilo mais natural para desenvolver paralelismo a partir de um programa seqüencial, sendo até possível a sua detecção por compiladores e analisadores de código fonte. Ele possui algumas desvantagens, tais como: pouca flexibilidade, visto que modificações no algoritmo podem requerer mudanças drásticas na rede de processadores; o balanceamento de carga deve ser feito de maneira cuidadosa, para que as sub-tarefas lentas não tornem a execução de todo o *pipeline* lenta; a relação entre comunicação e execução deve ser baixa, a fim de que a sobrecarga de comunicação não torne o algoritmo ineficiente.

Durante o desenvolvimento de algoritmo paralelo, deve-se procurar escolher o estilo de paralelismo mais natural ao problema. Desenvolver o algoritmo utilizando o método de programação mais natural ao estilo pode gerar resultados não satisfatórios em relação ao desempenho. Deve-se então transpor o algoritmo para um estilo de paralelismo mais eficiente. Pode-se utilizar métodos definidos na literatura para facilitar essa transição entre estilos de paralelismo.

Comunicação e Sincronismo

A comunicação é necessária para que processos interagindo na resolução de determinada aplicação troquem informações. E quando há comunicação, devem existir operações de sincronização, para fornecer controle de acesso e controle de seqüência.

Controle de seqüência (também chamado de sincronização condicional ou sincronização de atividades) é utilizado para que se determine uma ordem na qual os processos (ou partes deles) devem ser executados. O controle de acesso é necessário quando há competição entre processos para a manipulação de algum recurso. Deve-se então, garantir que acessos

concorrentes a esses recursos sejam controlados, para que se mantenha a consistência [Almasi, 1994; Quinn, 1987; Snow, 1992].

Comunicação e Sincronismo em Memória Centralizada

Comunicação e sincronismo (controle de acesso e seqüência) em memória centralizada são implementados através da utilização de variáveis compartilhadas entre os diversos processos concorrentes.

O controle de acesso é geralmente implementado através de exclusão mútua. São definidas seções críticas, isto é, conjuntos de instruções que devem ser executadas de maneira mutuamente exclusiva, para que os processos não sofram interferência durante a execução dessas seções.

Utilizando-se de variáveis compartilhadas, vários métodos podem ser utilizados para a implementação de exclusão mútua e controle de seqüência, entre eles: *busy-waiting*, semáforos e monitores [Almasi, 1994; Andrew, 1983; Quinn, 1987; Snow, 1992].

Comunicação e Sincronismo em Memória Distribuída

Comunicação e sincronismo em arquiteturas de memória distribuída devem ser implementados através de troca de mensagens entre processos [Almasi, 1994; Andrew, 1983; Quinn, 1987; Snow, 1992]. Uma operação de comunicação via mensagens, de maneira genérica, é realizada pela utilização das primitivas *send/receive* (envia/recebe), cujas sintaxes, por exemplo, podem ser:

Send mensagem ***to*** processo_destino

Receive mensagem ***from*** processo_fonte

Uma transferência de mensagem pode ser realizada de duas maneiras. Uma operação bloqueante implica que o processo que transmite a mensagem (transmissor) é bloqueado até que receba uma confirmação de recebimento da mensagem pelo processo receptor. Caso contrário, tem-se uma operação não bloqueante, isto é, o processo transmissor envia a mensagem (que deve ser armazenada em um *buffer*) e continua a sua execução.

As primitivas *send/receive* podem ser organizadas para efetuar trocas de mensagens, gerando três mecanismos básicos: comunicação ponto a ponto, *rendezvous* e RPC.

Suporte para a Programação Paralela

Devido aos fatores pertinentes no projeto, uma decisão extremamente importante é a escolha do tipo de ferramenta que será usada para a construção de um programa que represente o algoritmo paralelo definido.

Vários aspectos devem ser levados em consideração nessa escolha. Entre eles, o tipo de aplicação e a arquitetura que executará os códigos gerados. Baseado no tipo de aplicação e na arquitetura define-se a granulação desejada, que deve ser considerada também no processo de escolha. Segundo Almasi [Almasi, 1994], dois fatores principais devem ser considerados: o tempo de trabalho do programador (e aqui se inclui o tempo de aprendizado da ferramenta), e o desempenho obtido.

Almasi [Almasi, 1994], relaciona três tipos de ferramentas para construção de programas paralelos: ambientes de paralelização automática, extensões paralelas para linguagens seriais e linguagens concorrentes.

Compiladores paralelizadores, que caracterizam ambientes de paralelização automática, são responsáveis por gerarem automaticamente versões paralelas de programas não paralelos. Tais compiladores exigem o mínimo de trabalho do usuário, mas o desempenho obtido geralmente é modesto. Além disso, nem sempre esses compiladores são disponíveis, principalmente para sistemas de memória distribuída [Blech, 1994].

Extensões paralelas são bibliotecas que contêm um conjunto de instruções que complementam as linguagens seriais já existentes. Esses ambientes requerem algum trabalho do programador, mas evita a necessidade de aprendizagem de uma nova linguagem ou a completa reescrita do código fonte (no caso de se paralelizar um programa já implementado). Geralmente, o desempenho obtido é superior ao obtido pelos compiladores paralelizadores. Existem extensões para arquiteturas de memória distribuída (MPI [Dongarra, et al. 1995], PVM [Beguelin, 1994; Geist, 1994], P4) e para memória compartilhada (Pascal Concorrente).

O terceiro tipo de ferramenta relaciona as linguagens criadas especialmente para processamento concorrente, o que implica em tempo de aprendizagem de uma linguagem totalmente nova e reescrita total do código fonte. Essas ferramentas tendem a fornecer melhores desempenhos, de maneira que se compense a sobrecarga sobre o programador. Outra vantagem dessas linguagens é que geralmente elas possibilitam a construção de códigos bem estruturados, tornando fácil a identificação dos processos que estão executando em paralelo e a comunicação entre eles. Exemplos dessas linguagens são: *Occam* e *Ada*.

7.5 Considerações Finais

As diversas áreas na qual a computação se aplica, demandam cada vez mais poder computacional e vários autores acreditam que esse poder computacional só pode ser conseguido através do processamento paralelo.

Dentre as plataformas de execução de programas paralelos, destaca-se o modelo MIMD com memória distribuída, em função de sua grande flexibilidade e facilidade de ampliação [Blech, 1994; Zaluska, 1991]. Dentro do modelo MIMD, uma tendência atual é a utilização de sistemas distribuídos, interligados por redes de comunicação rápidas, como plataformas de programação paralela [Blech, 1994]. Sistemas distribuídos representam uma solução de custo acessível, em comparação às arquiteturas paralelas de alto custo [Beguelin, 1994; Blech, 1994; Zaluska, 1991].

Capítulo 8

Implementação e Resultados Computacionais

Neste capítulo serão discutidos alguns aspectos relativos à implementação do Método de Fase I paralelo, bem como sua execução sobre a máquina virtual paralela. Apresentaremos alguns testes computacionais realizados com o algoritmo por nós desenvolvido, bem como uma análise dos resultados obtidos em termos de *speedup* e eficiência.

8.1 Implementação do algoritmo

O método desenvolvido foi implementado em linguagem C e compilado no *GNU Compiler Collection* (GCC 3.2.1). Na implementação do algoritmo foram usadas apenas bibliotecas do ANSI C, tornando o código portátil para qualquer computador que tenha compilador ANSI C. Como a maioria dos compiladores é compatível com ANSI C, o nosso código é compatível com a maioria dos compiladores, como por exemplo: Borland C/C++, Visual C/C++, DEV C/C++, GCC e outros.

O Algoritmo implementado trabalha essencialmente com uma estrutura de dados que armazena a rede de multifluxo completa e mais duas outras estruturas para cada produto que circula na rede; sendo que uma representa a árvore de solução do subproblema e sua co-árvore, e a outra representa a rede de simples fluxo com as informações sobre fluxo, demanda, origem e destino do arco, capacidade e custo.

O programa inicia carregando a rede multifluxo através de um arquivo de dados. Os dados lidos deste arquivo são armazenados em uma estrutura que contém: o número de nós, número de arcos, número de produtos, a matriz de demanda da rede, e o vetor com as informações sobre cada arco da rede.

A matriz de demanda tem a dimensão de número de nós versus número de produtos. Cada coluna da matriz se refere à demanda de um produto, por exemplo, na primeira coluna é armazenada a demanda do produto um, na coluna dois, é armazenada a demanda do produto dois, etc. Então se o produto um entra no nó 1 e sai no nó 10, a posição $D_{1,1}$ da matriz guarda o valor da demanda $+b_1$, que é a quantidade do produto 1 que entra no nó 1, e a posição $D_{10,1}$ guarda o valor de $-b_1$, que é a quantidade do produto 1 que sai do nó 10.

O vetor que contém as informações sobre a rede é um vetor de registros onde, a primeira posição contém as informações se o arco está ou não na base, sua origem, destino, capacidade, fluxo e custo, relativos ao arco 1. Na segunda posição são armazenadas essas mesmas informações com relação ao arco 2 e assim por diante.

A partir dessa estrutura são geradas as redes de simples fluxo. Nas redes de simples fluxo a estrutura é semelhante à anterior com exceção da matriz de demanda que aqui é substituída por um vetor de demanda.

Essas estruturas são geradas no mestre. A seguir, o mestre envia para os escravos as estruturas referentes às redes de simples fluxo. Nos escravos, é gerada uma solução a partir dessas redes.

O armazenamento da solução utilizando uma matriz de incidência nó-arco exigiria uma quantidade elevada de memória, limitando assim a dimensão dos problemas que poderíamos tratar. Como essas matrizes são esparsas, e tendo em vista que o número de arcos que fazem parte da solução não é fixo e, variam a cada iteração, optou-se por uma estrutura de dados que seja mais enxuta e dinâmica.

A estrutura de dados que armazena a solução nos escravos contém: o número de nós da rede, o número de arcos da solução, a multa, os arcos artificiais, os arcos que fazem parte da solução (árvore + arcos com fluxo no limite superior) e os arcos que estão fora da solução (arcos da co-árvore que estão com fluxo igual a zero).

Os arcos que fazem parte da solução são armazenados da seguinte maneira: criou-se um vetor de nós da rede. Em cada posição desse vetor, sai uma lista encadeada dinâmica que contém os arcos que entram e saem do nó, relativo a esta posição do vetor. Se o arco sai do nó, ele tem índice positivo; se o arco entra no nó, ele tem índice negativo. Uma estrutura semelhante foi utilizada para armazenar os arcos que não fazem parte da solução (arcos fora da base com fluxo zero). Os arcos artificiais são armazenados através de um vetor de registros que contém a origem do arco, o destino e o fluxo que circula por eles.

Na figura 8.1a é apresentado um exemplo da estrutura de dados utilizada para a rede de simples fluxo. Em 8.1a temos a rede completa. Em 8.1b temos, o que seria, uma solução de 8.1a e em 8.1c temos a estrutura de dados utilizada para armazenar a solução mostrada em 8.1b.

Observe que na rede da figura 8.1a temos 5 nós e 8 arcos. Na rede de solução temos 5 nós e 6 arcos. Então, na estrutura que chamamos de *vert*, estão guardados os nós da rede de solução. No nó 1, por exemplo, l_1 , l_2 e l_3 indicam que estes arcos estão saindo do nó 1. No nó 2, $-l_3$ indica que este arco está entrando no nó 2 e l_5 indica que o arco 5 está saindo do nó 2.

Na estrutura *covert* estão armazenados os arcos que estão fora da base e com fluxo zero. No nosso exemplo, o nó 1, não tem nenhum arco saindo ou chegando com fluxo igual a zero. No nó 2, o arco l_6 , está saindo do nó e com fluxo zero.

Apesar das estruturas dinâmicas serem naturalmente mais lentas que as estáticas, a estrutura utilizada permite que se faça uma busca; como no caso da procura de um ciclo; bastante rápida, porque a dimensão das listas encadeadas dinâmicas é no máximo igual ao grau do nó. É vale a pena lembrar que, nas redes grandes dificilmente teremos nós cujo grau é igual ao número de nós. Desta forma, a dimensão das listas encadeadas é relativamente pequena e a quantidade de memória utilizada é inferior a quantidade de memória utilizada pela matriz de incidência nó-arco. No pior dos casos, a estrutura utilizada ocupa $(N \times N)$ posições de memória. Se fosse usada a matriz de incidência nó-arco, estaríamos armazenando $(N \times NA)$, e como numa rede conexa o número de arcos NA é sempre superior ao número de nós, o número de posições de memória utilizada seria bem maior.

OBS: A linha tracejada indica um arco fora da base no limite superior

Figura 8. 1- Exemplo da estrutura de dados utilizada para representar a solução

Para procurar um ciclo nesta estrutura, realiza-se uma busca em profundidade a partir do arco candidato. Esta busca ocorre da seguinte maneira:

No passo 1, o nó de origem do arco candidato a entrar na base é marcado como visitado e o arco candidato é inserido na lista encadeada que armazena o ciclo.

No passo 2, marca-se o nó destino do arco escolhido no passo anterior como visitado. A seguir procura-se um arco que tenha origem no nó que acabou de ser marcado. Este arco é inserido na lista que forma o ciclo.

No passo 3, verifica-se se o nó de destino do arco escolhido no passo 2 já foi marcado. Caso tenha sido marcado, a busca termina. Do contrário, volta-se ao passo 2.

Se no passo 2 não for possível encontrar nenhum arco com origem no último nó marcado, o nó é então desmarcado, o último arco inserido na lista é retirado e a busca retoma no passo 2, a partir do último (anteriormente penúltimo) nó marcado.

Na implementação paralela do algoritmo, adotou-se o modelo de computação paralela conhecido por *Processor Farm*, onde um mestre supervisiona os escravos. A atribuição das tarefas aos processadores é feita de modo estático, isto é, os processos são atribuídos aos escravos no início da execução, e cada escravo vai executar sempre o mesmo processo.

Ao mestre cabe o papel de distribuir os processos aos escravos e de verificar se a solução do problema de multifluxo de Fase I é factível. Aos escravos cabe o papel de calcular a solução dos problemas de simples fluxo.

Para verificar a factibilidade da solução, o mestre executa uma *function*, aqui chamada de função de multifluxo, que trabalha da seguinte maneira: a função multifluxo recebe a estrutura de dados que armazena a rede de multifluxo e gera as M redes de simples fluxo. A seguir as redes de simples fluxos são enviadas aos escravos. Após o envio de todas as redes, o mestre fica bloqueado a espera das soluções (todas) que serão computadas pelos escravos.

Neste algoritmo existem basicamente dois pontos de comunicação. O primeiro ponto de comunicação se dá no início do programa com o envio das tarefas (redes de simples fluxo) aos escravos. Os escravos retornam ao mestre a solução encontrada. Este passo é chamado de primeira iteração. Então, na primeira iteração do algoritmo, a função multifluxo do mestre, após receber dos escravos a solução dos subproblemas, verifica se existe algum subproblema infactível. A existência de um subproblema infactível neste passo significa que o problema de multifluxo não tem solução.

O segundo ponto de comunicação ocorre nas iterações seguintes após a verificação da factibilidade da solução de multifluxo (passo 2 e 3 do algoritmo). Se a solução for factível, o algoritmo finaliza; caso contrário o mestre calcula as novas capacidades dos arcos, para os problemas de simples fluxo (passo 4) e as envia aos escravos. Os escravos recalculam as redes de

simples fluxo e retornam a nova solução para o mestre. Este processo se repete até que o problema global de multifluxo seja factível.

As comunicações entre mestre e escravos foram feitas utilizando a biblioteca PVM (Parallel Virtual Machine) [Beguelin, 1994; Geist, 1994].

O número de tarefas que o mestre distribui (processos escravos) é sempre igual ao número de produtos que circulam na rede multifluxo, e cada produto corresponde a uma tarefa. Caso o número de tarefas (produtos) seja maior que o número de processadores existentes na máquina virtual paralela, mais de uma tarefa será designada a um processador escravo. Observe que, se o número de tarefas não for um múltiplo do número de processadores, haverá um desbalanceamento de carga, e por conseguinte, uma redução na eficiência.

A distribuição das tarefas (*spawn*) ocorre apenas uma vez no início do programa, minimizando o *overhead*. O mestre, através do PVM, decide qual processador irá executar qual tarefa.

8.2 Resultados Obtidos

O Algoritmo desenvolvido foi executado em uma rede de microcomputadores AMD K6-2 500MHz, com (32 + 32) kbytes de memória L1 e 512 Kbytes de memória L2, 128MB de memória principal de 64 bits a 100MHz. O sistema operacional usado foi o UNIX FreeBSD 5.0, executando apenas os serviços essenciais e de rede (NFS + DNS), isto é, a rede estava dedicada à execução da máquina virtual paralela durante os testes. A rede de computadores é composta por 22 microcomputadores que estão interligados por um Switch 3COM 10/100 SuperStack II 1100 24p 3C16950.

Para fins de comparação de *Speedup* e Eficiência, o algoritmo apresentado na seção 6.2 foi implementado de duas formas: uma versão seqüencial e outra paralela, usando a biblioteca de passagem de mensagens *Parallel Virtual Machine* (PVM) versão 3.4.4.

Foram testadas 4 redes de multifluxo, para cada uma dessas redes fizemos variações no número de produtos que circulavam por elas. Ao se variar a quantidade de fluxo numa rede, altera-se completamente o problema. Como queríamos fazer uma avaliação da influência do número de produtos no *speedup*, procuramos variar o problema o menos possível, por isso

tomamos o cuidado de manter constante a quantidade de fluxo total circulante, uma vez que a variação do fluxo poderia influenciar no tempo de processamento. A combinação entre as redes e a variação de produtos resultou em 46 problemas.

Na tabela abaixo fazemos uma descrição da rede01m3, que é uma rede por onde circulam 3 produtos. O arquivo de entrada de dados é dividido em três partes. Na primeira parte do arquivo, indicamos o número de nós, número de arcos e número de produtos da rede. Na segunda parte, mostramos o vetor de demanda, isto é, em quais nós cada produto entra e sai da rede. Por último, indicamos: a origem, o destino, o custo e a capacidade de cada arco da rede.

nnos: é o número de nós da rede;
narcos: o número de arcos da rede;
nprod: o número de produtos da rede;
no_or: nó origem do arco;
no_dt: nó destino do arco;
cap: capacidade do arco

nnos	narcos	nprod			
22	87	3			
arco	p1	p2	p3		
1	20	0	0		
2	16	0	0		
3	0	0	-25		
4	0	13	0		
5	0	0	-11		
6	0	23	0		
7	11	0	0		
8	0	24	0		
9	0	0	-12		
10	0	0	0		
11	21	0	0		
12	0	0	0		
13	0	0	11		
14	-11	0	0		
15	-16	0	0		
16	0	-13	0		
17	0	-23	0		
18	0	0	12		
19	0	-24	0		
20	0	0	25		
21	-20	0	0		
22	-21	0	0		

arco	no_or	no_dt	custo	cap
1	1	5	3	15
2	1	6	4	20
3	1	7	2	22
4	1	2	3	10
5	2	1	5	20
6	2	7	1	18
7	8	2	3	12
8	2	3	5	21
9	3	2	4	15
10	3	8	2	15
11	9	3	3	20
12	3	4	2	19
13	4	3	2	10
14	4	8	1	16
15	4	9	3	22
16	18	4	5	20
17	14	5	2	20
18	5	10	3	30
19	5	11	1	22
20	6	5	2	28
21	10	6	1	10
22	6	11	3	18
23	6	12	1	13
24	7	6	2	15
25	11	7	2	15
26	7	12	1	10
27	7	13	3	18
28	8	7	1	16
29	12	8	2	10
30	8	13	4	19
31	9	8	0	23
32	9	13	5	30
33	13	9	5	30
34	9	18	2	20
35	10	14	1	10
36	10	15	2	18
37	11	10	1	22
38	14	11	0	12
39	11	16	3	25
40	11	12	1	16
41	12	11	0	20
42	15	12	1	25
43	16	12	0	17
44	12	17	1	16
45	17	12	0	15
46	13	12	2	20
47	17	13	3	22

48	13	17	2	20
49	13	18	0	15
50	14	19	0	10
51	19	14	1	14
52	15	14	0	15
53	15	19	3	20
54	19	15	0	20
55	15	16	1	15
56	16	15	0	10
57	11	15	1	15
58	19	16	3	10
59	16	20	2	23
60	21	16	4	14
61	16	21	2	20
62	22	16	3	15
63	16	17	4	20
64	17	21	2	15
65	17	22	3	20
66	17	18	1	10
68	18	17	3	15
69	18	22	4	10
70	22	18	5	14
71	22	21	4	30
72	21	22	3	25
73	20	21	0	22
74	21	20	1	15
75	20	19	4	22
76	19	20	3	20
77	19	21	5	18
78	21	19	4	21
79	22	20	3	15
80	20	22	0	20
81	18	21	5	20
82	21	18	2	30
83	5	2	0	20
84	6	3	3	15
85	7	4	1	18
86	7	8	2	18
87	8	9	3	21

OBS: As demais redes serão descritas no Anexo A.

Os resultados obtidos são descritos nas tabelas a seguir:

A rede 1 contém 22 nós, 87 arcos. Por ser uma rede pequena, o número máximo de produtos que fizemos circular na rede foi 10. O fluxo total que circula na rede é 176 unidades. Nestes problemas o número de variáveis varia de 174 (rede02m2) a 870 (rede02m10) variáveis.

Rede 01	No. de processadores	Tempo paralelo (s)	Tempo seqüencial (s)	Speedup	Eficiência (%)
Rede01m2	3	0,0412	0,0244	0,5917	19,7245
Rede01m3	4	0,0741	0,1011	1,3636	34,0908
Rede01m4	5	0,1678	0,4256	2,5357	50,7137
Rede01m5	6	0,1366	0,4129	3,0234	50,3901
Rede01m6	7	0,3685	1,3939	3,7824	54,0347
Rede01m7	8	0,6262	2,9307	4,6802	58,5029
Rede01m8	9	0,8907	4,6081	5,1738	57,4871
Rede01m9	10	8,3837	55,7378	6,6484	66,4837
Rede01m10	11	5,4702	40,4961	7,4031	67,3006

Tabela 8.1 - Resultados da rede 01

A tabela 8.1 mostra os tempos de execução seqüencial e paralelo, o *speedup* e a eficiência da rede 01. No tempo paralelo considerou-se que um processador está encarregado do processo mestre. O processo que o mestre executa é menor que o processo dos escravos. Enquanto os escravos executam um algoritmo do simplex, o mestre verifica a factibilidade de cada subproblema, corrige as capacidades dos arcos no sentido de factibilizar os subproblemas, testa a factibilidade global da solução e coordena o envio e recepção de dados para os escravos. Então, na rede02m2 temos dois produtos e três processadores trabalhando.

Pelo gráfico 8.1 podemos observar que o *speedup* teve um crescimento praticamente linear com relação ao aumento de número de produtos. Isto era esperado uma vez que a dimensão do problema cresce em função do número de produtos. Podemos também observar, através da tabela 8.1 e do gráfico 8.2 que a eficiência do algoritmo oscilou entre 50 e 60%, com exceção das redes 01m2 e 01m3. Isto vem nos mostrar, o que já era esperado, que para problemas pequenos, os algoritmos paralelos são pouco eficientes, pois o número de comunicações tem um peso muito grande. À medida que os problemas crescem, diminui a influência das comunicações sobre o tempo total de processamento. A média da eficiência nesta rede foi de 50,96%.

Gráfico 8. 1 - Speedup da rede 01

Gráfico 8. 2 - Eficiência da rede 01

A tabela 8.2 abaixo mostra os resultados obtidos com a rede 02. Esta rede possui 29 nós e 132 arcos. Começamos a rede 02 com dois produtos e fomos aumentando o número de produtos até dez. Então, o número de variáveis dos problemas varia de 264 a 1320 variáveis. As características completas desta rede poderão ser encontradas no Anexo A. Aqui também

podemos observar, através do gráfico 8.2, um crescimento linear do *speedup* confirmando a tendência já observada na rede 01. Também observamos que a eficiência cresce com o aumento do problema, pois a granularidade das tarefas cresce. A média da eficiência obtida para esta rede está em 50,97%. Na rede02m10 (10 produtos e 11 processadores) a eficiência foi de 84%. Isto ocorreu porque neste problema o algoritmo convergiu bem mais rápido que nos outros, isto é, o número de iterações foi bem menor. Este comportamento é muito comum nos algoritmos iterativos.

Rede 02	No. de processadores	Tempo paralelo (s)	Tempo Sequencial (s)	Speedup	Eficiência (%)
Rede02m2	3	0,0534	0,037859	0,7090	23,6330
Rede02m3	4	0,0713	0,101376	1,4219	35,5476
Rede02m4	5	0,0667	0,131975	1,9786	39,5726
Rede02m5	6	0,1111	0,317203	2,8553	47,5877
Rede02m6	7	0,1075	0,334961	3,1160	44,5147
Rede02m7	8	0,6948	3,408704	4,9063	61,3290
Rede02m8	9	0,4200	2,242251	5,3387	59,3191
Rede02m9	10	1,7442	10,917333	6,2592	62,5924
Rede02m10	11	8,5865	79,946842	9,3107	84,6431

Tabela 8. 2 - Resultados da rede 02

Gráfico 8. 3 - Speedup da rede 02

Gráfico 8. 4 - Eficiência da rede 02

Na tabela 8.3 apresentados os resultados obtidos com a rede 03. Ela contém 60 nós e 396 arcos. Nesta rede o número de produtos variou de 4 a 14 e o número de variáveis a serem determinadas variou de 1584 (rede03m4) a 5544 variáveis (rede03m14). Aqui também o *speedup* (gráfico 8.5) conservou seu crescimento linear.

Podemos observar pelo gráfico 8.6 que a eficiência do algoritmo ficou num patamar mais constante entre 77 a 90%, com média em 83,62%. Como já havíamos dito anteriormente, a tendência da eficiência é aumentar com o tamanho do problema e se tornar estável. Esta estabilidade só é influenciada pelo fato do algoritmo convergir muito rápido em alguns casos, como por exemplo, na rede03m14 que o número de iterações foi bem menor que nas demais.

Rede 03	No. de processadores	Tempo Paralelo (s)	Tempo seqüencial (s)	Speedup	Eficiência (%)
Rede03m4	5	0,5981	2,1564	3,6053	72,1067
Rede03m5	6	0,7224	3,3520	4,6399	77,3313
Rede03m6	7	0,7353	4,3610	5,9306	84,7228
Rede03m7	8	0,6372	4,0602	6,3722	79,6522
Rede03m8	9	0,8958	6,3704	7,1111	79,0119
Rede03m9	10	0,7069	5,9965	8,4829	84,8292
Rede03m10	11	1,1234	11,0839	9,8667	89,6975
Rede03m11	12	1,9509	20,8747	10,6998	89,1652
Rede03m12	13	4,0345	47,2704	11,7166	90,1273
Rede03m13	14	6,0117	77,1770	12,8377	91,6980
Rede03m14	15	8,3484	102,1265	12,2331	81,5542

Tabela 8. 3 - Resultados da rede 03

Gráfico 8. 5 - Speedup da rede 03

Gráfico 8. 6 - Eficiência da rede 03

Na tabela 8.4 apresentamos os resultados obtidos com a rede 04. Esta rede possui 60 nós e 432 arcos. O número de produtos circulando na rede variou de 4 a 20. Com isto o número de variáveis variou de 1728 (rede04m4) a 8460 variáveis (rede04m20). Esta rede mais uma vez vem confirmar o crescimento linear do *speedup* (gráfico 8.7). A eficiência, confirmando uma tendência ocorrida anteriormente, também se mostrou mais ou menos estável num patamar entre 70 e 80%, com média de 73,18%.

Rede 04	No. de processadores	Tempo paralelo (s)	Tempo seqüencial (s)	Speedup	Eficiência (%)
Rede04m4	5	0,5025	1,5687	3,1221	62,4420
Rede04m5	6	0,5895	2,4750	4,1985	69,9746
Rede04m6	7	1,0095	5,1736	5,1248	73,2118
Rede04m7	8	0,8162	4,4944	5,5063	68,8288
Rede04m8	9	0,9328	6,0166	6,4503	71,6700
Rede04m9	10	1,2113	8,7808	7,2490	72,4900
Rede04m10	11	1,4569	11,5219	7,9084	71,8945
Rede04m11	12	1,1250	9,6462	8,5743	71,4522
Rede04m12	13	1,2353	11,8950	9,6293	74,0716
Rede04m13	14	2,1138	22,3857	10,5905	75,6464
Rede04m14	15	2,4774	28,8220	11,6342	77,5611
Rede04m15	16	3,2998	39,5781	11,9942	74,9640
Rede04m16	17	4,0655	52,8538	13,0004	76,4732
Rede04m17	18	4,3276	59,7242	13,8009	76,6718
Rede04m18	19	4,0690	58,4351	14,3612	75,5851
Rede04m19	20	2,5622	36,7798	14,3546	71,7731
Rede04m20	21	5,0569	84,2806	16,6665	79,3642

Tabela 8. 4 - Resultados da rede 04

Rede 04 - 60 nós e 432 arcos

Gráfico 8. 7 - Speedup da rede 04

Gráfico 8. 8 - Eficiência da rede 04

Em todas as redes testadas o *speedup* manteve uma taxa de crescimento linear, mostrando um bom desempenho do algoritmo paralelo. À medida que o tamanho dos problemas aumentava, o tempo gasto com comunicações de dados entre mestre e escravo se tornava menos influente, fazendo com que a eficiência chegasse a um patamar de 70%. Isto ocorre porque com o aumento no tamanho do problema, aumenta-se a granularidade das tarefas executadas pelo escravo.

Apesar de trabalharmos com um processador somente para executar as tarefas do mestre, o *speedup* e a eficiência do algoritmo foram excelentes. Vale observar aqui que o processo executado pelo mestre apesar de ser seqüencial é muito pequeno quando comparado com os processos dos escravos e por isso não chega a ser um gargalo muito grande para o algoritmo.

Conclusões

Nesta dissertação é proposto um método paralelo para encontrar a solução inicial do problema de multifluxo convexo. O objetivo inicial deste projeto consistia em encontrar a solução inicial do Problema de Roteamento de Dados. O problema de Roteamento de Dados é um problema de multifluxo com características especiais, como por exemplo, as restrições de acoplamento [Minoux, 1983] que já estão embutidas na função objetivo. Uma outra característica do problema de roteamento é que cada produto possui somente um par de nós origem-destino.

O método aqui desenvolvido é bem geral e pode ser utilizado para resolver qualquer problema de multifluxo, seja ele linear ou convexo. Isto é possível por duas razões:

1. consideramos que cada produto pode ter mais de um par de nós origem-destino.
2. nosso método não leva em conta o custo dos arcos. Ele trabalha somente com as capacidades dos arcos e estas são lineares, independentemente se o problema de multifluxo é linear ou convexo.

O método desenvolvido tem algumas características importantes, tais como:

- é eficaz, pois é capaz de solucionar problemas com um grande número de variáveis em segundos. Resolvemos problemas com 8460 variáveis (20 produtos) em 5 segundos. Para este número de variáveis, é impensável encontrar uma solução inicial manualmente.

- é de fácil implementação paralela. Nosso algoritmo é do tipo MIMD, isto é, cada escravo executa sempre um mesmo processo (Simplex linear) para diferentes dados. No início do algoritmo é enviado ao escravo a estrutura da rede na qual ele irá trabalhar. O escravo trabalha sempre na mesma rede, variando a cada iteração somente a capacidade dos arcos. Deste modo, foi necessário escrever somente dois códigos fontes: um para o mestre e outro que é utilizado por todos os escravos.
- o volume de comunicações é pequeno. Para redes de pequeno porte o volume de comunicações tem uma influência maior na eficiência, mantendo-a entre 30 a 40%, mas para redes um pouco maiores a eficiência do algoritmo ficou acima de 70%.
- *speedup* com crescimento linear. Uma dos fatores que indicam que um código paralelo é eficiente é o *speedup* linear.

Durante o desenvolvimento deste projeto encontramos algumas dificuldades que consideramos importante serem citadas. Os principais problemas encontrados foram:

- encontrar uma maneira de recalcular, a cada iteração, as capacidades dos arcos de maneira que as redes dos subproblemas não se tornassem infactíveis e ao mesmo tempo possibilitasse uma rápida convergência do algoritmo. Inicialmente, cada arco da sub-rede possui uma capacidade igual à capacidade total do arco (C_l). Esta capacidade vai diminuindo para os arcos cuja soma dos fluxos (fluxo total (X_l)) seja maior que a capacidade total dos arcos. A dificuldade encontrada foi justamente, na maneira de fazer esta redução.
- como trabalhamos com alocação dinâmica de memória, tivemos que ter o cuidado de desalocar a memória após cada iteração do método. Se tivéssemos utilizado uma alocação estática de memória, não teríamos que fazer esta desalocação, mas teríamos uma utilização de memória com crescimento quadrático com relação ao tamanho da entrada do problema. No caso da alocação dinâmica o crescimento é linear. Optamos pela alocação dinâmica para que pudéssemos executar problemas

maiores. A desvantagem da alocação dinâmica de memória é o aumento do tempo computacional global.

- Uma outra dificuldade encontrada é devido ao fato que não existem depuradores para programas em paralelo. Se por ventura ocorrer algum erro no programa, não existe nenhuma ferramenta eficaz para ajudar a encontrar o erro, sobretudo se o erro ocorrer nos processos escravos. Também não é possível fazer impressões a partir dos escravos. Tudo isso dificulta bastante as implementações em paralelo. Para amenizar este problema, inicialmente implementamos o algoritmo seqüencialmente. Só depois que o programa estava sendo executado sem nenhum problema, é que procedemos à implementação em paralelo. Este procedimento, apesar de demandar um tempo muito grande, é a maneira mais eficaz de se fazer uma implementação em paralelo.

Gostaríamos de ressaltar algumas contribuições oferecidas por este trabalho:

- Desenvolvemos um algoritmo paralelo que encontra a solução inicial de qualquer problema de multifluxo, seja ele linear ou convexo.
- O algoritmo paralelo é eficiente e mais rápido que o seqüencial, mesmo para redes de pequeno porte.
- O algoritmo foi projetado para resolver problemas de grande porte.

Propostas de trabalhos futuros

Uma proposta interessante de trabalho futuro seria desenvolver um programa que fosse capaz de gerar as redes, que seriam utilizadas para testar o programa, aleatoriamente. É muito trabalhoso gerar essas redes manualmente e, além disso, existe o risco de se gerar uma rede desconexa. Com a geração automática este risco desapareceria.

Uma outra proposta é desenvolver uma segunda versão do algoritmo de Fase I que utilizasse o Método Dual Simplex para factibilizar os arcos que estão com fluxo total maior que a capacidade total do arco. Chegamos até a pensar em como fazê-lo, mas não houve tempo hábil para implementá-lo.

Referências Bibliográficas

ALMASI, G. S.; GOTTLIEB, A. *Highly Parallel Computing*. 2.ed., The Benjamin Cummings Publishing Company, 1994.

AHUJA, R. K.; ORLIN, J. B. The Scaling Network Simplex Algorithm. *Operations Research*, vol.40, supp. no.1, jan./fev. 1992.

ANDREWS, G. R.; SCHNEIDER, F. B. Concepts and Notations for Concurrent Programming. *ACM Computing Survey*, v.15, no.1,p.3-43, 1983.

ARMSTRONG, R. D.; JIN, Z. A New Strongly Polynomial Dual Network Simplex Algorithm. *Mathematical Programming*, v.78, p.131-148, 1997.

AUTHIE, Gerard. *Contribution à L'optimization des Flots Dans les Réseaux. Un Multiprocesseur Expérimental por L'étude des itérations Asynchrones*. Toulouse, França, 1987. Thèse de Doctorat d'Etat, Université Paul Sabatier.

BAZARAA, M. S.; JARVIS, J. J. *Linear Programming and Network Flows*. John Wiley & Sons, 1977.

BEGUELIN A.; *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.

BEN-DYKE, A. D. *Architectural taxonomy, A brief review*. University of Birmingham, 1993.

BERTSEKAS, D. P.; GAFNI, M. Projected Newton methods and optimization of multicommodity flows. *IEEE Trans. Automat. Control*, v.28, p.1090-1096, 1983.

BERTSEKAS, D. P.; GALLAGER, R. *Data Networks*, Eaglewood Cliffs, Prentice-Hall, 1987.

BERTSEKAS, D. P.; GAFNI, M. Projected Newton Methods and Optimization of Multicommodity Flows. *IEEE Trans. Automat. Control*, v.28, p.1090-1096, 1983.

- BLAND, R. G. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, v.2, p.103-107, 1977.
- BLECH, R. A. *An Overview of Parallel Processing*, Slides, Parallel Computing with PVM Workshop, Nasa Lewis Research Center, http://www.lerc.nasa.gov/Other_Groups/IFMD/2620/tutorialPP.html, 1994.
- CUNNINGHAM, W. H.; KLINCEWICZ, J. G. On Cycling in the Network Simplex Method. *Mathematical Programming*, v.23, p.182-189, 1983.
- DEMBO, R. S.; KLINCEWICZ, J. G. A Scaled Reduced Gradient Algorithm for Network Flow Problem with Convex Separable Costs. *Mathematical Programming*, v.15, p.125-147, 1981.
- DONGARRA, J.; OTTO, S. W.; SNIR, M.; WALKER, D. *An Introduction to the MPI Standard* University of Tennessee Technical, report CS-95-274, January 1995.
- DUNCAN, R. A Survey of Parallel Computer Architectures. *IEEE Computer*, p.5-16, fev. 1990.
- FLYNN, M. J. Some Computer Organizations and their Effectiveness IEEE. *Transactions on Computers*, v.C-21, p.948-960, 1972.
- FRATA, L.; GERLA, M.; KLEINROCK, L. The Deviation Method: Na Approach to Store and Forward Communication Network Design. *Networks*, v.3, p.97-133, 1973.
- GALLAGER, R. G. A Minimum Delay Routing Algorithm Using Distributed Commutation. *IEEE Transactions on Communications*, v.25, n.1, p.73-85, 1977.
- GEIST, A. *PVM3 - User's Guide and Reference Manual*. Oak Ridge National Laboratory, set. 1994.
- GLOVER F.; KARNEY, D.; KLINGMAN, D. Implementation and Computational Comparisons of Primal, Dual and primal-Dual Computer Codes for Minimum Cost Network Flow Problem. *Networks*, v.4, p.191-212, 1974.
- GOLDBERG, A. V.; TARJAN, R. E. Solving Minimum Cost Flow Problem by Successive Approximation, *Proceeding In: 19th Annual Symposium on Theory of Computing*, p.136-146, 1987.
- GOLDFARB, D.; CHEN, W. On Strongly Polynomial Dual Simplex Algorithms for the Maximum Flow Problem. *Mathematical Programming*, v.8, p.159-168, 1997.
- GONDRAN, M.; MINOUX, M. *Graphs and Algorithms*, John Wiley & Son, 1984.
- HERNANDES, Fábio. *Implementação do Algoritmo Paralelo para o Problema de Roteamento de Dados no Computador Paralelo IBM-SP2*. São Carlos, 1999. Dissertação (Mestrado) - Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo.

HWANG, K.; BRIGGS, F. A. *Computer Architecture and Parallel Processing*. McGraw-Hill International Editions, 1994.

HU, T. C. *Integer Programming and Network Flows*. Addison-Wesley, 1969.

JENSEN, P. A.; BARNES, J. W. *Network Flow Programming*. John Wiley & Sons, 1980.

KENNINGTON, J. L.; HELGASON, R. V. *Algorithms for Network Programming*. John Wiley & Sons, 1980.

KIRNER, C., Arquiteturas de sistemas avançados de computação. In: JORNADA EPUSP/IEEE EM SISTEMAS DE COMPUTAÇÃO DE ALTO DESEMPENHO, São Paulo, 1991. *Anais*. São Paulo, USP, 1991. p.307-353.

KLEINROCK, L. *Communication Nets: Stochastic Message Flow and Delay*, New York, McGraw-Hill, 1964.

LUVEZUTE, Rosana Maria. *Algoritmos para Problema de Multifluxo Não Linear: Uma Aplicação ao Problema de Roteamento de Dados em Redes de Comutação*. São Carlos, 1995. Dissertação (Mestrado) Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo.

MCBRIDE, R. D. Progress Made in Solving the Multicommodity Flow Problem. *Siam Journal on Optimization*, v.8, n.4, p.947-955, 1998.

MINOUX, M., *Programmation Mathématique: Théorie et Algorithmes*, tomes 1 e 2, Dunod, 1983.

MULVEY, J. Pivot Strategies for Primal-Simplex Network Codes. *J. ACM*, v.25, p.266-270, 1978.

NAVAUX, P. O. A. Introdução ao processamento paralelo. *RBC - Revista Brasileira de Computação*, v.5, n.2, p.31-43, Out. 1989.

ORLIN, J. B. On the Simplex Algorithm for Networks and Generalized Networks. *Math. Prog. Study*, v.24, p.166-178, 1985.

QUINN, M. J. *Designing Efficient Algorithms for Parallel Computers*, McGraw Hill, 1987.

RIBEIRO, C. M.; HERNANDES, F.; OLIVEIRA, F. C. J.; RIBEIRO, J. F. F. A Method for Optimal Routing Problem: An implementation in the System PVM-W95. In: INTERNATIONAL SYMPOSIUM ON MATHEMATICAL THEORY OF NETWORKS AND SYSTEMS, Perpignan, 2000. *Anais*. Perpignan, França, Jun. 2000.

RIBEIRO, C. M.; HERNANDES, F.; OLIVEIRA, F. C. J.; RIBEIRO, J. F. F. A Parallel Algorithm for the Routing Problem: An Implementation in The Parallel Computer IBM-SP2. In: 14th ACM SYMPOSIUM ON APPLIED COMPUTING, Villa Olmo, Como, Itália, Mar. 2000.

- RIBEIRO, C. M. BAZ, D. E. A Parallel Optimal Routing Algorithm. *Parallel Computing*, v.18, p.1393-1402, North Holland, 1992.
- RIBEIRO, C. M.; LUVEZUTE, R. M.; RIBEIRO, J. F. F. An Algorithm for Optimal Routing. In: 8th IFAC/IFORS/IMACS/IFIP. Symposium on Large Scale Systems: Theory and Application, University of Patras. Patras, Grécia, Julho, 1998. p.207-211.
- ROCKFELLAR, R. T. *Network Flows and Monotropic Optimization*. John Wiley & Sons, 1984.
- ROTHSCHILD, B. On Two-commodity Network Flows. *Operations Research*, v.14, p.377-387, 1966.
- SCHWARTZ, M.; CHENG, C. The Gradient Projection Algorithm for Multiple Routing in Message-Switched Networks. *IEEE Trans. on Commun*, p.449-456, 1976.
- SNOW, C. R. *Concurrent Programming*, Cambridge University Press, 1992.
- SOUZA, Paulo Sérgio Lopes. *Máquina Paralela Virtual no Ambiente Windows*. São Carlos. Maio, 1996. Dissertação (Mestrado) - Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo.
- SAKAROVITCH, M. *Optimisation Combinatoire: Graphes et Programation Linéaire*. Hermann, 1984.
- SOKKALINGAM, P. T.; SHARMA, P.; AHUJA, R. K. A New Pivot Selection Rule for the Network Simplex Algorithm. *Mathematical Programming*, v.78, p.149-158, 1997.
- STERN, T. A Class of Decentralized Routing Algorithms Using Relaxation. *IEEE Transaction on Computers*, v.c-25, p.1092-1102, 1977.
- SZWARFITER, J. L. *Grafos e Algoritmos*. Editora Campus, 1986.
- TARJAN, R. E. Dynamic Trees as Search Trees via Euler Tours, Applied to the Network Simplex Algorithm. *Mathematical Programming*, v.78, p.169-177, 1997.
- TSAI, K.; HUANG, G.; ANTONIO, K.; TSAI, T. Distributed Iterative Aggregation Algorithms for Box Constrained Minimization Problems and Optimal Routing in Data Networks. *IEEE Trans. Automat. Control*, v.34, p.34-46, 1989.
- TSITSIKLIS, J.; BERTSEKAS, D. P. Distributed Asynchronous Optimal Routing in Data Networks. *IEEE Trans. Automat Control*, v.31, p.325-332, 1986.
- ZALUSKA, E. J. *Research lines in distributed computing systems and concurrent computation*, In: WORKSHOP EM PROGRAMAÇÃO CONCORRENTE, *Anais. Sistemas Distribuidos e Engenharia de Software*, 1991. p.132-155.

Anexo A

Apresentamos a seguir as características gerais das redes 02, 03 e 04. Gostaríamos de ressaltar que não estamos indicando os nós onde entram e saem os fluxos relativos a cada produto da rede.

Logo abaixo apresentamos a rede 02. Esta rede tem 29 nós e 132 arcos.

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
1	1	5	3	15	24	7	6	2	15
2	1	6	4	20	25	11	7	2	15
3	1	7	2	22	26	7	12	1	10
4	1	2	3	10	27	7	13	3	25
5	2	1	5	20	28	8	7	1	18
6	2	7	1	18	29	12	8	2	12
7	8	2	3	12	30	8	13	4	19
8	2	3	5	25	31	9	8	0	23
9	3	2	4	15	32	9	13	5	30
10	3	8	2	15	33	13	9	5	30
11	9	3	3	20	34	9	18	2	20
12	3	4	2	26	35	10	14	1	10
13	4	3	2	10	36	10	15	2	18
14	4	8	1	16	37	11	10	1	22
15	4	9	3	22	38	14	11	0	12
16	18	4	5	20	39	11	16	3	25
17	14	5	2	20	40	11	12	1	16
18	5	10	3	30	41	12	11	0	20
19	5	11	1	22	42	15	12	1	25
20	6	5	2	28	43	16	12	0	17
21	10	6	1	10	44	12	17	1	24
22	6	11	3	18	45	17	12	0	15
23	6	12	1	13	46	13	12	2	20

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
47	17	13	3	22	91	24	19	1	10
48	13	17	2	20	92	19	20	4	13
49	13	18	0	18	93	19	21	5	20
50	14	19	0	10	94	21	19	2	15
51	19	14	1	14	95	20	23	1	25
52	15	14	0	15	96	26	20	0	30
53	15	19	3	20	97	20	24	1	28
54	19	15	0	20	98	20	21	3	26
55	15	16	1	15	99	21	26	4	25
56	16	15	0	10	100	24	21	5	10
57	11	15	1	17	101	21	28	3	14
58	19	16	3	13	102	25	21	1	10
59	16	20	2	27	103	21	25	0	20
60	21	16	4	14	104	22	21	5	12
61	16	21	2	20	105	21	22	6	18
62	22	16	3	15	106	25	22	2	15
63	16	17	4	20	107	22	25	3	16
64	17	21	2	15	108	22	28	1	22
65	17	22	3	20	109	22	29	2	28
66	17	18	1	10	110	26	23	1	11
67	18	8	2	15	111	23	26	3	12
68	18	17	3	15	112	23	27	2	15
69	18	22	4	10	113	23	24	4	15
70	22	18	5	14	114	26	24	3	13
71	22	21	4	30	115	24	27	2	12
72	21	22	3	25	116	24	28	3	18
73	20	21	0	22	117	24	28	1	16
74	21	20	1	15	118	25	27	6	28
75	20	19	4	22	119	25	28	4	19
76	19	20	3	20	120	25	29	3	14
77	19	21	5	18	121	27	26	5	10
78	21	19	4	21	122	28	26	4	10
79	22	20	3	15	123	29	26	1	14
80	20	22	0	20	124	28	27	1	10
81	18	21	5	20	125	28	29	2	18
82	21	18	2	30	126	28	23	3	10
83	5	2	0	20	127	29	28	0	13
84	6	3	3	15	128	28	25	1	12
85	7	4	1	18	129	29	22	1	10
86	7	8	2	18	130	29	18	3	15
87	8	9	3	21	131	29	27	2	12
88	19	26	2	18	132	26	14	1	16
89	23	19	3	20					
90	19	23	2	12					

A rede 03 tem 60 nós e 396 arcos. Suas características principais são:

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
1	1	5	3	15	42	15	12	1	25
2	1	60	4	20	43	16	12	0	17
3	1	7	2	22	44	12	17	1	24
4	1	2	3	10	45	17	12	0	15
5	2	1	5	20	46	13	12	2	20
6	2	7	1	18	47	17	13	3	22
7	8	2	3	12	48	13	17	2	20
8	2	3	5	25	49	13	18	0	18
9	3	2	4	15	50	14	19	0	10
10	3	8	2	15	51	19	14	1	14
11	9	3	3	20	52	15	14	0	15
12	3	4	2	26	53	15	19	3	20
13	4	3	2	10	54	19	15	0	20
14	4	8	1	16	55	15	16	1	15
15	4	9	3	22	56	16	15	0	10
16	18	4	5	20	57	11	15	1	17
17	14	5	2	20	58	19	16	3	13
18	5	10	3	30	59	16	20	2	27
19	5	11	1	22	60	21	16	4	14
20	6	5	2	28	61	16	21	2	20
21	10	6	1	10	62	22	16	3	15
22	6	11	3	18	63	16	17	4	20
23	6	12	1	13	64	17	21	2	15
24	7	6	2	15	65	17	22	3	20
25	11	7	2	15	66	17	18	1	10
26	7	12	1	10	67	18	8	2	15
27	7	13	3	25	68	18	17	3	15
28	8	7	1	18	69	18	22	4	10
29	12	8	2	12	70	22	18	5	14
30	8	13	4	19	71	22	21	4	30
31	9	8	0	23	72	21	22	3	25
32	9	13	5	30	73	20	21	0	22
33	13	9	5	30	74	21	20	1	15
34	9	18	2	20	75	20	19	4	22
35	10	14	1	10	76	19	20	3	20
36	10	15	2	18	77	19	21	5	18
37	11	10	1	22	78	21	19	4	21
38	14	11	0	12	79	22	20	3	15
39	11	16	3	25	80	20	22	0	20
40	11	12	1	16	81	18	21	5	20
41	12	11	0	20	82	21	18	2	30

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
83	5	2	0	20	130	29	18	3	15
84	6	3	3	15	131	29	27	2	12
85	7	4	1	18	132	26	14	1	16
86	7	8	2	18	133	1	35	3	15
87	8	9	3	21	134	1	36	4	20
88	19	26	2	18	135	1	37	2	22
89	23	19	3	20	136	1	32	3	10
90	19	23	2	12	137	2	31	5	20
91	24	19	1	10	138	2	37	1	18
92	19	20	4	13	139	8	32	3	12
93	19	21	5	20	140	2	33	5	25
94	21	19	2	15	141	3	30	4	15
95	20	23	1	25	142	3	38	2	15
96	26	20	0	30	143	9	33	3	20
97	20	24	1	28	144	3	34	2	26
98	20	21	3	26	145	4	33	2	10
99	21	26	4	25	146	4	38	1	16
100	24	21	5	10	147	4	30	3	22
101	21	28	3	14	148	18	34	5	20
102	25	21	1	10	149	14	35	2	20
103	21	25	0	20	150	5	40	3	30
104	22	21	5	12	151	5	41	1	22
105	21	22	6	18	152	60	45	2	28
106	25	22	2	15	153	10	46	1	10
107	22	25	3	16	154	6	41	3	18
108	22	28	1	22	155	6	42	1	13
109	22	29	2	28	156	7	46	2	15
110	26	23	1	11	157	11	47	2	15
111	23	26	3	12	158	7	42	1	10
112	23	27	2	15	159	7	43	3	25
113	23	24	4	15	160	8	47	1	18
114	26	24	3	13	161	12	38	2	12
115	24	27	2	12	162	8	43	4	19
116	24	30	3	18	163	9	38	0	23
117	24	28	1	16	164	9	43	5	30
118	25	27	6	28	165	13	39	5	30
119	25	28	4	19	166	9	48	2	20
120	25	29	3	14	167	10	44	1	10
121	27	26	5	10	168	10	45	2	18
122	28	26	4	10	169	11	40	1	22
123	29	26	1	14	170	14	41	0	12
124	30	27	1	10	171	11	46	3	25
125	28	29	2	18	172	11	42	1	16
126	28	23	3	10	173	12	41	0	20
127	29	28	0	13	174	15	42	1	25
128	28	25	1	12	175	16	42	0	17
129	30	22	1	10	176	12	47	1	24

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
177	17	42	0	15	224	19	50	4	13
178	13	42	2	20	225	19	51	5	20
179	17	43	3	22	226	21	49	2	15
180	13	47	2	20	227	20	53	1	25
181	13	48	0	18	228	26	50	0	30
182	14	49	0	10	229	20	54	1	28
183	19	44	1	14	230	20	51	3	26
184	15	44	0	15	231	21	56	4	25
185	15	49	3	20	232	24	51	5	10
186	19	30	0	20	233	21	58	3	14
187	15	46	1	15	234	25	51	1	10
188	16	45	0	10	235	21	55	0	20
189	11	45	1	17	236	22	51	5	12
190	19	46	3	13	237	21	52	6	18
191	16	50	2	27	238	25	52	2	15
192	21	46	4	14	239	22	55	3	16
193	16	51	2	20	240	22	58	1	22
194	22	46	3	15	241	22	59	2	28
195	16	47	4	20	242	26	53	1	11
196	17	51	2	15	243	23	56	3	12
197	17	52	3	20	244	23	57	2	15
198	17	48	1	10	245	23	54	4	15
199	18	38	2	15	246	26	54	3	13
200	18	47	3	15	247	24	57	2	12
201	18	52	4	10	248	24	58	3	18
202	22	48	5	14	249	24	58	1	16
203	22	51	4	30	250	30	57	6	28
204	21	52	3	25	251	25	58	4	19
205	20	51	0	22	252	25	59	3	14
206	21	50	1	15	253	27	56	5	10
207	20	49	4	22	254	28	56	4	10
208	19	50	3	20	255	29	56	1	14
209	19	51	5	18	256	28	57	1	10
210	21	49	4	21	257	28	59	2	18
211	22	50	3	15	258	28	53	3	10
212	20	52	0	20	259	29	58	0	13
213	18	51	5	20	260	28	55	1	12
214	21	48	2	30	261	29	52	1	10
215	5	32	0	20	262	29	48	3	15
216	6	60	3	15	263	29	57	2	12
217	7	34	1	18	264	26	44	1	16
218	7	38	2	18	265	31	5	3	15
219	8	39	3	21	266	31	6	4	20
220	19	56	2	18	267	30	7	2	22
221	23	49	3	20	268	31	2	3	10
222	19	53	2	12	269	32	1	5	20
223	24	49	1	10	270	32	7	1	18

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
271	38	2	3	12	318	49	15	0	20
272	32	3	5	25	319	45	16	1	15
273	33	2	4	15	320	46	15	0	10
274	33	8	2	15	321	41	15	1	17
275	39	3	3	20	322	49	16	3	13
276	33	4	2	26	323	46	20	2	27
277	34	3	2	10	324	51	16	4	14
278	34	8	1	16	325	46	21	2	20
279	34	9	3	22	326	52	16	3	15
280	48	4	5	20	327	46	17	4	20
281	44	5	2	20	328	47	21	2	15
282	35	10	3	30	329	47	22	3	20
283	35	11	1	22	330	47	18	1	10
284	36	5	2	28	331	48	8	2	15
285	40	6	1	10	332	48	17	3	15
286	36	11	3	18	333	48	22	4	10
287	36	12	1	13	334	52	18	5	14
288	37	6	2	15	335	52	21	4	30
289	41	7	2	15	336	51	22	3	25
290	37	12	1	10	337	50	21	0	22
291	37	13	3	25	338	51	20	1	15
292	38	7	1	18	339	50	19	4	22
293	42	8	2	12	340	49	20	3	20
294	38	13	4	19	341	49	21	5	18
295	39	8	0	23	342	51	19	4	21
296	39	13	5	30	343	52	20	3	15
297	43	9	5	30	344	50	22	0	20
298	39	18	2	20	345	48	21	5	20
299	40	14	1	10	346	51	18	2	30
300	40	15	2	18	347	35	2	0	20
301	30	10	1	22	348	36	30	3	15
302	44	11	0	12	349	37	4	1	18
303	41	16	3	25	350	37	8	2	18
304	41	12	1	16	351	38	9	3	21
305	42	11	0	20	352	49	26	2	18
306	45	12	1	25	353	53	19	3	20
307	46	12	0	17	354	49	23	2	12
308	42	17	1	24	355	54	19	1	10
309	47	12	0	15	356	49	30	4	13
310	43	12	2	20	357	49	21	5	20
311	47	13	3	22	358	51	19	2	15
312	43	17	2	20	359	50	23	1	25
313	43	18	0	18	360	56	20	0	30
314	44	19	0	10	361	50	24	1	28
315	49	14	1	14	362	50	21	3	26
316	45	14	0	15	363	51	26	4	25
317	45	19	3	20	364	54	21	5	10

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
365	51	28	3	14	381	54	28	1	16
366	55	21	1	10	382	55	27	6	28
367	51	25	0	20	383	55	28	4	19
368	52	21	5	12	384	55	30	3	14
369	51	22	6	18	385	57	26	5	10
370	55	22	2	15	386	58	26	4	10
371	52	25	3	16	387	59	26	1	14
372	52	28	1	22	388	58	27	1	10
373	52	29	2	28	389	58	29	2	18
374	56	23	1	11	390	58	23	3	10
375	53	26	3	12	391	59	28	0	13
376	53	27	2	15	392	58	25	1	12
377	53	24	4	15	393	59	22	1	10
378	56	24	3	13	394	59	18	3	15
379	54	27	2	12	395	59	27	2	12
380	54	28	3	18	396	56	14	1	16

A rede 04 descrita abaixo tem 60 nós e 432 arcos.

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
1	1	5	3	15	25	11	7	2	15
2	1	60	4	20	26	7	12	1	10
3	1	7	2	22	27	7	13	3	25
4	1	2	3	10	28	8	7	1	18
5	2	1	5	20	29	12	8	2	12
6	2	7	1	18	30	8	13	4	19
7	8	2	3	12	31	9	8	0	23
8	2	3	5	25	32	9	13	5	30
9	3	2	4	15	33	13	9	5	30
10	3	8	2	15	34	9	18	2	20
11	9	3	3	20	35	30	14	1	10
12	3	4	2	26	36	10	15	2	18
13	4	3	2	10	37	11	10	1	22
14	4	8	1	16	38	14	11	0	12
15	4	9	3	22	39	11	16	3	25
16	18	4	5	20	40	11	12	1	16
17	14	5	2	20	41	12	11	0	20
18	5	10	3	30	42	15	12	1	25
19	5	11	1	22	43	16	12	0	17
20	6	5	2	28	44	12	17	1	24
21	10	6	1	10	45	17	12	0	15
22	6	11	3	18	46	13	12	2	20
23	6	12	1	13	47	17	13	3	22
24	7	6	2	15	48	13	17	2	20

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
49	13	18	0	18	96	26	20	0	30
50	14	19	0	10	97	20	24	1	28
51	19	14	1	14	98	20	21	3	26
52	15	14	0	15	99	21	26	4	25
53	15	19	3	20	100	24	21	5	10
54	19	15	0	20	101	21	28	3	14
55	30	16	1	15	102	25	21	1	10
56	16	15	0	10	103	21	25	0	20
57	11	15	1	17	104	22	21	5	12
58	19	16	3	13	105	21	22	6	18
59	16	20	2	27	106	25	22	2	15
60	21	16	4	14	107	22	25	3	16
61	16	21	2	20	108	22	28	1	22
62	22	16	3	15	109	22	29	2	28
63	16	17	4	20	110	26	23	1	11
64	17	21	2	15	111	23	26	3	12
65	17	22	3	20	112	23	27	2	15
66	17	18	1	10	113	23	24	4	15
67	18	8	2	15	114	26	24	3	13
68	18	17	3	15	115	24	27	2	12
69	18	22	4	10	116	24	28	3	18
70	22	18	5	14	117	30	28	1	16
71	22	21	4	30	118	25	27	6	28
72	21	22	3	25	119	25	28	4	19
73	20	21	0	22	120	25	29	3	14
74	21	20	1	15	121	27	26	5	10
75	20	19	4	22	122	28	26	4	10
76	19	30	3	20	123	29	26	1	14
77	19	21	5	18	124	28	27	1	10
78	21	19	4	21	125	28	29	2	18
79	22	20	3	15	126	28	23	3	10
80	20	22	0	20	127	29	28	0	13
81	18	21	5	20	128	28	25	1	12
82	21	18	2	30	129	29	22	1	10
83	5	2	0	20	130	29	18	3	15
84	6	3	3	15	131	29	27	2	12
85	7	4	1	18	132	26	14	1	16
86	7	8	2	18	133	1	35	3	15
87	8	9	3	21	134	1	36	4	20
88	19	26	2	18	135	1	37	2	22
89	23	19	3	20	136	1	32	3	10
90	19	23	2	12	137	2	30	5	20
91	24	19	1	10	138	2	37	1	18
92	19	20	4	13	139	8	32	3	12
93	19	21	5	20	140	2	33	5	25
94	21	19	2	15	141	3	32	4	15
95	20	23	1	25	142	3	38	2	15

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
143	9	33	3	20	190	19	46	3	13
144	3	34	2	26	191	16	50	2	27
145	4	33	2	10	192	21	46	4	14
146	4	30	1	16	193	16	51	2	20
147	4	39	3	22	194	22	46	3	15
148	18	34	5	20	195	16	47	4	20
149	14	35	2	20	196	17	51	2	15
150	5	40	3	30	197	17	52	3	20
151	5	41	1	22	198	17	48	1	10
152	60	45	2	28	199	18	38	2	15
153	10	46	1	10	200	18	47	3	15
154	6	41	3	18	201	18	52	4	10
155	6	42	1	13	202	22	48	5	14
156	7	46	2	15	203	22	51	4	30
157	11	47	2	15	204	21	52	3	25
158	7	42	1	10	205	20	51	0	22
159	7	43	3	25	206	21	50	1	15
160	8	47	1	18	207	20	49	4	22
161	12	38	2	12	208	19	50	3	20
162	8	43	4	19	209	19	51	5	18
163	9	38	0	23	210	21	49	4	21
164	9	43	5	30	211	22	50	3	15
165	13	39	5	30	212	20	52	0	20
166	9	48	2	20	213	18	51	5	20
167	10	44	1	10	214	21	48	2	30
168	10	45	2	18	215	5	32	0	20
169	11	40	1	22	216	6	60	3	15
170	14	41	0	12	217	7	34	1	18
171	11	46	3	25	218	7	38	2	18
172	11	42	1	16	219	8	39	3	21
173	12	41	0	20	220	19	56	2	18
174	15	42	1	25	221	23	49	3	20
175	16	42	0	17	222	19	53	2	12
176	12	47	1	24	223	24	49	1	10
177	17	42	0	15	224	19	50	4	13
178	13	42	2	20	225	19	51	5	20
179	17	43	3	22	226	21	49	2	15
180	13	47	2	20	227	20	53	1	25
181	13	48	0	18	228	26	50	0	30
182	14	49	0	10	229	20	54	1	28
183	19	44	1	14	230	20	51	3	26
184	15	44	0	15	231	21	56	4	25
185	15	49	3	20	232	24	51	5	10
186	19	45	0	20	233	21	58	3	14
187	15	46	1	15	234	25	51	1	10
188	16	45	0	10	235	21	55	0	20
189	11	45	1	17	236	22	51	5	12

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
237	21	52	6	18	284	36	5	2	28
238	25	52	2	15	285	40	6	1	10
239	22	55	3	16	286	36	11	3	18
240	22	58	1	22	287	36	12	1	13
241	22	59	2	28	288	37	6	2	15
242	26	53	1	11	289	41	7	2	15
243	23	56	3	12	290	37	12	1	10
244	23	57	2	15	291	37	13	3	25
245	23	54	4	15	292	38	7	1	18
246	26	54	3	13	293	42	8	2	12
247	24	57	2	12	294	38	13	4	19
248	24	58	3	18	295	39	8	0	23
249	24	58	1	16	296	39	13	5	30
250	25	57	6	28	297	43	9	5	30
251	25	58	4	19	298	39	18	2	20
252	25	59	3	14	299	40	14	1	10
253	27	56	5	10	300	40	15	2	18
254	28	56	4	10	301	41	10	1	22
255	29	56	1	14	302	44	11	0	12
256	28	57	1	10	303	41	16	3	25
257	28	59	2	18	304	41	12	1	16
258	28	53	3	10	305	42	11	0	20
259	29	58	0	13	306	45	12	1	25
260	28	55	1	12	307	46	12	0	17
261	30	52	1	10	308	42	17	1	24
262	29	48	3	15	309	47	12	0	15
263	29	57	2	12	310	43	12	2	20
264	26	44	1	16	311	47	13	3	22
265	31	5	3	15	312	43	17	2	20
266	31	6	4	20	313	43	18	0	18
267	30	7	2	22	314	44	19	0	10
268	31	2	3	10	315	49	14	1	14
269	32	1	5	20	316	45	14	0	15
270	32	7	1	18	317	45	19	3	20
271	38	2	3	12	318	49	15	0	20
272	32	3	5	25	319	45	16	1	15
273	33	2	4	15	320	46	15	0	10
274	33	8	2	15	321	41	15	1	17
275	39	3	3	20	322	49	16	3	13
276	33	4	2	26	323	46	20	2	27
277	34	3	2	10	324	51	16	4	14
278	34	8	1	16	325	46	21	2	20
279	34	9	3	22	326	52	16	3	15
280	48	4	5	20	327	46	17	4	20
281	44	5	2	20	328	47	21	2	15
282	35	10	3	30	329	47	22	3	20
283	35	11	1	22	330	47	18	1	10

arco	no_or	no_dt	custo	cap	arco	no_or	no_dt	custo	cap
331	48	8	2	15	378	56	24	3	13
332	48	17	3	15	379	54	27	2	12
333	48	22	4	10	380	54	28	3	18
334	52	18	5	14	381	54	28	1	16
335	52	21	4	30	382	55	27	6	28
336	51	22	3	25	383	55	28	4	19
337	50	21	0	22	384	55	29	3	14
338	51	20	1	15	385	57	26	5	10
339	50	19	4	22	386	58	26	4	10
340	49	20	3	20	387	59	26	1	14
341	49	21	5	18	388	58	27	1	10
342	51	19	4	21	389	58	29	2	18
343	52	20	3	15	390	58	23	3	10
344	50	22	0	20	391	59	28	0	13
345	48	21	5	20	392	58	25	1	12
346	51	18	2	30	393	59	22	1	10
347	35	2	0	20	394	59	18	3	15
348	36	3	3	15	395	59	27	2	12
349	37	4	1	18	396	56	14	1	16
350	37	8	2	18	397	8	11	3	15
351	38	9	3	21	398	11	7	2	15
352	49	26	2	18	399	10	14	1	10
353	53	19	3	20	400	10	13	2	18
354	49	23	2	12	401	11	13	1	22
355	54	19	1	10	402	14	10	0	12
356	49	30	4	13	403	11	6	3	25
357	49	21	5	20	404	11	22	1	16
358	51	19	2	15	405	22	11	0	20
359	50	23	1	25	406	15	12	1	25
360	56	20	0	30	407	30	12	0	17
361	50	24	1	28	408	12	20	1	24
362	50	21	3	26	409	20	12	0	15
363	51	26	4	25	410	23	12	2	20
364	54	21	5	10	411	27	13	3	22
365	51	28	3	14	412	33	17	2	20
366	55	21	1	10	413	13	28	0	18
367	51	25	0	20	414	14	29	0	10
368	52	21	5	12	415	19	24	1	14
369	51	22	6	18	416	25	14	0	15
370	55	22	2	15	417	25	19	3	20
371	52	25	3	16	418	29	15	0	20
372	52	28	1	22	419	35	16	1	15
373	52	29	2	28	420	36	15	0	10
374	56	23	1	11	421	21	45	1	17
375	53	26	3	12	422	29	46	3	13
376	53	27	2	15	423	16	40	2	27
377	53	24	4	15	424	11	56	4	14

arco	no_or	no_dt	custo	cap
425	36	11	2	20
426	22	16	3	15
427	46	17	4	20
428	57	13	2	15
429	47	10	3	20
430	37	18	1	10
431	18	48	2	15
432	48	13	3	15