RAQUEL DA SILVA CABRAL

Uma implementação em paralelo para Decomposição de Benders aplicada a sistemas eixo–raio com múltipla atribuição

Maceió

Fevereiro de 2006

RAQUEL DA SILVA CABRAL

Uma implementação em paralelo para Decomposição de Benders aplicada a sistemas eixo–raio com múltipla atribuição

Dissertação apresentada ao Curso de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas como requisito para a obtenção do grau de Mestre em Modelagem Computacional de Conhecimento.

Orientador:
Henrique Pacca L. Luna

Maceió

Fevereiro de 2006

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa Multidisciplinar de Pós-Graduação em
Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 09, Cidade Universitária
57.072-900 Maceió AL Brasil CGC: 24.464.109/0001-48
Telefone: (082) 214-1364/1401 Fax: (082) 214-1615

MODELAGEM COMPUTACIONAL DE CONHECIMENTO

Ata da defesa de dissertação da aluna
Raquel da Silva Cabral

Realizou-se no dia 23 de fevereiro de dois mil e seis, às 10:00 horas, na sala de aula do Curso de Mestrado em Modelagem Computacional de Conhecimento, da Universidade Federal de Alagoas, a defesa de dissertação de Mestrado em Modelagem Computacional de Conhecimento, intitulada "Uma implementação em paralelo do método de decomposição de Benders aplicado a sistemas eixo-raio com múltipla atribuição", apresentada por Raquel da Silva Cabral, diplomada em Ciência da Computação, como requisito parcial para a obtenção do grau de Mestre em Modelagem Computacional de Conhecimento, à seguinte comissão examinadora:

Professor Gilberto de Miranda Júnior
Escola de Engenharia – UFMG

Professor Leonardo Viana Pereira
Instituto de Computação – IC/UFAL

Professor Henrique Pacca Loureiro Luna
Instituto de Computação – IC/UFAL

A dissertação foi considerada aprovada pela Comissão. Finalizados os trabalhos, lavrou-se a presente ata, que vai assinada por mim e pelos membros da Comissão.

Maceió, 23 de fevereiro de 2006.

José Vitor de Menezes Torres
Secretário

Gilberto de Miranda Júnior
Doutor em Ciência da Computação – Universidade Federal de Minas Gerais/Brasil

Leonardo Viana Pereira
Doutor em Física – Universidade Federal de Pernambuco/Brasil

Henrique Pacca Loureiro Luna
Doutor em Automática Otimização – Universite de Toulouse III (Paul Sabatier)/França

AGRADECIMENTOS

Agradeço a Deus por estar sempre comigo.

À minha filha Maria Júlia, razão da minha vida, que me ensina todos os dias com sua inocência de criança.

André, que mesmo distante esteve sempre comigo, acompanhando minhas dificuldades e alegrias, me aconselhando e ouvindo minhas reclamações. Serei eternamente grata a você.

Aos meus pais, Nazaré e Cabral, que apesar de toda sua simplicidade, e com todas as dificuldades que tiveram conseguiram me colocar no caminho certo.

Eunice, que foi minha companheira, nas horas de estudo, na hora das provas e nas horas das notas também. Uma pessoa que me mostrou o que significado das palavras: bondade, simplicidade e persistência.

Ao meu sábio orientador Henrique Pacca, sem ele não poderia jamais estar escrevendo esses agradecimentos agora. Do seu jeito sempre me ensinou o melhor caminho para chegar ao lugar certo.

Às minhas amigas Josi e Angélica, que também sempre estiveram por perto, proporcionando-me momentos de alegria, quando parecia que tudo estava perdido.

À universidade, os alunos, e aos professores que contribuíram para minha formação, em especial o Prof. Dr. Hilário Alencar e a Profa. Dra. Solange Bessa.

RESUMO

Sistemas do tipo eixo–raio, tornaram-se uma importante área de pesquisa da teoria de localização nas últimas décadas. Esse destaque deve-se em grande parte ao sucesso de sua utilização em sistemas logísticos, tanto de transporte de passageiros quanto de cargas, e em redes de telecomunicações. Ao invés de servir cada par origem–destino de demanda com uma conexão direta, sistemas do tipo eixo–raio substituem essas conexões diretas por uma rede de concentradores. Esses concentradores permitem que o tráfego seja agrupado e transportado através de um meio de transporte compartilhado, para ser então entregue aos respectivos destinos. Sendo um problema NP, é necessário o uso de métodos eficientes para sua resolução. Neste trabalho, é desenvolvida uma implementação em paralelo do método de Decomposição de Benders para o problema de localização de concentradores de alocação múltipla não-capacitados. A implementação em paralelo do método de Decomposição de Benders para o problema eixo–raio não é conhecido na literatura, entretanto os bons resultados obtidos pelo algoritmo paralelo desenvolvido revelam que a abordagem paralela é aplicável e mais eficiente. Nos experimentos realizados, o algoritmo paralelo apresentou um tempo de resposta até 70% menor que o tempo de resposta do algoritmo seqüencial.

Palavras-chave: sistemas eixo-raio, programação paralela, decomposição de Benders

ABSTRACT

Hub and Spoke systems, is a important research area in localization theory. This occur, because of these systems are very used in logistics problems, e.g., telecommunication networks and transport of passenger and load.To serve the demand of each pair source–destination, basically, the Hub and Spoke system replaces direct connections between the pairs for a hubs network. These hubs group the traffic sharing the transportation medium. To get the best hubs configuration is necessary efficient methods, because this problem, hubs allocation, is a NP-problem. In this work was developed an parallel implementation of the Benders Decomposition method for the uncapacitated multiple allocation hub location problem. In our implementation we use the Skorin-Kapov model. The parallel implementation of Benders Decomposition for hub and spoke problem is not known in literature. The results show that the parallel approach is applicable and more efficient that nonparallel one. The experiments reveals that the parallel algorithm had a time execution 70% minor when compared with the nonparallel one.

Key-words: hub and spoke systems, parallel computing, Benders decomposition

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

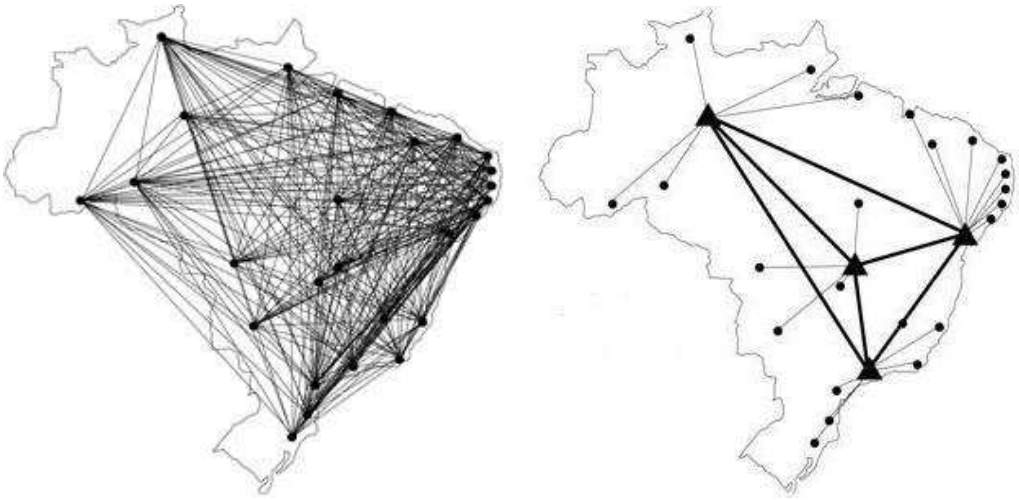
1.1 Sistemas do tipo eixo–raio

Sistemas do tipo eixo–raio (*hub-and-spoke*) (O'KELLY; MILLER, 1994; CAMPBELL; ERNST; KRISHNAMOORTHY, 2002), tornaram-se uma importante área de pesquisa da teoria de localização nas últimas décadas. Esse destaque se deve em grande parte ao sucesso de sua utilização em sistemas logísticos, tanto de transporte de passageiros quanto de cargas, e em redes de telecomunicações. Ao invés de servir cada par origem–destino de demanda com uma conexão direta, sistemas do tipo eixo–raio substituem essas conexões diretas por uma rede de concentradores. Esses concentradores permitem que o tráfego seja agrupado e conduzido através de um meio de transporte compartilhado, para ser então entregue aos respectivos destinos. Esse compartilhamento do meio de transporte permite com que esses sistemas eixo–raio usufruam dos benefícios da economia de escala, isto é, o custo por unidade transportada torna-se menor ao se aumentar o volume do tráfego através do meio de transporte.

A Figura 1(a) ilustra o tipo de conexão direta entre os pares origem–destino, no caso as capitais brasileiras, enquanto que a Figura 1(b) mostra as conexões via sistemas eixo–raio, percebemos que o número de conexões diminui consideravelmente com o uso dos sistemas eixo–raio. Na Figura 1.1, os pontos representam origem e destino, os triângulos são concentradores, enquanto as linhas mais grossas entre concentradores representam as conexões que possuem economia de escala.

Esses problemas diferem dos problemas clássicos de localização em função do papel dos concentradores. Nos problemas clássicos, geralmente, pontos de demanda são atendidos, total ou parcialmente, por facilidades a serem localizadas. A função objetivo, normalmente, é uma função dos custos de transporte dos pontos de demanda e dos custos de instalação das facilidades. Já nos problemas do tipo eixo–raio, os concentradores funcionam como centros de concentração de fluxo. Dessa forma é

permitida a concentração, o redirecionamento e a separação de fluxos de demandas, sendo pontos intermediários no caminho entre pontos de origem e de destino.



(a) (b)

Figura 1: Conexões do tipo direta (a) e eixo–raio (b) entre pares origem–destino. Os pontos(●) representam os pontos de demanda e os triângulos(▲) os concentradores

Os problemas do tipo eixo–raio envolvem a localização dos concentradores e o desenho da rede, isto é, a atribuição das origens e dos destinos a cada concentrador. Dependendo do problema, o número de concentradores, a serem localizados a partir de um conjunto de pontos candidatos, pode ser conhecido previamente ou pode ser também uma incógnita a ser determinada.

Modelos de localização de concentradores abrangem um vasto conjuntos de aplicações, desde problemas de transporte até telecomunicações. Entre os tipos de problemas de transporte estão: o transporte aéreo de passageiros e de cargas (DREZNER; DREZNER, 2001; SASAKI; SUZUKI; DREZNER, 1999; SKORIN-KAPOV, 1998); serviços expressos para entregas, sistemas de transporte de cargas rodoviárias e ferroviárias (TAYLOR et al., 1995; NAGY; SALHI, 2005) e sistemas postais (DONALDSON et al., 1999; ERNST; KRISHNAMOORTHY, 1996, 1999). Nesses casos, a demanda é especificada em termos de fluxos de passageiros ou produtos entre pares de cidades, ou localidades, sendo transportada em algum tipo de veículo (avião, carros, caminhões, etc.) e os concentradores são conhecidos como terminais, ou centros de triagem. A agregação de pequenas quantidades de fluxos em quantidades maiores, o compartilhamento do meio de transporte e, por conseqüência, rateio de custo permitem grande economia de escala.

As aplicações no setor de telecomunicações incluem desde o processamento dis-

tribuído até vídeo–conferência, passando por redes de telefonia pública (GAVISH, 1987; KLINCEWICZ, 1998). Nesses casos, o fluxo é dado em função da transmissão da informação (voz, dados, vídeo) através de uma variedade de meios físicos de transporte ou pelo ar, com enlaces de satélite e de micro ondas. O papel do concentrador nessas redes é o de funcionar como centro de multiplexação, comutação e roteamento, interligando desde pequenas redes corporativas até redes globais.

Como visto, problemas desse tipo envolvem a definição da localização do concentrador e a atribuição dos pares origem–destino a cada concentrador instalado. Devido a sua complexidade e o grande número de aplicações que esses modelos englobam, tornam-se necessárias, técnicas que consigam resolvê-lo de forma eficiente. O objetivo desse trabalho é usar o método de decomposição de Benders (BENDERS, 1962) que é um método eficiente na resolução de problemas de programação linear inteira mista, combinado à técnica de programação paralela, que oferece grande poder computacional para realização dos experimetnos.

Como estamos tratando problemas do tipo eixo–raio faremos uma breve classificação. Em um sistema do tipo eixo–raio, temos três principais elementos: os nós, que são os pontos de origem–destino, de onde o fluxo sai e para onde o fluxo chega; os concentradores, que têm as características de um nó comum, mas neles o fluxo nem termina e nem começa, ele apenas concentra o fluxo; e os arcos que interligam os pares origem–destino e os concentradores.

Considerando a atribuição dos nós a cada concentrador, os problemas eixo–raio podem ser classificados como: problemas de alocação simples, no qual cada ponto só pode ser atendido por um único concentrador, como podemos ver na Figura 2(a); ou problemas de alocação múltipla, no qual cada ponto pode interagir com mais de um concentrador, como podemos ver na Figura 2(b).

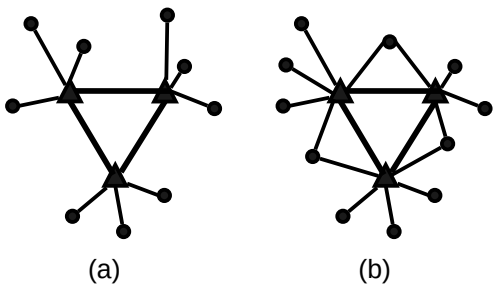


Figura 2: Exemplos de atribuição de nós a concentradores em um sistema eixo–raio

O trabalho de O'Kelly e Miller (O'KELLY; MILLER, 1994) propõe uma taxonomia para os modelo do tipo eixo–raio, classificando os problemas de acordo com o tipo de

atribuição, considerando os seguintes casos:

- se os concentradores estão todos diretamente interconectados;
- se os nós da rede podem comunicar-se diretamente, sem o uso de algum concentrador.

Dessa forma, foram criados oito protocolos exemplificados na Figura 3, que podem ser resumidos da seguinte forma:

- **Protocolo A**: Envolve problemas de localização de concentradores com atribuição simples de nós a concentradores (Figura 3(a));
- **Protocolo B**: Envolve problemas de localização de concentradores com atribuição simples de nós a concentradores e a escolha dos arcos que compõe a rede entre concentradores (Figura 3(b));
- **Protocolo C**: Envolve problemas de localização de concentradores com atribuição simples de nós a concentradores e conexões diretas entre nós que não são concentradores (Figura 3(c));
- **Protocolo D**: Envolve problemas de localização de concentradores com atribuição simples de nós a concentradores, conexões diretas entre nós que não são concentradores e a escolha dos enlaces entre concentradores (Figura 3(d));
- **Protocolo E**: Envolve problemas de localização de concentradores com atribuição múltipla de nós a concentradores (Figura 3(e));
- **Protocolo F**: Envolve problemas de localização de concentradores com atribuição múltipla de nós a concentradores e a escolha dos enlaces entre os concentradores (Figura 3(f));
- **Protocolo G**: Envolve problemas de localização de concentradores com atribuição múltipla de nós a concentradores e conexão direta entre nós que não são concentradores (Figura 3(g));
- **Protocolo H**: Envolve problemas de localização de concentradores com atribuição múltipla de nós a concentradores, conexão direta entre nós que não são concentradores e a escolha dos enlaces que não são concentradores (Figura 3(h)).

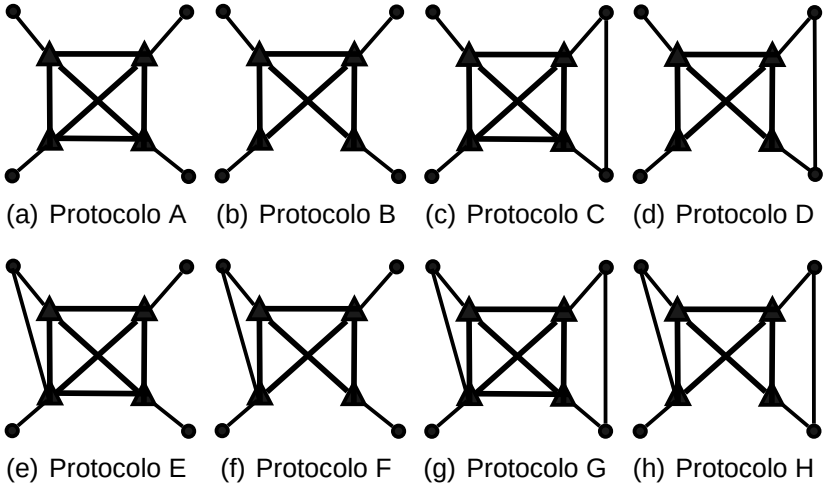


Figura 3: Exemplos de redes dos protocolos A-H

Podemos ver que, os protocolos A-D estão relacionados aos problemas de alocação simples com localização de concentradores, enquanto que os protocolos E-H estão relacionados com os problemas de alocação múltipla. Os problemas de alocação múltipla podem ser reduzidos aos problemas de alocação simples, desde que não se permita que pontos de origem ou destino de demanda sejam conectados a mais de um concentrador.

Além disso, os concentradores e as conexões entre os mesmos podem ser ou não capacitados, isto é, a capacidade de concentração/divisão e transporte de fluxo pode ser limitada.

Uma característica interessante desse problema é que uma determinada rota entre pares de origem–destino pode possuir um ou dois concentradores no máximo. Essa característica deve-se historicamente à linearização proposta inicialmente pelo modelo de O'Kelly (O'KELLY, 1986, 1987).

No presente trabalho foi considerado o problema de localização de concentradores não capacitados com múltipla atribuição de pontos de demanda, utilizando-se o modelo proposto por Skorin-Kapov em (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996a) com pequenas modificações, que será mostrado no Capítulo 2.

1.2 Computação paralela

Processamento paralelo pode ser definido como a manipulação concorrente de dados que pertencem a um ou mais processadores, mas que resolvem um único prob-

lema (QUINN, 1994). Então temos um grande problema dividido em problemas menores, os quais são resolvidos por vários processadores.

Como utilizaremos paralelismo para resolver o problema de localização de concentradores, serão mostrados os conceitos fundamentais de paralelismo utilizados neste trabalho.

Existem diversos campos de aplicação da computação paralela, e sua grande maioria se encontra em simulações e cálculos de experimentos científicos voltados à ciência, isto porque muitos dos problemas científicos são tão complexos que resolvê-los através de simulação numérica requer um extraordinário poder computacional, somente obtido em máquinas paralelas, e se encontram em diversas áreas, tais como (QUINN, 1994):

- Química Quântica, Estatísticas da Mecânica, e Física Relativista;
- Cosmologia e Astrofísica;
- Dinâmica dos Fluídos e Turbulência;
- Projeto de Materiais e Supercondutividade;
- Biologia, Farmacologia, Sequência do Genoma, Engenharia Genética, Cadeias de Proteínas, Atividade Enzimática, e Modelagem de Células;
- Medicina, Modelagem de órgãos e ossos humanos;
- Clima Global e Modelagem de Meio Ambiente.

Além de toda esta gama de aplicações, outro forte motivo para o uso da computação paralela encontra-se na própria Ciência da Computação, pois esta ferramenta é de fundamental importância para a solução em tempo hábil de problemas de otimização, processamento de imagens de grande porte, processamento de grandes massas de dados, entre outros.

Neste trabalho trataremos de um problema de otimização, apresentando uma solução em paralelo a fim de obtermos soluções para problemas maiores, que com a computação seqüencial não seria possível de se resolver.

Quando vamos construir um algoritmo paralelo, a primeira coisa que pensamos é como decompor o problema em problemas menores. Então, os problemas menores são associados a processadores para serem resolvidos de forma simultânea. Existem basicamente dois tipos de decomposição: decomposição de domínio e de dados.

Na **decomposição de domínio**, ou paralelismo de dados, os dados são divididos em pedaços aproximadamente do mesmo tamanho e então distribuídos para diferentes processadores. Cada processador trabalha apenas com uma porção dos dados que foi designado para ele. Então os processadores se comunicam periodicamente para trocar informações. Esse tipo de paralelismo tem a vantagem de manter apenas um único fluxo de controle.

Freqüentemente o paralelismo de dados é a estratégia que mostra-se mais eficiente para programas em paralelo. No caso em que as partes associadas a cada processador requerem tempos de processamento muito diferentes, pode-se obter uma limitação na performance do código devido a velocidade dos processadores mais lentos. Em casos como esse, a **decomposição funcional**, ou paralelismo de tarefas faz mais sentido que a decomposição de domínio, porque nesse tipo de decomposição o problema é dividido em um grande número de tarefas, que são associadas aos processadores ao passo que os mesmos terminam suas tarefas

Neste trabalho usamos a abordagem de decomposição de dados, por se mais eficiente e devido a natureza do problemas, como veremos na seção 3.2.

Um computador paralelo é um computador que tem múltiplos processadores e é capaz de realizar computação paralela. Esse tipo de computadores possuem basicamente dois tipos de arquitetura (GROUP, 2001): memória distribuída e memória compartilhada.

Em arquiteturas com memória distribuída, temos um conjunto de processadores operando de forma cooperativa ou concorrente, na execução de um ou vários aplicativos. Cada nó tem um acesso rápido a sua memória local e acesso as memórias dos demais nós. Os dados são trocados entre os nós como mensagens através da rede.

Em computadores com memória compartilhada, vários processadores compartilham o acesso a uma memória global via um barramento de memória de alta velocidade. Este espaço de memória global permite que os processadores tenham um acesso eficiente aos dados. Nesse tipo de arquitetura o número de processadores usados é limitado, tipicamente variam entre 2 e 16 processadores, isso se deve ao fato da quantidade de dados que pode ser processada é limitada pela banda passante do barramento de memória que liga os diversos processadores.

Neste trabalho utilizaremos um tipo de máquina paralela chamada NOW (*Network of Workstations*), ou Cluster de computadores(COW– *Cluster of Workstations*) que se enquadra na arquitetura de memória distribuída. Essa máquinas são construídas a

partir de computadores comuns (PCs) ligados por redes de interconexão tradicionais de alta velocidade (ANDERSON; CULLER; PATTERSON, 1955). Essas máquinas caracterizam-se pelo fato de que cada processador enxerga somente a sua própria memória. E para a troca de mensagens e dados é preciso o envio de requisições através da rede de interconexão. Com estas características, tais máquinas paralelas podem ser implementadas através de um conjunto de máquinas autônomas, ou seja, computadores tradicionais. A Figura 4 mostra esse tipo de arquitetura, onde podemos ver que cada máquina possui sua memória, processador e entrada e saída próprias ligadas por uma rede. Dessa forma trabalham independentes umas das outras, comunicando-se através de mensagens.

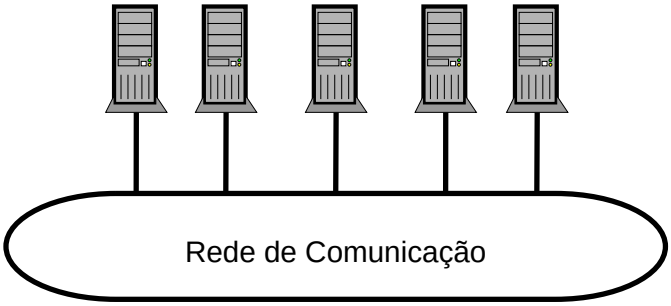


Figura 4: Arquitetura de máquinas NOW

Neste trabalho usamos uma arquitetura com memória distribuída e decomposição funcional para a paralelização do algoritmo, primeiro pela natureza do problema, que adequa-se a esse tipo de arquitetura e também por ser uma estratégia mais eficiente. Na seção 3.1 vamos descrever a solução que foi desenvolvida para paralelizar o algoritmo para a decomposição de Benders apresentada no Algoritmo 1.

1.3 Trabalhos relacionados

Após uma breve descrição dos problemas do tipo eixo raio na seção 1.1, aqui serão apresentados os principais trabalhos relacionados a sistemas eixo–raio, ao método de decomposição de Benders, e ao uso de programação paralela para este método.

Desde o trabalho pioneiro de O'Kelly (O'KELLY, 1986, 1987), um crescente número de pesquisadores vêm propondo abordagens e modelos diferentes para os sistemas eixo–raio. Exames sobre os mais diversos tipos de problemas dentro do contexto podem ser encontrados em (CAMPBELL, 1994a, 1994c; CAMPBELL; ERNST; KRISHNAMOORTHY, 2002).

Uma abordagem comum ao se tratar o problema consiste em decompô-lo em dois subproblemas, um para achar a melhor localização dos concentradores; e outro para identificar o melhor roteamento para o tráfego de cada par origem–destino através dos concentradores. Baseando-se nisso, Klincewicz (KLINCEWICZ, 1991) propõe duas heurísticas diferentes e uma meta-heurística para encontrar boas soluções para cada um dos subproblemas. O trabalho de Skorin- Kapov e Skorin-Kapov (SKORIN-KAPOV; SKORIN-KAPOV, 1994) apresentam um algoritmo baseado em busca tabu e para teste deste método de solução, foi utilizada a mesma base de dados que O'Kelly (O'KELLY, 1987) utilizou, sendo os dados do fluxo aéreo de passageiros entre as 25 maiores cidades dos Estados Unidos em 1970. Os resultados computacionais foram apresentados em subconjuntos de 10, 15, 20 e 25 cidades e 2, 3 e 4 terminais para cada conjunto de cidades. No trabalho de Aykin (AYKIN, 1994) são desenvolvidas 2 abordagens diferentes: um *branch-and-bound* para o problema de localização e atribuição simples e um algoritmo de enumeração para o problema de localização e atribuição múltipla. Em (CAMPBELL, 1996) uma heurística é elaborada para o problema de localização e atribuição múltipla.Ernst e Krishnamoorthy (ERNST; KRISHNAMOORTHY, 1996) usam o resfriamento simulado(*simulating annealing*) para obter bons limites superiores que são usados posteriormente num algoritmo de *branch-and-bound*. Os trabalhos (ABDINNOUR-HELM; VENKATARAMANAN, 1998) e (O'KELLY; SKORIN-KAPOV; SKORIN-KAPOV, 1995) são os primeiros a propor metodologias para se calcular limites inferiores que permitem medir a qualidade das soluções heurísticas.

Em (KLINCEWICZ, 1996) é proposto um método de subida e de ajuste do dual do modelo apresentado no trabalho de Campbell (CAMPBELL, 1996). Skorin Kapov *et.al* (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996b) reformula o modelo de Campbell (CAMPBELL, 1996) obtendo um modelo com uma relaxação linear mais justa. Usando esse novo modelo, Pirkul e Schilling (PIRKUL; SCHILLING, 1998) abordam o problema através de uma otimização baseada no método do subgradiente. Mayer e Wagner (MAYER; WAGNER, 2002) desenvolveram um *branch-and-bound* usando o método de Klincewicz (KLINCEWICZ, 1996) e o modelo desenvolvido por Skorin-Kapov (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996b). O trabalho de (ERNST; KRISHNAMOORTHY, 1996) propõem métodos exatos e heurísticos para uma formulação baseada não mais no par origem–destino, mas apenas na origem. No trabalho (HAMACHER et al., 2000) os autores conseguem combinar duas restrições do modelo originalmente proposto Skorin-Kapov (SKORIN-KAPOV; SKORIN-KAPOV;

O'KELLY, 1996b), obtendo assim uma modelo com conjunto de restrições que definem facetas do poliedro do problema. Essa formulação apresenta atualmente a melhor relaxação linear para o problema.

O método de decomposição de Benders foi apresentado em (BENDERS, 1962), e a partir daí vários problemas foram resolvidos usando o método.Geoffrion e Graves (GEOFFRION; GRAVES, 1974) aplicam o método a sistemas de distribuição e localização de facilidades instaladas entre as fábricas e os clientes finais. A decomposição foi utilizada em problemas do tipo eixo–raio, no trabalho de Miranda (MIRANDA, 2004), onde são solucioanados problemas do tipo *Quadratic Assignment Problem* e problemas de localização de *hubs*.

Existem muitas implementações paralelas de algoritmos exatos e metaheurísticas para se resolver problemas de otimização, tais como: o trabalho Gondzion, Sarkissian e Vial (1998) usa relaxação lagrangeana e decomposição de Benders, para resolver problemas de planejamento de energia, modelos de decisão de Markov e problemas em telecomunicações, juntamente com o método de plano de corte com centro analítico (*Analytic Center Cutting Plane Method*) e programação paralela. O algoritmo é executado em duas plataformas diferentes de computação paralela e são obtidos bons *speedups* para a classe de problemas tratada. Nielsen e Stavros (NIELSEN; ZENIOS, 1997) apresentam uma implementação paralela do método de decomposição de Benders para um programa linear estocástico dois estágios que alivia a dificuldade de balanceamento de carga. O trabalho de MirHassani *et.al* (MIRHASSANI et al., 2000) considera duas abordagens de modelo e técnicas de solução para o problema de planejamento de produção com incertezas, a primeira solução envolve o cenário de soluções *wait and see* e a segunda envolve programação inteira estocástica. O método de decomposição de Benders é utilizado e são feitas duas implementações do algoritmo, uma seqüencial e outra paralela.

Não foram encontrados na literatura trabalhos utilizando algoritmos paralelos em aplicações do tipo eixo–raio com decomposição de Benders.

1.4 Estrutura do trabalho

O texto está dividido em cinco capítulos. O presente Capítulo, apresenta uma visão geral do trabalho, descreveu problemas do tipo eixo–raio e conceitos sobre paralelismo, motivação, objetivos e revisão bibliográfica. O Capítulo 2 descreve o modelo matemático

utilizado para o problema tratado, o método de decomposição de Benders aplicado a esse modelo e o algoritmo de decomposição de Benders. O Capítulo 3 apresenta a solução em paralelo para o problema e um estudo sobre os benefícios da paralelização do algoritmo seqüencial. O Capítulo 4 mostra os resultados alcançados pela solução apresentada fazendo uma comparação com os resultados seqüenciais. O Capítulo 5 faz a conclusão do trabalho mostrando as principais contribuições e possíveis trabalhos futuros.

2 DECOMPOSIÇÃO DE BENDERS PARA O MODELO DE SKORIN-KAPOV

Vários modelos foram propostos para problemas de localização de concentradores, desde o pioneiro trabalho de O´Kelly (O'KELLY, 1987), que introduziu os primeiros conceitos dos sistemas eixo–raio propondo uma formulação não-linear inteira para o problema. Seguindo o estudo de O'Kelly, Campbell (CAMPBELL, 1994b) introduz os conceitos iniciais de que cada par origem destino deve selecionar a rota que apresenta o melhor custo benefício via os concentradores e criou um modelo linear. Esses conceito mais geral introduziu os modelos de localização de concentradores com atribuição múltipla. Em seguida Skorin-Kapov (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996a) propôs modificações no modelo de Campbell uma relaxação mais justa do que a original. Isto porque os espaço de soluções no modelo de de Skorin-Kapov satisfaz as restrições do modelo de Campbell, mas não virse-versa. Intuitivamente, espera-se que as variáveis y terão valores fechados para os valores integrais.

Deste Capítulo constará o modelo de Skorin-Kapov utilizado neste trabalho para problemas do tipo eixo–raio e o método de decomposição de Benders aplicado ao mesmo. Nesse tipo de problema, se todos os nós são concentradores candidatos, o número de variáveis de fluxo é da ordem de n^4 variáveis, dessa forma o método de Benders torna-se eficiente na resolução do problema já que permite sua separação em problemas menores, reduzindo um problema de programação linear inteira mista em simples problemas de transporte. A seção 2.1 apresenta o modelo utilizado. A seção 2.2 discorre sobre o método de decomposição de Benders. Na seção 2.3 é mostrado o algoritmo e sua análise e na última seção as considerações finais.

2.1 Modelo de Skorin-Kapov

Nessa seção apresentaremos a formulação para o modelo de Skorin-Kapov (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996b). Consideramos aqui os problemas de

localização de concentradores, não capacitados e com múltipla atribuição, dessa forma vamos enunciá-lo como segue:

Definição 1. *Dados um conjunto de pontos, as demandas de cada par origem–destino e um conjunto de concentradores candidatos, o problema consiste em localizar os concentradores atribuindo os pontos de origem e de destino aos mesmos, de forma que o custo total seja mínimo e que a rota entre cada par de origem-destino passe por 1 ou 2 concentradores. O custo total é a composição dos custos de instalação dos concentradores e de transporte.*

Para a compreensão da formulação aqui apresentada, vamos considerar as seguintes definições:

- Seja N o conjunto de nós, isto é, pontos de origem e de destino;
- Seja K o conjunto de nós candidatos a concentradores, tal que $K \subseteq N$.
- Para qualquer par de nós i e j, onde $i, j \in N$, representando um ponto de origem e de destino respectivamente, tem-se w_{ij}, a demanda de fluxo do nó i para o nó j, sendo $w_{ij} \neq w_{ji}$.
- Sejam a_k o custo de instalação de um concentrador no nó $k \in K$ e c_{ijkm} o custo unitário de transporte do nó i até o nó j através dos concentrador k e m, tal que $i, j \in N$ e $k, m \in K$. A Figura 5 mostra o custo unitário de transporte como a composição de três segmentos do caminho do nó i até o nó j, então temos que $c_{ijkm} = c_{ik} + bc_{km} + c_{mj}$, onde:
  - c_{ik} e c_{mj} são os custos unitário de transporte do nó $i(j)$ até o nó $k(m)$.
  - bc_{km} é o custo unitário de transporte com desconto entre os concentradores k e m, representando a economia de escala, com $b \leq 1$;
- Se apenas um concentrador é usado, tem-se $k = m$, nesse caso $c_{mm} = 0$;

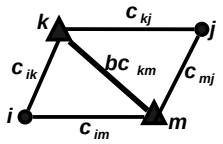


Figura 5: Representação dos segmentos de custo.

O problema que estamos considerando aqui, consiste em decidir onde os concentradores devem ser instalados e como o fluxo entre cada par de nós deve ser roteado, dessa forma temos que as variáveis de decisão do modelo são:

- $y_k = \begin{cases} 1, \text{se o nó } k \text{ é instalado como concentrador} \\ 0, \text{caso contrário} \end{cases}$
- x_{ijkm} é o fluxo do nó i até o nó j que é roteado via os concentradores k e m nessa ordem.

O roteamento via concentradores para cada par de origem–destino é feito de forma implícita, através da variável x_{ijkm}. Além disso, originalmente, o modelo proposto por (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996a) não contemplava o custo de instalação dos concentradores, requerendo o conhecimento a priori do número de concentradores a serem instalados. Conforme argumentado por (CAMPBELL, 1994b), esse conhecimento exógeno nem sempre é coerente com a realidade, sendo mais interessante ser computado endogenamente ao se incorporar os custos de instalação dos concentradores. Nesse caso, o modelo de Skorin-Kapov (SKORIN-KAPOV; SKORIN-KAPOV; O'KELLY, 1996a) é reescrito da seguinte forma:

$$Min\left[\sum_{k\in K} a_k y_k + \sum_{i\in N}\sum_{j\in N}\sum_{k\in K}\sum_{m\in K} c_{ijkm} x_{ijkm}\right] \tag{2.1}$$

Sujeito a

$$\sum_{k\in K} x_{ijkm} \leq w_{ij} y_m \qquad \forall i,j \in N; m \in K \tag{2.2}$$

$$\sum_{m\in K} x_{ijkm} \leq w_{ij} y_k \qquad \forall i,j \in N; k \in K \tag{2.3}$$

$$\sum_{k\in K}\sum_{m\in M} x_{ijkm} = w_{ij} \qquad \forall i,j \in N \tag{2.4}$$

$$x_{ijkm} \geq 0 \qquad \forall i,j \in N; k,m \in K \tag{2.5}$$

$$y_k \in \{0,1\} \qquad \forall k \in K \tag{2.6}$$

A função objetivo 2.1 é composta por dois termos, o primeiro representa o custo de instalação dos concentradores, enquanto que o segundo, é o custo total de transporte. O conjunto de restrições 2.2 e 2.3 garantem que as demandas dos pares origem–destino só são roteadas por concentradores instalados. A restrição 2.4 assegura que as demandas dos pares origem–destino são roteadas via algum par de concentradores. Quando $k = m$, o roteamento só acontece via um único concentrador.

E por fim, as restrições 2.5 e 2.6 são as de não negatividade e de integralidade, respectivamente.

Considerando n=—N—, observa-se que o modelo possui n^4 variáveis de fluxo x_{ijkm}. Quando esse valor é comparado com o número de variáveis inteiras y_k, constata-se uma enorme diferença de magnitude, mesmo para valores pequenos de n. A fixação do número relativamente pequeno de variáveis inteiras induz uma inerente separabilidade desse problema de grande porte. Essa característica foi uma das motivações para a aplicação do método de decomposição de Benders (BENDERS, 1962). Na seção 2.2 veremos como é feita essa decomposição usando o método de Benders.

2.2 Método de decomposição de Benders

O método de decomposição de Benders (BENDERS, 1962; GEOFFRION, 1972) utiliza a teoria da dualidade (GOLDBARG; LUNA, 2000) em programação matemática (linear e não linear), para particionar um problema de difícil solução quando são consideradas todas as variáveis envolvidas, aplicando os conceitos de dualização e linearização externa, em subproblemas com variáveis específicas, que são resolvidas interativamente até que uma solução ótima seja obtida para o problema completo.

Originalmente desenvolvido para problemas de programação linear inteira mista, o método também é capaz de resolver problemas não- lineares, desde que a não linearidade seja incorporada à decomposição de forma apropriada (GEOFFRION, 1972). Esse método tem sido empregado com sucesso, ao longo das décadas, na resolução de problemas de desenho de sistemas de distribuição multiproduto de larga escala (GEOFFRION; GRAVES, 1974; MAGNANTI; WONG, 1984; MIRANDA, 2004). Usaremos o método aqui para resolver um problema de programação linear inteira mista, com variáveis 0/1 e variáveis de fluxo.

No método, o problema original é decomposto em dois problemas chamados de problema mestre e subproblema, que serão formulados nas seções 2.2.1 e 2.2.2, respectivamente. O mestre possui essencialmente as variáveis y_k sendo responsável pela instalação dos concentradores e pela obtenção de um limite inferior para o problema. Por outro lado, o subproblema possui apenas as variáveis de fluxo x_{ijkm}, divididas por pares de origem–destino, que são responsáveis por decidir como o fluxo entre cada par de nós é roteado. A resolução do subproblema obtém um limite superior e desigualdades válidas que são adicionadas ao problema mestre a cada iteração do

algoritmo. Quando o limite superior torna-se igual ao limite inferior, uma solução ótima para o problema é encontrada.

2.2.1 Problema mestre

Considerando a formulação (2.1)–(2.6), vamos determinar o problema mestre, que tem a função de instalar os concentradores para que os subproblemas resolvam os problemas de fluxo entre os pares origem–destino, portanto o problema mestre possui apenas as variáveis y_k.

Então em uma dada iteração h fixando-se o vetor de variáveis binárias, $y = y^h$, que é resolvido no problema mestre, podemos escrever o problema primal como:

$$Min \sum_{i \in N} \sum_{j \in N} \sum_{k \in K} \sum_{m \in K} c_{ijkm} x_{ijkm} \tag{2.7}$$

Sujeito a

$$\sum_{k \in K} x_{ijkm} \leq w_{ij} y_m^h \qquad \forall i, j \in N; m \in K \tag{2.8}$$

$$\sum_{m \in K} x_{ijkm} \leq w_{ij} y_k^h \qquad \forall i, j \in N; k \in K \tag{2.9}$$

$$\sum_{k \in K} \sum_{m \in M} x_{ijkm} = w_{ij} \qquad \forall i, j \in N \tag{2.10}$$

$$x_{ijkm} \geq 0 \qquad \forall i, j \in N; k, m \in K \tag{2.11}$$

Para formular o problema mestre precisamos encontrar o dual do problema primal acima, então vamos associar o conjunto de variáveis duais u_{ijk}^h, n_{ijm}^h e r_{ij}^h às restrições 2.8, 2.9 e 2.10, respectivamente. Pois para cada restrição no problema primal, temos uma variável no problema dual.

Pode-se escrever o dual do subproblema (2.8)–(2.11) como o conjunto de equações (2.12)–(2.16):

$$Max \left[w_{ij} r_{ij} - \sum_{i \in N} \sum_{j \in N} \sum_{k \in K} w_{ij} y_k^h u_{ijk} - \sum_{i \in N} \sum_{j \in N} \sum_{m \in K} w_{ij} y_m^h n_{ijm} \right] \tag{2.12}$$

Sujeito a

$$r_{ij} - n_{ijm} - u_{ijk} \leq c_{ijkm} \qquad \forall i,j \in N; k,m \in K \qquad (2.13)$$

$$r_{ij} \in R \qquad \forall i,j \in N \qquad (2.14)$$

$$u_{ijk} \geq 0 \qquad \forall i,j \in N; k \in K \qquad (2.15)$$

$$n_{ijm} \geq 0 \qquad \forall i,j \in N; m \in K \qquad (2.16)$$

O problema mestre formado têm na função objetivo uma variável que chamamos de subsestimativa dos custos de transporte e as variáveis y_k. As restrições são derivadas do problema dual dado acima. Dessa forma o problema mestre pode ser escrito como (2.17)-(2.20):

$$Min\left[\eta + \sum_{k \in K} a_k y_k\right] \qquad (2.17)$$

Sujeito a

$$\eta \geq \sum_{i \in N}\sum_{j \in N} w_{ij} r^h_{ij} - \sum_{i \in N}\sum_{j \in N}\sum_{k \in K} w_{ij} u^h_{ijk} y_k - \sum_{i \in N}\sum_{j \in N}\sum_{k \in K} w_{ij} n^h_{ijm} y_m \qquad \forall h \in H \qquad (2.18)$$

$$\sum_{k \in K} y_k \geq 1 \qquad (2.19)$$

$$y_k \in \{0,1\} \qquad \forall k \in K \qquad (2.20)$$

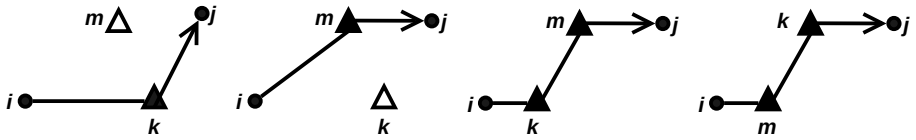
$$\eta \geq 0 \qquad (2.21)$$

Onde h representa a iteração de Benders. A função objetivo 2.17 possui os custos de instalação e uma variável η , veja restrição 2.21, subestimadora do custo de transporte. A restrição 2.19 apenas determina que pelo menos um concentrador deve ser instalado, garantindo a viabilidade de todas as soluções propostas pelo problema mestre. As restrições 2.18 são conhecidas como os cortes de Benders do tipo I, sendo os valores de: u^h_{ijk}, n^h_{ijm} e r^h_{ij}, calculados de forma eficiente através da resolução do subproblema, como veremos a seguir.

2.2.2 Subproblemas

A determinação da solução ótima no nível inferior, em uma dada iteração h, envolve o roteamento de menor custo de transporte para cada par ij de origem–destino. Isto é, pode-se dividir a resolução nesse nível em um subproblema para cada par ij. Lembrando que esse roteamento é feito via um ou dois concentradores, então existem

quatro situações possíveis de roteamento para um dado par ij e para uma dada configuração de concentradores instalados. A Figura 6 ilustra as quatro situações possíveis: em 6(a) e 6(b) a rota de menor custo pode passar por apenas um concentrador instalado, k ou m ; em 6(c) e 6(d) a rota de menor custo pode passar por dois concentradores instalados, tanto no sentido concentrador $k \rightarrow m$ ou, o contrário, $m \rightarrow k$.



(a) (b) (c) (d)

Figura 6: Rotas possíveis para o par origem-destino ij, para um dado y^h

A solução ótima para o subproblema primal dado pelas equações (2.7)–(2.11) é dada pela equação:

$$\sum_{i,j \in N} w_{ij} Min_{k,m \in K} \{ c_{ijkm} \mid y_k^h = y_m^h = 1 \} \tag{2.22}$$

Além disso, os valores ótimos das variáveis duais devem ser obtidos para a adição do corte de Benders do tipo I ao problema mestre. A obtenção dos valores das variáveis duais, u_{ijk}^h, n_{ijm}^h e r_{ij}^h pode ser feito através da interpretação econômica de cada variável e pela propriedade das folgas complementares (LUNA, 1978). Essa interpretação permite implementações eficientes na resolução do problema. Considerando v_{ij}^h como a maior diferença de preço possível entre os pontos i e j que se está disposto a pagar, tem-se que v_{ij}^h é o menor custo de transporte entre i e j, dada pela equação:

$$r_{ij}^h = Min_{k,m \in K} \{ c_{ijkm} \mid y_k^h = y_m^h = 1 \} \tag{2.23}$$

Interpretando u_{ijk}^h e n_{ijm}^h como taxas adicionais a serem pagas para se rotear o fluxo de i até j via os concentradores k e m, têm-se, que quando os concentradores k e m estão instalados $(y_k^h = y_m^h = 1)$, essas taxas são iguais a zero, como mostram as equações a seguir:

$$u_{ijk}^h = 0, \quad se \quad y_k^h = 1 \tag{2.24}$$

$$n_{ijm}^h = 0, \quad se \quad y_m^h = 1 \tag{2.25}$$

Por outro lado, se os concentradores não estão instalados, deve-se analisar qual é a contribuição marginal de cada concentrador, isto é, qual deles ofereceria a menor taxa para se rotear o fluxo. Essas taxas são dadas pelas relações:

$$u_{ijk}^{h} = Max\ \{0, Max_{m \in M}\{r_{ij}^{h} - c_{ijkm}\}\},\ \ se\ \ y_{k}^{h} = 0 \tag{2.26}$$

$$n_{ijm}^{h} = Max\ \{0, Max_{k \in K}\{r_{ij}^{h} - c_{ijkm}\}\},\ \ se\ \ y_{m}^{h} = 0 \tag{2.27}$$

Dessa forma, consegue-se resolver o subproblema adicionando os cortes de Benders ao problema mestre. É importante observar que devemos esperar que a maior parcela de custo da solução se deva aos subproblemas, uma vez que para cada par origem–destino, haverá a necessidade de se determinar a rota de custo mínimo entre os diversos concentradores alocados pelo mestre.

2.3 Algoritmo de decomposição de Benders

Dada a formulação matemática mostrada na seção anterior, apresentaremos aqui o algoritmo utilizado na resolução computacional do problema, sua complexidade, limitações e um exemplo de sua execução.

2.3.1 Algoritmo

A aplicação do método de decomposição de Benders ao modelo dado pelas equações (2.1)–(2.6) conduz ao Algoritmo 1, para cuja compreensão vamos considerar a descrição das variáveis utilizadas, mostrada na Tabela 1.

Para a execução do algoritmo temos os seguintes dados de entrada:

- O número de nós (N);
- Uma matriz de custos de transporte entre os pares ij (c_{ijkm}) dos pares origem–destino;
- As demandas (w_{ij});
- Os custos de instalação dos concentradores (a_k);
- O desconto (b).

As variáveis $CF, LS, LI, ls, li, \eta, \mu$ e v são modificadas a cada iteração do algoritmo.

Variável	Descrição
N	Número de nós do problema
K,M	Conjunto de nós candidatos a concentradores
LI	Limite Inferior do custo de transporte e instalação dos concentradores
LS	Limite superior do problema do custo de transporte e instalação dos concentradores
ls	Limite superior da iteração
CF	Custo total do nós que são concentradores em uma iteração h
CT	Custo de transporte do nó i ao nó j
η	Subestimadora do custo de transporte no problema mestre
a_k	Custo de instalação do concentrador k
y	Vetor que contém os nós
c_{ijkm}	Custo de transporte do nó i ao nó j passando pelos concentradores k e m
x_{ijkm}	Quantidade de fluxo do nó i até o nó j que roteado via os concentradores k e m
w_{ij}	Demanda de fluxo do nó i ao nó j
μ e v	Variáveis duais
i, j, k *e* m	Variáveis de controle

Tabela 1: Descrição das variáveis do algoritmo

Os dados de saída do algoritmo são:

- A configuração final dos nós y, ou seja quais os nós que deverão ser instalados de forma a obter o menor custo de transporte entre os pares ij;
- O custo fixo total (CF), dada a configuração da rede e o custo de transporte total (CT).

Algoritmo 1 Decomposição de Benders

1: $LI \leftarrow 0$;
2: $CF \leftarrow \sum_k a_k$;{onde k refere-se os nós que são concentradores}
3: $CT \leftarrow$ Resolve subproblema (2.7)–(2.11)
4: $LS \leftarrow CF + CT$;
5: Adiciona o corte ao problema mestre (2.18);
6: **while** $LS \neq LI$ **do**
7: $LI \leftarrow$ Resolve problema mestre (2.17)–(2.21)
8: $CF \leftarrow \sum_k a_k$;{onde k são os nós que são concentradores}
9: $CT \leftarrow$ Resolve subproblema (2.7)–(2.11)
10: $ls \leftarrow CF + CT$;
11: **if** $LS < ls$ **then**
12: $LS \leftarrow ls$;
13: **end if**
14: Adiciona o corte ao problema mestre (2.18);
15: **end while**

As linhas 1 a 5 do Algoritmo 1 fazem todas as inicializações das variáveis para a primeira iteração do algoritmo. A linha 1 inicia o limite inferior com 0. Para que se tenha o valor do custo fixo é necessário estabelecer uma configuração inicial da rede, ou seja, quais nós serão instalados como concentradores. Nesse caso a configuração inicial utilizada foi a instalação de todos os nós como concentradores, assim na linha 2

o custo fixo é inicializado com a soma dos custos de instalação dos nós que foram instalados como concentradores. Na linha 3 temos a resolução do subproblema que obtém o menor custo de transporte, dada a configuração inicial da rede, cuja solução é dada pelo conjunto de equações (2.22)-(2.27). Na linha 4 o valor do limite superior é calculado, sendo a soma dos custos totais de instalação e transporte. Na linha 5 é adicionado o corte de Benders.

Da linha 6 a 15 temos um laço que implementa os ciclos de Benders. O algoritmo pára quando os limites inferior e superior convergem. Na linha 7 o problema mestre é resolvido e o valor do limite inferior para o problema é obtido. As linhas 8, 9 e 10 são correspondentes as linhas 2, 3 e 4 do algoritmo, considerando a configuração dos concentradores na iteração corrente que é representada por h, que nos dá o custo de transporte para se rotear o fluxo entre os pares ij.

Na linha 12 é feita a atribuição do novo valor para o limite superior, verificando se o valor atual é melhor do que o anterior. A linha 14 adiciona o corte de Benders ao problema mestre. Voltamos ao início da repetição que para quando os limites superior e inferior são iguais.

É importante observar que devemos esperar que a maior parcela de tempo da execução do algoritmo se deva a resolução dos subproblemas, uma vez que para cada par origem–destino haverá a necessidade de se determinar a rota de custo mínimo entre os diversos concentradores alocados pelo mestre. Essa foi uma característica importante para se obter efciência na solução em paralelo.

2.3.2 Execução do algoritmo

Para ilustrar a execução do Algoritmo 1 vamos considerar um exemplo simples, com os seguintes dados de entrada:

- Número de nós (N) igual a cinco;
- Uma matriz de custos de transporte entre os pares ij;
- Uma matriz de demandas;
- Os custos de instalação dos concentradores.

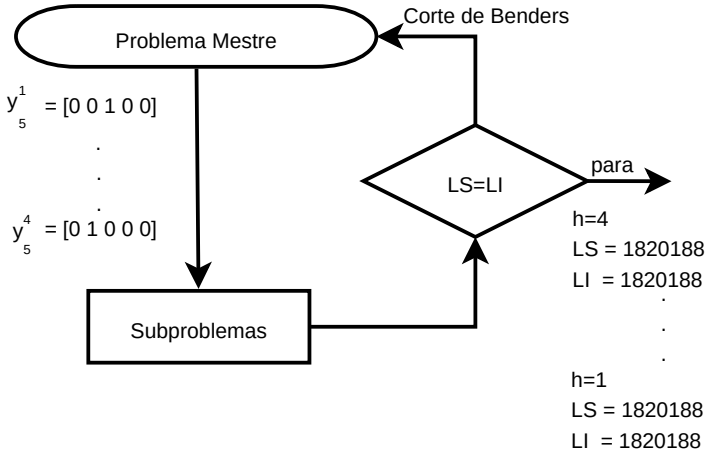


Figura 7: Ilustração do algoritmo de Benders

A execução do algoritmo é ilustrada na Figura 7. No nível superior é estabelecida uma configuração para os nós na primeira iteração, $y_5^1 = [00100]^*$, enviando esses dados para o nível inferior são resolvidos os subproblemas e um novo limite superior estabelecido e comparado com o limite inferior, esses passos são repetidos até que esses valores se igualem, nesse caso na iteração quatro. Os valores obtidos estão na Tabela 2.

h	y^h	LI	LS	CF	CT	CF/CT
1	[0 0 1 0 0]	1025929.8000	4278273.0000	360325.0000	3917948.0000	0.0920
2	[0 1 0 0 0]	1227762.8000	1820188.0000	562158.0000	1258030.0000	0.4469
3	[0 1 1 0 0]	1588087.8000	1841617.0000	922483.0000	919134.0000	1.0036
4	[0 1 0 0 0]	1820188.0000	1820188.0000	562158.0000	1258030.0000	0.4469

Tabela 2: Execução do algoritmo

A seguir é feita a análise de complexidade de tempo para o algoritmo.

*y_5^1 significa o valor do vetor y, que tem 5 nós na iteração 1

2.4 Análise do algoritmo

Para analisar a complexidade do Algoritmo 1 vamos recordar as seguinte definições dadas na seção 2.1: k é o número de nós candidatos a concentradores e N o número de nós, com $k \subseteq N$. Então no pior caso temos $k = N$. Dessa forma temos: na linha 7 no problema mestre temos complexidade de $O(n)$, na linha 8 temos complexidade de $O(n)$, na linha 9 temos a execução dos subproblema que é $O(n^4)$ e na linha 14, o corte de Benders temos $O(n^3)$, dessa forma temos que a complexidade do algoritmo é $O[h(2n+n^4+n^3)]$, onde h é o número de ciclos de Benders. Desconsiderando-se h por ser uma constante e tomando-se apenas o maior dos termos temos que a complexidade do algoritmo seqüencial é $O(n^4)$.

Algoritmos de $O(n^4)$ são complicados de resolver quando o tamanho do problema cresce , por exemplo, para um problema com dez nós, ou seja, $N = 10$ temos um tempo de execução da ordem de 10^2 u.t., se esse número for aumentado para $N = 100$ percebemos que o tempo aumenta consideravelmente ficando na ordem de 10^8 u.t. o que representa um tempo de execução muito alto.

Considerando a natureza do método utilizado para se resolver o problema, percebe-se que é embaraçosamente paralelizável, dessa forma podemos obter bons tempos de execução para a resolução de grandes problemas, que as vezes não conseguimos resolvê-los com o algoritmo seqüencial.

2.5 Considerações finais

O Capítulo mostrou o modelo que é utilizado, o método de decomposição de Benders aplicado a esse modelo e o algoritmo para sua execução seqüencial. Como vimos, o problema possui um grande número de subproblemas e com o uso de um método de decomposição, instâncias maiores são resolvidas. No Capítulo 3 é apresentada a solução em paralelo do método, bem como o algoritmo utilizado na implementação e um estudo sobre a eficiência do algoritmo.

3 IMPLEMENTAÇÃO EM PARALELO DO MÉTODO DE DECOMPOSIÇÃO DE BENDERS

O método de decomposição de Benders, visto no Capítulo 2 é uma técnica adequada à paralelização, por permitir a decomposição de problemas separáveis. Este Capítulo mostra a solução em paralelo utilizada e um estudo do algoritmo paralelo.

3.1 Solução em paralelo da decomposição de Benders

Como vimos anteriormente, para paralelizar um problema devemos dividi-lo em problemas menores, de forma que estes sejam resolvidos pelos processadores de forma conjunta, com o objetivo de obter uma solução em um curto espaço de tempo. Para compreender a solução aqui adotada, vamos considerar que P é o número de processos utilizados na execução do programa, sendo um deles chamado de processo mestre, ou processo 0, e P-1 é o número de processos escravos, enumerado de 1 até P-1, que podemos chamar de arquitetura cliente–servidor, como mostra a Figura 8. Nosso problema consiste em localizar os concentradores, considerando que é dado

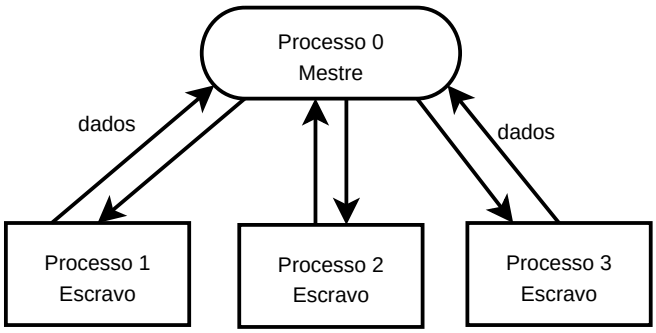


Figura 8: Arquitetura cliente–servidor

um conjunto de nós, as demandas de cada par origem-destino e um conjunto de concentradores candidatos, atribuindo os pontos de origem e de destino aos mesmos, de

forma que o custo total seja mínimo e que a rota entre cada par de origem-destino passe por um ou dois concentradores.

Dessa forma, como o problema é decomposto em problemas menores pelo método utilizado, usaremos essa característica para paralelizar o nosso algoritmo. Então a solução em paralelo consiste em dividir o problema entre os processadores, da seguinte forma: o problema mestre, representado pelas equações (2.17)-(2.6) é resolvido no processo zero e os subproblemas, o conjunto de equações (2.12)-(2.16), são distribuídos aos processos escravos.

Então temos que cada processo escravo resolve problemas de roteamento de fluxo para um par ij, levando em conta que quando um processo termina é atribuído um novo subproblema a ele.

Na solução em paralelo, o processo 0 é responsável por resolver o problema mestre e adicionar o corte de Benders, estabelecendo um limite inferior para o problema e a configuração dos concentradores, isto é, quais nós são instalados como concentradores. A nova configuração da rede em uma iteração h, y^h, é enviada para os processos. Quando o subproblema é resolvido ele obtém o limite superior e envia os valores das variáveis duais, para o processo mestre, que adiciona o corte de Benders e envia as novas informações para os processos escravos. O número de processos, nesse caso, é fixo, dessa forma a distribuição dos pares ij é feita de modo que cada processo escravo receba uma quantidade de subproblemas.

Como estamos utilizando a abordagem de paralelismo de dados, então temos que dividir os dados entre os processadores, com blocos de dados aproximadamente iguais. Nesse caso, os pares ij estão sendo divididos entre os processadores, isto é, as matrizes que são dadas como entrada para o problema são divididas por linha para cada processador.

O número de linhas a serem enviadas QL para cada processador é determinado pela razão entre o tamanho do problema(N) e o número de processadores utilizados(P), ou seja $QL = N/P$. Dessa forma se tivermos o número de nós igual a dez e o número de processadores igual a cinco, então cada processo vai receber duas linhas da matriz para, pois $10/5 = 2$. Para facilitar a implementação do algoritmo, a solução aqui exposta exige que o número de nós seja múltiplo do número de processos utilizados.

Essas linhas representam os dados dos subproblemas que serão resolvidos pelos processadores. Então a paralelização do algoritmo permite a resolução de vários subproblemas ao mesmo tempo, resultando em um tempo de execução menor difer-

entemente da computação seqüencial, onde apenas um subproblema para um par ij é resolvido por vez. Na seção 3.2, que segue, é mostrado o algoritmo utilizado.

3.2 Algoritmo paralelo

Como vimos na seção 1.2 em máquinas NOW a comunicação entre os processadores são baseados no envio e recebimento de mensagens entre os processadores. Essa mensagens podem proceder de várias formas, tais como um único processador envia a mensagem e outro recebe, um único processador envia a mensagem para todos os outros, vários processadores enviam a mensagem para um único processador, etc. Neste trabalho dois tipos de mensagens foram utilizadas:

- Mensagens enviadas de vários processos para um único processador, onde operações de reduções são utilizadas;
- Mensagens enviadas de um processo para vários processos, o que podemos chamar de *broadcast*.

O Algoritmo 2 é uma versão em paralelo do Algoritmo 1 mostrado na seção 2.3.1, dessa forma vamos considerar todas as variáveis definidas na Tabela 1. Para implementação em paralelo, foram necessárias novas variáveis, que estão descritas na Tabela 3.

Variável	Descrição
P	Número de processos utilizados
QL	Quantidade de linhas a serem resolvidos por cada processo ESCRAVO
processo	Número do processo corrente

Tabela 3: Descrição das variáveis adicionais para o algoritmo paralelo

Como no Algoritmo 1 apresentado na seção 2.3.1, temos a partir da linha 1 até a linha 10 todas as inicializações das variáveis para a primeira iteração do algoritmo. A linha 2 define a quantidade de linhas da matriz de entrada que serão resolvidos por cada processo ESCRAVO, o primeiro somatório da função objetivo 2.7 do subproblema é dividido entre os provessadores seguindo a seguinte fórmula $i = processo * QL$. Na linha 3 o custo fixo é inicializado com a soma dos custos de instalação dos nós que foram instalados como concentradores. Na linha 4 temos a resolução do subproblema que obtém o menor custo de transporte em todos os processos, o índice do primeiro somatório faz a distribuição entre os processos variando de acordo com o processo

Algoritmo 2 Decomposição de Benders em paralelo

1: $LI \leftarrow 0$;
2: $QL \leftarrow N/P$; {Definindo a quantidade de dados a enviar para cada processador}
3: $CF \leftarrow \sum_k a_k$;{onde k são os nós que são concentradores}
4: $CT \leftarrow$ Resolve subproblema (2.7)–(2.11){Nesse momento os subproblemas estão sendo resolvidos em paralelo}
5: Envia os valores de u, n, r e CT dos processos ESCRAVOS para o MESTRE;
6: **if** MESTRE **then**
7: $LS \leftarrow CF + CT$;
8: Adiciona o corte ao problema mestre (2.13);
9: **end if**
10: Enviar LS do MESTRE para os processos ESCRAVOS;
11: **while** $LS \neq LI$ **do**
12: **if** MESTRE **then**
13: Resolve problema mestre (2.17)–(2.21)
14: $CF \leftarrow \sum_k a_k$;{onde k são os nós que são concentradores}
15: **end if**
16:
17: $CT \leftarrow$ Resolve subproblema (2.7)–(2.11){Nesse momento os subproblemas estão sendo resolvidos em paralelo}
18: Envia os valores de u, n, r e CT dos processos ESCRAVOS para o MESTRE;
19: **if** MESTRE **then**
20: $ls \leftarrow CF + CT$;
21: **if** $LS < ls$ **then**
22: $LS \leftarrow ls$;
23: **end if**
24: Adiciona o corte ao problema mestre (2.13);
25: **end if**
26: Enviar LS e LI do MESTRE para os processos ESCRAVOS;
27: **end while**

corrente, ou seja, se tivermos $processo = 1$ e $QL = 1$, então a linha um será resolvida pelo processo um. Depois que todos os subproblemas são calculados em paralelo, os valores de u, n, r e CT são atualizados no MESTRE, isso é feito na linha 5. A seguir, a atribuição do novo limite superior e o corte de Benders são feitos pelo MESTRE, linhas 7 e 8, respectivamente.

Da mesma forma que no algoritmo seqüencial, a partir da linha 11 até 27 temos um laço que implementa os ciclos de Benders, que para quando os limites inferior e superior convergem. Na linha 13 é calculado o problema mestre apenas no processo MESTRE e um limite inferior para o problema é obtido. As linhas 14 e 17 são correspondentes as linhas 3 e 4 do algoritmo, sendo nesse caso o CF calculado apenas no MESTRE, com a nova configuração da rede que foi determinada pela resolução do problema mestre. Na linha 18 os valores de u, v e CT são atualizados no MESTRE.

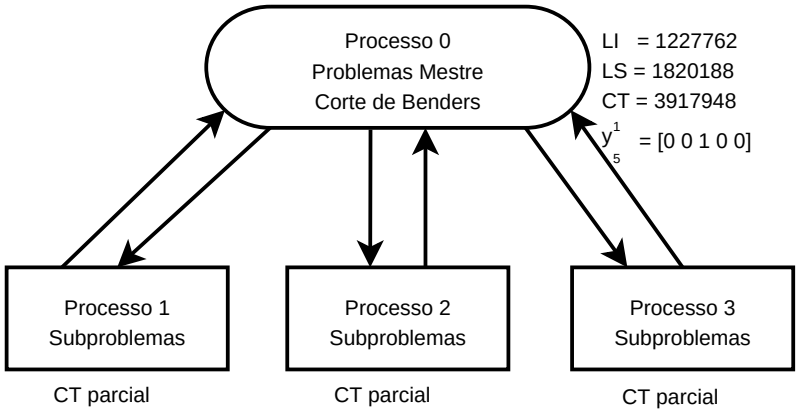
A comparação com o limite superior anterior e a atribuição do novo limite superior é feita apenas no mestre nas linhas 19 e 22. A linha 24 adiciona o corte de Benders ao problema mestre. Se os valores dos limites superior e inferior são diferentes voltamos

ao início da repetição.

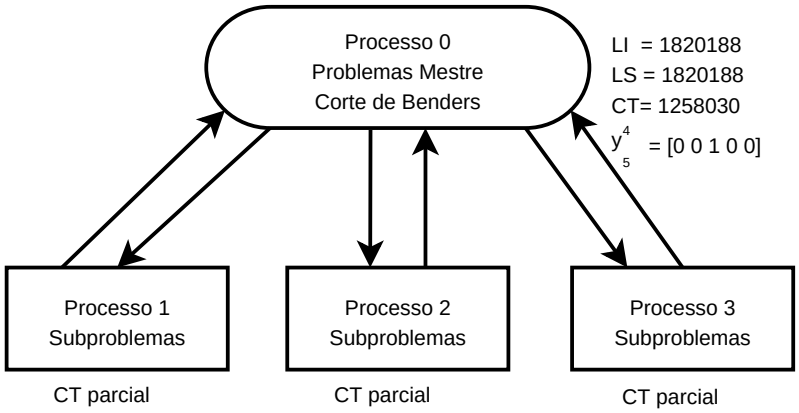
3.2.1 Exemplo de execução do algoritmo

Para exemplificar a execução do algoritmo em paralelo foi utilizado o mesmo exemplo mostrado na seção 2.3.2. A Figura 9(a) mostra a primeira iteração da execução em paralelo, temos os valores iniciais obtidos do, limite inferior, limite superior na iteração um, como também a configuração dos concentradores, que no caso é apenas um concentrador instalado. Como podemos ver cada um dos processadores resolve vários subproblemas obtendo custos parciais, que são reduzidos a um só no mestre, através de uma operação de redução.

Na última iteração 9(b), quando o algoritmo para, pois o limite inferior e superior se igualam. Dessa forma temos uma configuração final para a rede, com três nós sendo concentradores.



(a) Execução na iteração 1



(b) Execução iteração na última iteração

Figura 9: Execução em paralelo para 5 nós com 4 iterações no total

3.3 Análise e *Benchmarking*

Como foi visto na seção 2.4 o algoritmo seqüencial para decomposição de Benders tem complexidade $O(n^4)$. Agora vamos analisar a complexidade da versão em paralelo do Algoritmo 2 que foi desenvolvido no presente trabalho.

Para analisar a complexidade do Algoritmo 2, vamos dividi-lo em duas partes: a primeira parte corresponde as inicializações das variáveis para a primeira iteração do algoritmo começando na linha 1 até a linha 10; a segunda parte é iniciada no laço que implementa os ciclos de Benders na linha 11 indo até o fim do algoritmo na linha 27.

Na primeira parte do algoritmo, da mesma forma que no seqüencial, temos complexidade de $O(n)$ e $O(n^3)$ para as linhas 3 e 8, respectivamente. Na linha 10 temos o envio dos dados para os processos escravos–*broadcast*. De acordo com (QUINN, 2004) o *broadcast* para p processos requer $\lceil log\ p \rceil$ passos para passar a mensagem, a complexidade do tempo total de um *broadcast* para n itens é de $O(nlog\ p)$. No algoritmo temos o envio de u, v e CT que é da ordem de n^3, n^3 e n^2, respectivamente. Dessa forma se tem uma complexidade de $O(n^3 log\ p)$ para o envio do u e v, e $O(n^2 log\ p)$ para o envio de CT. A complexidade de comunicação para o *broadcast* nesse problema é $O((n^3+n^3+n^2)log\ p) = O(n^3 log\ p)$.

A solução em paralelo descrita na seção 3.1 mostra que estamos dividindo os subproblemas entre vários processadores. Então, no somatório mais externo da linha 4 temos que cada processo executa $\lceil n/p \rceil$ iterações. Assim sendo, a complexidade do subproblema é $O(n^4/p)$. Considerando essa análise é fácil ver que a complexidade da primeira parte do algoritmo é:

$$O(n^3) + O(n) + O(n^4/p) + O(n^3 log\ p) = O(n^4/p + n^3 log\ p)$$

A segunda parte do algoritmo é similar a primeira, sendo que nesta temos todas as instruções dentro de um laço que implementa as iterações de Benders (h) e a resolução do problema mestre na linha 13, que tem complexidade $O(n)$. Dessa forma, considerando as duas partes do algoritmo, temos que a complexidade de execução do algoritmo paralelo é:

$$O[h(n^4/p + n^3 log\ p) + (n^4/p + n^3 log\ p)] = O(n^4/p + n^3 log\ p)$$

Feita a análise de complexidade de tempo, pode-se fazer uma previsão de como será

o comportamento da execução do algoritmo paralelo. Esta previsão é útil para obter uma estimativa de como é a relação entre o número de processadores utilizados e o tempo de execução (QUINN, 2004).

Para realizar esta análise temos que considerar alguns fatores importantes quando estamos tratando de algoritmos paralelos, tais como largura de banda, latência e outros. Dessa forma pode-se derivar uma fórmula baseada nesses fatores, mapeando cada um deles em constantes, tais como:

- χ – tempo necessário para executar uma operação, ou seja, um ciclo de Benders;
- λ (latência) – tempo necessário para comunicar um valor de uma tarefa para outra por meio de um canal de comunicação;
- β (largura de banda)– representa o número de itens de dados que podem ser enviados por um canal em uma unidade de tempo;
- n – tamanho do problema.

O envio de uma mensagem contendo n itens de dados requer um tempo $\lambda + n/\beta$ (QUINN, 2004). Cada envio de mensagem gasta $\lceil log\ (p) \rceil$ passos e cada passo envolve o envio de uma mensagem. Como o tipo de dado utilizado é *double* (4 *bytes*), a quantidade de dados enviados é igual a $4n$ *bytes*. Dessa forma, a complexidade do *broadcasts* para o algotitmo em questão é $O(nlog(p))$. A partir dessas expressões, O tempo de comunicação para um algoritmo paralelo considerando os envios de mensagem é $\lceil nlog\ p \rceil (\lambda + 4n/\beta)$ (QUINN, 2004).

Sendo χ a média de tempo necessária para executar uma iteração de Benders, o tempo de computação esperado do programa paralelo é $n^3 \lceil n/p \rceil \chi$ e o tempo total de execução esperado é dado por:

$$n^3 \lceil n/p \rceil \chi + \lceil nlog\ p \rceil (\lambda + 4n/\beta)$$

A Figura 10 mostra a previsão do tempo de execução e o tempo de execução real para uma instâncias de 18 nós, onde $\chi = 10^{-5}\ sec$, $\lambda = 250\ \mu seg$ e $\beta = 10^5$. Verificamos que o tempo real de execução da implementação em paralelo aproximou-se consideravelmente da previsão realizada.

Com essa previsão podemos saber se a solução adotada foi a melhor forma de paralelização do algoritmo. Como vimos, os tempos de execução diminuíram consideravelmente com o aumento do número de processadores, então temos uma boa solução

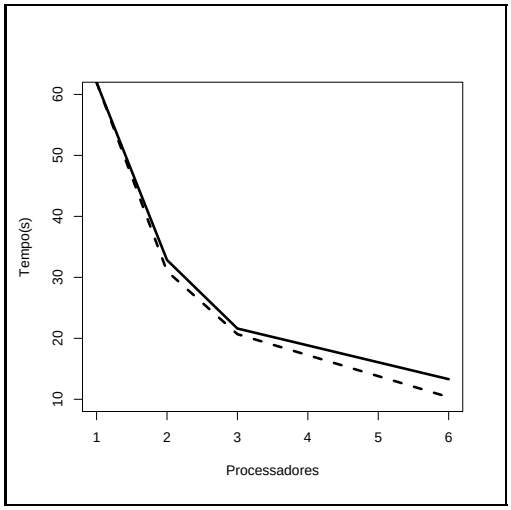


Figura 10: Comparação da previsão do tempo de execução usando 1 até 6 processadores e o tempo real obtido para o algoritmo paralelo para uma instância de 18 nós

em paralelo. Na seção seguinte serão mostradas outras medidas de performance importantes em programas paralelos que foram utilizadas para analisar o algoritmo.

3.4 Análise de performance

Aqui serão descritas duas medidas de performance para algoritmos paralelos e também uma análise dessas medidas para o algoritmo de Benders em Paralelo.

3.4.1 Speedup e Eficiência

O speedup e a eficiência são as medidas mais importantes da qualidade de um algoritmo paralelo. O *speedup* representa o quanto o algoritmo paralelo é melhor que o seqüencial e é dado pela razão entre o tempo de execução seqüencial e o tempo de execução em paralelo, ou seja:

$$Speedup = \frac{\text{Tempo de execução sequëncial}}{\text{Tempo de Execução em paralelo}}$$

Enquanto que a eficiência é a capacidade do algoritmo paralelo manter o tempo de execução com o crescimento do tamanho do problema. Essa medida da utilização do processador e é definida pela razão entre o *speedup* o número de processadores utilizados. Temos a seguinte expressão que representam a eficiência:

$$\text{Eficiência} = \frac{\text{Tempo de execução seqüencial}}{\text{Número de Processadores} \times \text{Tempo de Execução em paralelo}}$$

Essas expressões representam medidas reais que são obtidas da execução dos algoritmos paralelos, mas pode-se obter uma previsão dos valores como veremos a seguir.

3.4.2 Lei de Amdahl

É possível se obter uma previsão do speedup que será obtido uma vez que o algoritmo paralelo foi desenvolvido, isso pode ser feito através da *lei de Amdahl* que pode ser definida como segue:

Lei de Amdahl (AMDAHL, 1967). Seja f a fração de operacões em uma computação que deve ser executada seqüencialmente, onde $0 \leq f \leq 1$. O speedup máximo S obtido pela execução em paralelo com p processadores executando a computação é:

$$S \leq \frac{1}{f + (1-f)/p} \tag{3.1}$$

Então temos uma forma de expressar o speedup máximo em função do paralelismo e da fração de computação que é executada de forma seqüencial, prevendo um limite superior para o speedup com uma certa quantidade de processadores para resolver o problema em paralelo.

Para exemplificar a análise do speedup do algoritmo 2 vamos considerar um problema de 18 nós e um total de seis processadores. A fração executada seqüencialmente para a instância em estudo corresponde a 3% da execução total do programa, então podemos dizer que o algoritmo paralelo terá no máximo um *speedup* de 4.35 segundo a lei de Amdahl, ou seja:

$$\psi \leq \frac{0.03 + 0.97}{0.03 + (0.97)/6} = \frac{1}{0.03 + 0.2} = 4.35$$

A Figura 11 mostra a relação do *speedup* dado pela lei de Amdahl e o *speedup* no melhor caso que se pode ter. Podemos ver que quando o número de processadores vai aumentando o speedup real vai se distanciando do speedup previsto pela lei de Amdahl, isto porque a lei de Amdahl ignora o tempo de comunicação gasto na troca de mensagens entre os processadores.

Geralmente, quando o tamanho do problema cresce a fração f de operações inerente-

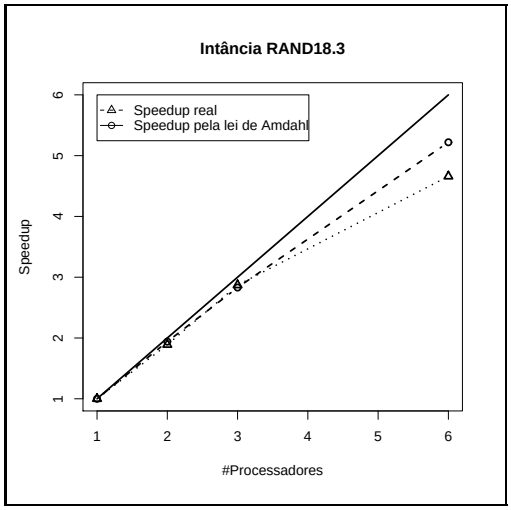


Figura 11: Previsão do speedup pela lei de Amdahl(linha pontilhada) e speedup real(linha sólida)

mente seqüenciais decresce, tornando o problema mais viável à paralelização. Este fenômeno é chamado de Efeito Amdahl. A Figura 12 ilustra o efeito Amdahl quando calculamos a previsão do speedup para o algoritmo de Benders em paralelo. Com o aumento no tamanho do problema (N) a altura da curva do speedup aumenta. Podemos concluir que para um número fixo de processadores, o aumento do speedup está em função do crescimento do tamanho do problema.

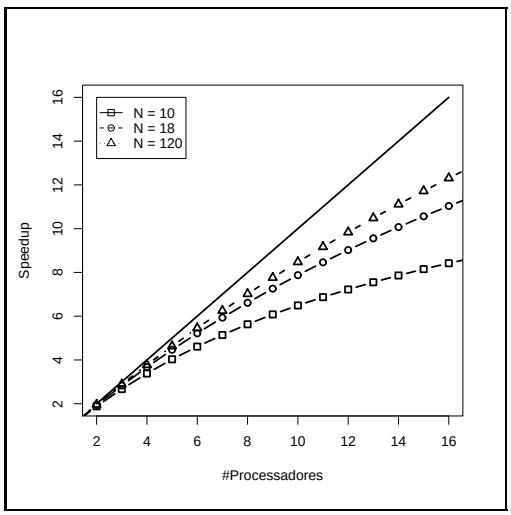


Figura 12: Efeito Amdahl

Constatamos aqui com as previsões realizadas que a solução apresentada na seção 3.1 é uma boa forma de paralelizar o algoritmo 1, já que podemos obter bons *speedups*

para o algoritmo paralelo tanto com o aumento do número de processadores, como também o aumento do tamanho do problema. A seguir apresentaremos o algoritmo paralelo desenvolvido.

3.5 Considerações finais

Neste capítulo a descrição da solução em paralelo para o algoritmo de Benders, uma análise sobre viabilidade da implementação do mesmo. Dessa análise deduziu-se que a natureza da solução é inerente a paralelização, já que tormou o problema decomponível. No capítulo que segue será apresentado os resultados obtidos da execução em paralelo e uma análise da performance do algoritmo de acordo com esses resultados, expondo os benefícios trazidos pela implementação em paralelo.

4 EXPERIMENTOS E RESULTADOS

Aqui serão apresentados os resultados obtidos da implementação em paralelo do algoritmo de decomposição de Benders apresentado na seção 4.2, mostrando uma comparação com o algoritmo seqüencial e as melhorias trazidas. Bem como um estudo do algoritmo paralelo, a fim de comprovar sua eficiência, e também verificar limitações e melhorias que podem ser realizadas.

4.1 Cenário

Os algoritmos foram desenvolvidos em linguagem C(GCC 3.3.5), utilizando-se o sistema operacional Debian 1:3.3.5-13, LAM/MPI 7.1.1 e GLPK LP/MPI 4.8.

O **MPI–Message Passing Interface** (GROUP, 2001) é uma biblioteca de passagem de mensagem, que define um conjunto de rotinas para facilitar a comunicação entre processos paralelos. A biblioteca MPI é portável para qualquer arquitetura, tem aproximadamente 125 funções e ferramentas para se analisar a performance. O método de comunicação é baseado no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. Existem várias implementações do MPI, aqui utilizamos o LAM/MPI (LAM/MPI, 2005).

O **GLPK–GNU Linear Programming Kit** (KIT, 2005) é um pacote para resolução de problemas de programação linear, programação inteira–mista e outros problemas relacionados. É um conjunto de rotinas escritas em C e organizadas na forma de biblioteca. O pacote inclui componentes, tais como: o método simplex, o método de ponto interior primal-dual, o método *Branch–and–Bound*, entre outros.

Foram utilizadas dois conjuntos de instâncias, a saber:

- Instâncias pseudo-aleatórias (RAND) geradas pelo compilador C de tamanhos 8, 12, 16 e 18.

- Um conjunto de dados chamados de AP introduzidas por Ernst e Krishnamorthy (ERNST; KRISHNAMOORTHY, 1996, 1999). Esse conjunto de dados é derivado de uma aplicação real para redes postais de entrega. Consistem de 10 até 200 nós variando de 10, que representam os códigos postais de distritos, ao longo com suas coordenadas e volumes de fluxo (*mailflow*).
- Instâncias da base de dados CAB do *US Civil Aeronautics Board*, dividas em problemas de tamanho (N) $10, 15, 20$ e 25.

Para cada uma das instâncias foram gerados problemas testes variando-se o desconto utilizado. Utilizamos os descontos de 2%, 3%, 4%, 6% e 8% para cada conjunto de instâncias. AS instâncias aleatórias utilizadas tem tamanho 8, 12, 16 e 18 e desconto de 3%. No caso das instâncias AP usamos as seguintes: as de tamanho 10, 20 e 30 variando o desconto de 2%, 4% e 6%; as de tamanho 100 variando o desconto de 2%, 4% e 6%; e a instância de tamanho 120 com desconto de 2%. As instâncias CAB utilizadas foram as com tamanho 10, 15 e 20 com descontos de 2%, 4%, 6% e 8%. As demais não foram utilizadas por limitações no número de máquinas e tempo de execução.

Para cada instância, o algoritmo foi executado dez vezes e retirado o tempo de execução de cada uma delas. Foi então retirada a média desses tempos de execução e o intervalo de confiança assintótico de 95%. Os tempos de execução têm uma variação muito pequena, com isso tivemos intervalos de confiança pequenos.

Para apresentação dos resultados as instâncias foram nomeadas como APn.α, RANDn.α e CABn.α, onde n é o número de nós na rede e α é o desconto que foi utilizado nos testes(2 para 20%,4 para 40%,...).

Os experimentos computacionais foram realizados no *cluster* Candiru*, com a seguinte configuração:

- 4 máquinas AMD Athlon(tm) 64 Processor 3200+ 2200MHz com 1GB de memória;
- 1 máquina AMD Opteron(tm) 246 1990MHz com dois processadores;
- Uma rede *fast ethernet* 10/100Mbps.

Na seção seguinte temos os resultados obtidos mostrados em tabelas e alguns gráficos

*O *cluster* candiru encontra-se instalando no Laboratório de Métodos Númericos e Simulação computacional do Instituto de Computação na Universidade Federal de Alagoas

para algumas instâncias que selecionamos para mostrar o comportamento do algoritmo.

4.2 Resultados

Os resultados serão apresentados em forma de gráficos e tabelas como já foi mencionado anteriormente. Para todas as instâncias utilizadas, foi definida uma tabela com as seguintes colunas:

- **Instância**: o tamanho do problema;
- **H**: número de iterações de Benders para se encontrar a solução ótima;
- **#P**: número de processadores;
- **Tempo de execução(s)**: tempo de execução do algoritmo em segundos;
- **% subproblemas**: a quantidade de tempo na execução, em porcentagem, dedicada a resolução dos subproblemas;
- *Speedup*: Speedup, que foi definido no capítulo 3, tendo o valor 1 quando a execução é seqüencial, ou seja, #P é um.

Instâncias	**H**	**#P**	**Tempo(s)**	**%subproblemas**	*Speedup*
RAND8.3	36	1	0,14	93	1,00
		2	0,15	47,3	0,93
		4	0,13	29,7	1,08
RAND12.3	34	1	0,59	95	1,00
		2	0,43	66,5	1,37
		3	0,36	54	1,64
		4	0,33	45,2	1,79
RAND16.3	44	1	148,95	95,7	1,00
		2	71,96	89,3	1,75
		4	46,00	77,9	3,24
RAND18.3	52	1	61,93	96,9	1,00
		2	32,84	89,9	1,89
		3	21,60	83,8	2,87
		6	13,30	68,5	4,66

Tabela 4: Resultados obtidos para o *speedup* e o tempo de execução para as instâncias geradas aleatoriamente.

As Tabela 4, 6 e 5 mostram os resultados da execução do algoritmo seqüencial, quando #P é igual a um, e paralelo, quando #P é igual a 2, 3 e 4.

Instâncias	H	#P	Tempo(s)	%subproblemas	*Speedup*
CAB10.2	36	1	0,19	94,2	1,00
		2	0,15	58,7	1,26
CAB10.4	34	1	0,17	94,1	1,00
		2	0,14	57,3	1,21
CAB10.6	44	1	0,28	93,9	1,00
		2	0,21	61,7	1,33
CAB10.8	52	1	0,38	93,5	1,00
		2	0,29	64,2	1,31
CAB15.2	152	1	15,73	95,8	1,00
		3	6,41	78,9	2,45
CAB15.4	234	1	37,16	95,7	1,00
		3	14,36	81,8	2,58
CAB15.6	289	1	56,44	95,7	1,00
		3	21,92	82,7	2,57
CAB15.8	334	1	74,50	95.6	1,00
		3	28,92	83,2	2,57
CAB20.2	107	1	23,51	97	1,00
		2	12,72	88,7	1,84
CAB20.4	178	1	64,31	97	1,00
		2	34,41	90,8	1,86
CAB20.6	310	1	193,89	97	1,00
		2	102,23	92,1	1,89
CAB20.8	461	1	426.38	97	1,00
		2	223,45	92,5	1,90

Tabela 5: Resultados para as instâncias CAB.

Como se pode perceber, para todas as instâncias a fração de tempo total tomada pela resolução de subproblemas é de uma ordem de grandeza muito maior que o tempo gasto na solução do problema mestre. Percebendo essa característica, foi tomada a decisão de se resolver os subproblemas em paralelo.

Analisando a instância AP100.2 da Tabela 6, temos um tempo de execução seqüencial, com um processador, de aproximadamente 108 segundos e a porcentagem para a resolução dos subproblemas foi de 99%; na execução em paralelo obtivemos um tempo de 62 segundos utilizando-se 2 processadores e 37 segundos com quatro processadores. Considerando esses resultados percebemos uma redução significativa no tempo de execução e também no tempo gasto na resolução dos subproblemas. Esse resultado era esperado, pois a maior parte do tempo de execução é gasto na resolução dos subproblemas, justamente o que está sendo executado em paralelo. Esse comportamento do tempo é semelhante para todas as instâncias testadas.

No caso de problemas menores, tais como, AP10.2 6 e CAB10.4 5 que têm um número de iterações relativamente pequenos, o uso de várias máquinas não faz muita diferença no tempo de execução, pois o problema é resolvido rapidamente, e o tempo

Instâncias	H	#P	Tempo(s)	%subproblemas	*Speedup*
AP10.2	12	1	0,02	94	1,00
		2	0,02	34	1,00
AP10.4	30	1	0,13	93,7	1,00
		2	0,11	53,9	1,18
AP10.6	162	1	3,64	92,7	1,00
		2	2,26	76,6	1,61
AP20.2	9	1	0,14	96,6	1,00
		2	0,13	52,2	1,07
AP20.4	13	1	0,31	96,7	1,00
		2	0,25	61,1	1,24
AP20.6	45	1	3,95	96,9	1,00
		2	2,35	81,7	1,68
AP30.2	8	1	0,57	95,7	1,00
		2	0,46	59,2	1,23
AP30.4	10	1	0,91	95,9	1,00
		2	0,68	64,2	1,33
AP30.6	24	1	5,50	96	1,00
		2	3,37	78,4	1,63
AP100.2	10	1	107,40	99,1	1,00
		2	61,39	87,20	1,75
		4	37,82	70,10	2,84
AP100.4	18	1	363,69	99,1	1,00
		2	196,79	92,10	1,85
		4	111,6	80,60	3,26
AP100.6	46	1	2450,32	99,2	1,00
		2	1272,79	96,00	1,93
		4	756,21	85,00	3,23
AP120.2	764	1	1502378,62	99,4	1,00
		2	758361,79	98,70	1,98
		3	509695,09	98,00	2,95
		4	384185,20	97,30	3,91

Tabela 6: Resultados para as instâncias AP.

de comunicação entre os processadores é alto em relação ao tempo total da execução. Mas para instâncias maiores, conseqüentemente, mais difíceis de serem resolvidas, temos um desempenho muito bom do algoritmo em paralelo.

Lembrando que o *speedup* é obtido dividindo-se o tempo de execução seqüencial pelo tempo de execução em paralelo, obtivemos na maioria dos casos valores razoáveis, já que conseguimos diminuir muito o tempo de execução. Podemos ver na Tabela 4 o valor do *speedup* para a instância RAND18.3 foi de 4,66 quando o algoritmo foi executado em seis processadores, isto significa que o algoritmo paralelo é quatro vezes mais rápido que o seqüencial, um resultado muito significativo em relação ao tempo de processamento. Esse foi o nosso melhor resultado para o *speedup*. Em alguns casos, tais como, a instância RAND8.3 4 o *speedup* foi baixo, isso justifica-se pelo tamanho do problema, pois em todos os problemas resolvidos em paralelo, existe um momento

em que o algoritmo paralelo é melhor que o seqüencial em relação ao tamanho do problema.

Esse resultados também são mostrados através de gráficos. Foram escolhidas as instâncias RAND16.3, RAND18.3, AP100.2 e AP120.2 e foram e gerou-se dois tipos de gráficos: um relacionando o número de processadores e o tempo de execução do algoritmo, mostrados nas figuras 13, 14, 15 e 16. E outro relacionando o número de processadores e o *speedup*, mostrados nas figuras 17, 18, 19 e 20.

Em relação ao tempo, como já foi comentado antes, em todos os casos o tempo foi reduzido consideravelmente, tomemos como exemplo a Figura 16, podemos ver que quando o algoritmo em paralelo é utilizado com dois processadores, temos aproximadamente o tempo de execução reduzido a metade, e esse resultado se mantém com o aumento do número de processadores. Como conseqüência disso, temos também excelentes resultados para o *speedup*, como podemos ver na Figura 20.

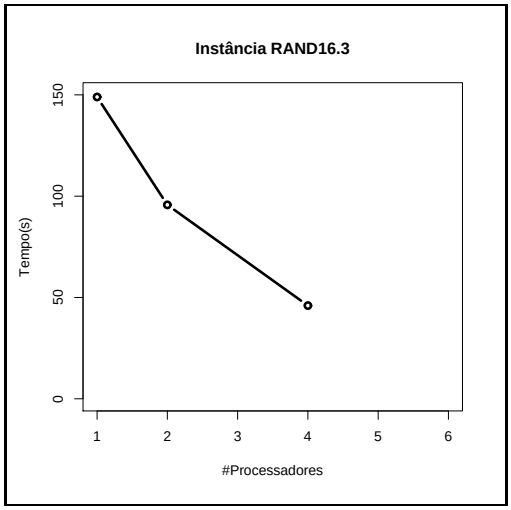


Figura 13: Tempo de execução para instância RAND16.3 com #processadores igual a 1, 2 e 4

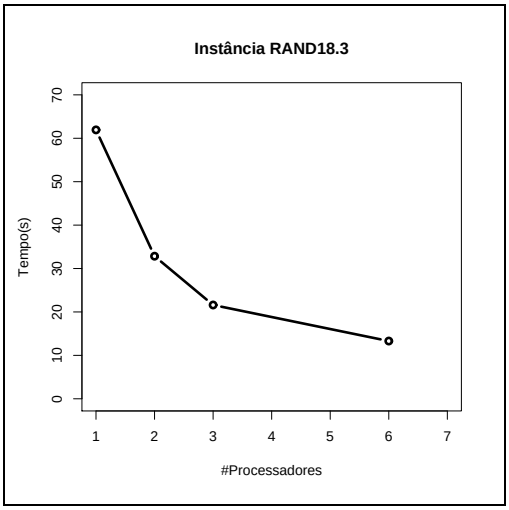


Figura 14: Tempo de execução para instância RAND16.3 com #processadores igual a 1, 2,3 e 6

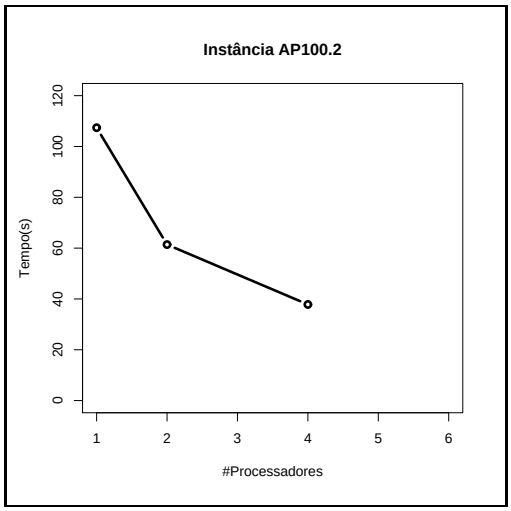


Figura 15: Tempo de execução para instância AP120.2 com #processadores igual a 1, 2 e 4

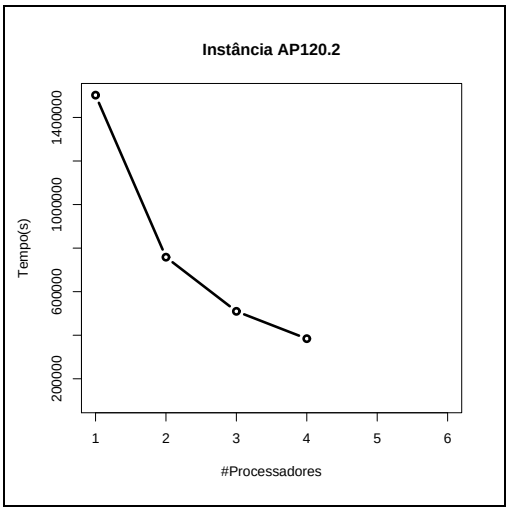


Figura 16: Tempo de execução para instância AP120.2 com #processadores igual a 1, 2, 3 e 6

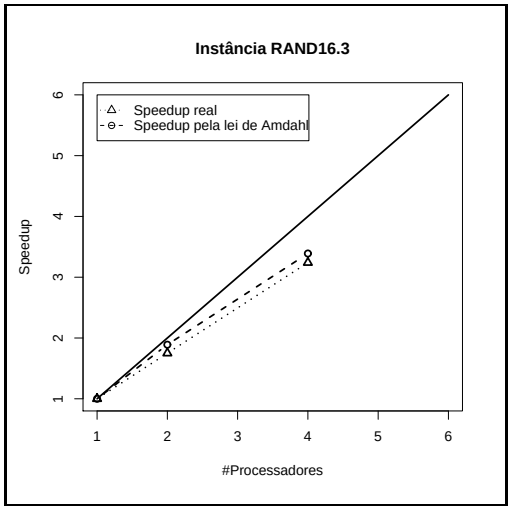


Figura 17: Speedup para instância RAND16.3 com #processadores igual a 1, 2 e 4

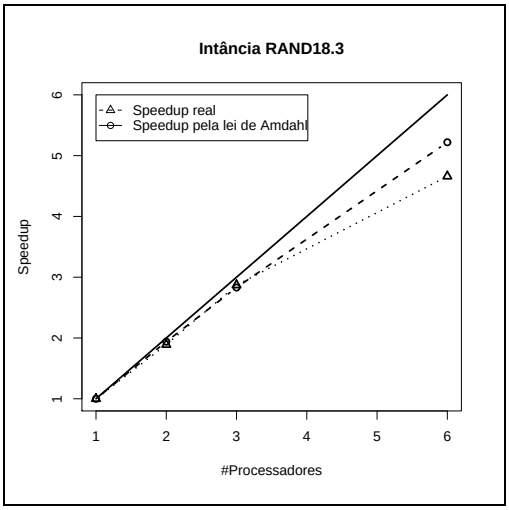


Figura 18: Speedup para instância RAND18.3 com #processadores igual a 1, 2, 3 e 4

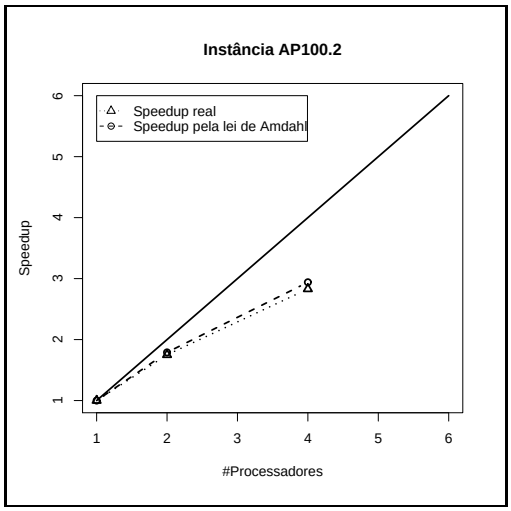


Figura 19: Speedup para instância AP100.2 com #processadores igual a 1, 2 e 4

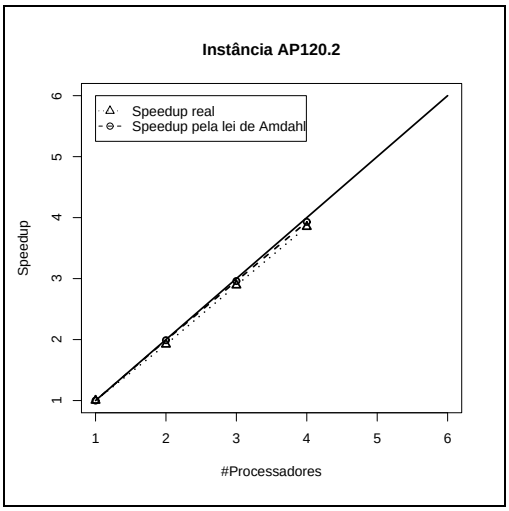


Figura 20: Speedup para instância AP120.2 com #processadores igual a 1, 2, 3 e 4

Nas seções seguintes é feita uma análise analítica da performance do algoritmo.

4.3 Análise de Performance

Uma vez que os algoritmos paralelos estão desenvolvidos, como se comprovar que o algoritmo é eficiente? Ou ainda, de que forma pode-se melhorar o algoritmo?

A análise de performance de algoritmos paralelos pode ser feita utilizando algumas métricas. Aqui foram utilizadas a métrica de Karp–Flatt e a métrica de isoeficiência. Descritas a seguir.

1. **Métrica de Karp–Flatt**

   Karp e Flatt propuseram uma métrica (KARP; FLATT, 1990) que calcula a fração seqüencial do código. Essa métrica é útil por duas razões. Primeiro, porque ela considera a quantidade de *overhead* da parte paralela do programa. Segundo, com ela é possível avaliar quais o obstáculos que temos para se obter bons *speedups* (QUINN, 2004).

   A **Métrica de Karp-Flatt** pode ser definida como segue:

   Dada uma computação paralela exibindo um speedup ψ *em* p *processadores, onde* $p > 1$*, fração seqüencial determinada experimentalmente* ***e*** *é definida como:*

$$e = \frac{1/\psi - 1/p}{1 - 1/p} \tag{4.1}$$

   Para calcular o valor de ***e*** , usaremos a instância AP120.2, os valores de ψ, p e e são substituídos na equação 4.1 e os valores encontrados são mostrados na Tabela 7.

p	ψ	e
2	1.98	0.010
3	2.95	0.009
4	3.91	0.008

Tabela 7: Valores de **e** para a instância AP120.2

   Os valores da Tabela 7 mostram que o valor de e está diminuindo com o número de processadores, dessa forma teremos sempre um *speedup* bom, resultado esperado, já que a parte seqüencial do problema aqui tratado é pequena em relação a parte que foi paralelizada. Caso o valor de e fosse constante, poderíamos concluir que o problema seria a limitação do paralelismo, que não é o nosso

caso. Ou ainda, se o valor de e estivesse crescendo, poderíamos concluir que o problema seria o *overhead* de comunicação.

2. **Métrica de isoeficiência**

   A medida de isoeficiência está relacionada a escalabilidade do programa. Indica qual é a taxa de crescimento do tamanho do problema em relação ao número de processadores para que a eficiência seja mantida. Essa taxa determina a escalabilidade do problema.

   A eficiência, como já foi definida no capítulo 3, é a relação entre o *speedup* e o número de processadores. A Figura 21 mostra a relação de eficiência do algoritmo com o crescimento do número de processadores com um tamanho fixo para o problema, no caso foram utilizadas duas instâncias, RAND18.2 e AP100.2. Podemos ver uma situação comum, onde a eficiência diminui com o aumento no número de processadores. Já a Figura 22 mostra a relação de eficiência com a variação do tamanho do problema e um número fixo de processadores, no caso $p = 2$. Nesse caso a eficiência tende a aumentar com o tamanho do problema.

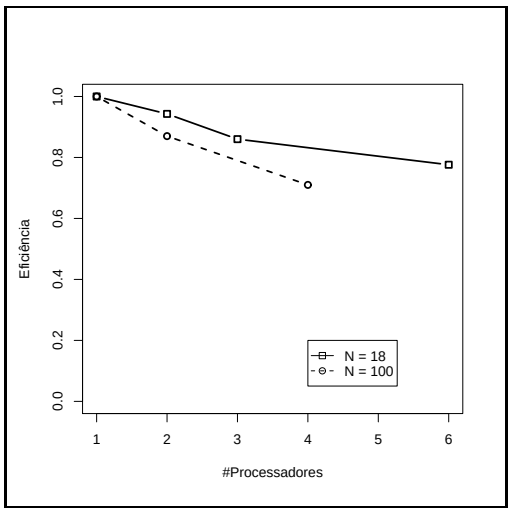


Figura 21: Eficiência do algoritmo para as instâncias RAND18.2 e AP100.2 com o aumento do #p

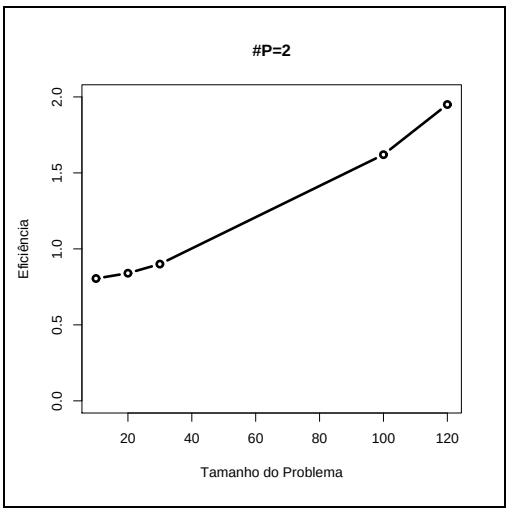


Figura 22: Eficiência com o aumento do tamanho do problema para p=2

Podemos utilizar a métrica de isoeficiência para determinar se o programa é escalável, se é possível manter um determinado nível de eficiência. A relação de isoeficiência (QUINN, 2004) é definida como segue:

Suponha um programa que está sendo executado em um computador paralelo tenha uma eficiência $\varepsilon(n,p)$. Defina $C = \varepsilon(n,p)/(1-\varepsilon(n,p))$ e $T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$. Para manter o mesmo nível de eficiência com o aumento do número de processadores, n deve ser incrementado tal que satisfaça a seguinte equação:

$$T(n,1) \geq CT_0(n,p)$$

No caso do algoritmo de Benders em paralelo o algoritmos seqüencial tem complexidade de tempo $O(n^4)$. Cada um dos processadores executando em paralelo gasta $O(nlogp)$ para comunicação. Então a relação de isoeficiência é dada por:

$$n^4 \geq C(pnlog\ p) \Longrightarrow n \geq (Cplog\ p)^{1/3}$$

Considerando a memória utilizada $M(n) = n^4$, temos:

$$M((Cplog\ p)^{1/3})/p = C\ logp\sqrt{Cplog\ p}$$

Com isso podemos verificar que o problema tem escalabilidade pobre, pois é necessário uma quantidade muito grande de memória cada vez que aumentamos o tamanho do problema.

4.4 Considerações Finais

Neste capítulo foram apresentados os resultados obtidos com a implementação em paralelo do algoritmo de decomposição de Benders. Os resultados foram bons, mostrando que podemos ter um tempo de execução menor para a execução do que no caso seqüencial. E utilizando-se algumas métricas pôde-se verificar que temos um bom algoritmo, mas que ainda é necessário melhorar de alguma forma sua eficiência.

5 CONCLUSÃO

Este trabalho abordou o problema de localização de concentradores de consolidação de cargas e alocação de ponto de demandas a esses concentradores. Muitas empresas de transporte de carga parcelada estruturam seus sistemas de coleta, transferência e entregas usando redes do tipo *hub-and-spoke*. Foi apresentada a formulação matemática para esse problema formulada por Skorin-Kapov e a decomposição de Benders aplicada a esse modelo.

A implementação em paralelo do algoritmo de Benders foi desenvolvida a partir de um algoritmo seqüencial usando-se alocação dinâmica de memória. Foi encontrada uma grande dificuldade com esse tipo de alocação, pois o MPI lê apenas áreas contíguas de memória, o que nem sempre acontece quando se faz alocação dinâmica. Assim a alocação dinâmica teve que ser implementada de forma que ocupasse áreas contíguas na memória.

Importantes estudo são feitos para análise do algoritmo paralelo, mesmo antes da sua implementação. Fizemos esse estudo utilizando algumas leis e métricas mais conhecidas chegando a resultados satisfatórios que provou que o problema era paralelizável. Pode-se confirmar a veracidade dessas análises com testes realizados utilizando a implementação em paralelo que diminuiu o tempo de execução mesmo para problemas pequenos.

Como trabalhos futuros, pretendo utilizar um *cluster* maior com um solver mais robusto para testar problemas de tamanhos maiores e utilizar o conjunto de dados CAB(Civil Aeronautics, instâncias do conjunto de dados AP que consiste de 200 nós que representam distritos dos códigos postais, divididas em problemas de 10 a 200 nós.

Devido a grande gama de problemas que podem ser resolvidos utilizando o método de Benders e como foi comprovado que ele torna o problema paralelizável, podemos construir algoritmos para outras aplicações, tais como, problemas de distribuição multiproduto.

REFERÊNCIAS

ABDINNOUR-HELM, S.; VENKATARAMANAN, M. A. Solution approaches to the hub location problems. In: *Operations Research*. [S.l.: s.n.], 1998. v. 78, p. 31–50.

AMDAHL, G. M. Validity of the singleprocessor approach to achieving large scale compoting capabilities. *AIPS Conference Proceedings*, 1967.

ANDERSON, T.; CULLER, D.; PATTERSON, D. A case for now (network of workstations). *IEEE Micro*, v. 15, n. 1, p. 54–64, 1955.

AYKIN, T. Lagragian relaxation based approaches to capacitated hub-and-spoke network de-sign problem. *European Journal of Operational Research*, v. 79, p. 501–523, 1994.

BENDERS, J. F. Partitioning procedures for solving mixed integer variables programming problems. *Numerische Methematik*, v. 4, p. 238–252, 1962.

CAMPBELL, J. F. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, v. 72, p. 387–405, 1994.

CAMPBELL, J. F. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, v. 72, p. 387–405, 1994.

CAMPBELL, J. F. A survey of network hub location. *Studies in Location al Analysis*, v. 6, p. 31–49, 1994.

CAMPBELL, J. F. Hub location and the p-hub median problem. *Operations Research*, v. 44, n. 6, p. 923–935, 1996.

CAMPBELL, J. F.; ERNST, A. T.; KRISHNAMOORTHY, M. Hub location problems. In: ____. 1. ed. [S.l.: s.n.], 2002. cap. 12, p. 373–402.

DONALDSON, H. et al. *Schedule-Driven Cross-Docking Networks*. [S.l.], 1999. Disponível em http://www.isye.gatech.edu/apps/research-papers/papers/misc9904.pdf.

DREZNER, T.; DREZNER, Z. A note on applying the gravity rule to the airline hub problem. *Journal of Regional Science*, Blackwell Publishing, v. 41, n. 1, p. 67–72, Fevereiro 2001.

ERNST, A. T.; KRISHNAMOORTHY, M. Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science*, v. 4, n. 3, p. 139–154, Outubro 1996. Doi:10.1016/S0966-8349(96)00011-3.

ERNST, A. T.; KRISHNAMOORTHY, M. Solution algorithms for the capacitated single allocation hub location hub location problem. *Annals of Operations Research*, v. 86, n. 0, p. 141–159, Janeiro 1999.

GAVISH, B. Optimization models for configuring distributed computer systems. *IEEE Transactions on Computers*, IEEE Computer Society, Washington, DC, USA, v. 36, n. 7, p. 773–793, 1987. ISSN 0018-9340.

GEOFFRION, A. M. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, v. 10, n. 4, p. 237–260, 1972.

GEOFFRION, A. M.; GRAVES, G. W. Multicomodity distribution system design by benders decomposition. *Management Science*, v. 20, p. 822–844, 1974.

GOLDBARG, M. C.; LUNA, H. P. L. *Otimização Combinatória e Programação Linear*. 1st edition. ed. [S.l.: s.n.], 2000. ISBN 85-352-0541-1.

GONDZION, J.; SARKISSIAN, R.; VIAL, J.-P. *Parallel Implementation of a Central Decomposition Method for Solving Large Scale Planning Problems*. [S.l.], 1998. Available at http://ideas.repec.org/p/fth/ehecge/98.1.html.

GROUP, P. T. *Introduction to MPI. [Online] Disponível:http://webct.ncsa.uiuc.edu:8900/public/MPI/*. 2001.

HAMACHER, H. W. et al. *Polyhedral properties of the uncapacitated multiple allocation hub location problem*. [S.l.], Junho 2000.

KARP, A. H.; FLATT, H. P. Measuring parallel processor performance. *Commun. ACM*, ACM Press, New York, NY, USA, v. 33, n. 5, p. 539–543, 1990. ISSN 0001-0782.

KIT, G.-G. L. P. *[Online] Disponível:http://www.gnu.org/software/glpk/glpk.html*. Accesso: Novembro 2005.

KLINCEWICZ, J. G. Heuristics for the p-hub location problem. *European Journal of Operational Research*, v. 53, p. 25–37, 1991.

KLINCEWICZ, J. G. A dual algorithm for the uncapacitated hub location problem. *Location Science*, v. 4, n. 3, p. 173–184, 1996.

KLINCEWICZ, J. G. Hub location in backbone/tributary network design: a review. *Location Science*, v. 6, n. 1–4, p. 307–335, Dezembro 1998. ISSN 0018-9340.

LAM/MPI. *LAM/MPI Parallel Computing. [Online] Disponível:http://www.lam-mpi.org/*. Accesso: Novembro 2005.

LUNA, H. P. L. *Les Techniques de Décomposion-Coordination dans les Modèles Économiques d'Optimisation*. Tese (Doutorado) — Universite de Toulouse III (Paul Sabatier), 1978.

MAGNANTI, T. L.; WONG, R. T. Network design and transportation planning: Models and algorithms. *Transportation Science*, v. 18, p. 1–55, 1984.

MAYER, G.; WAGNER, B. Hublocator: an exact solution method for the multiple allocation hub location problem. *Computer Operations Research*, v. 29, p. 715–739, 2002.

MIRANDA, G. J. *Localização de Servidores e Projeto de Redes com Custos de Interdependência e Congestionamento*. Tese (Doutorado) — Universidade Federal de Minas Gerais, 2004.

MIRHASSANI, S. A. et al. Computational solution of capacity planning models under uncertainty. *Parallel Computing*, v. 26, n. 5, 2000.

NAGY, G.; SALHI, S. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, v. 162, n. 1, p. 126–141, Abril 2005.

NIELSEN, S. S.; ZENIOS, S. A. Scalable parallel benders decomposition for stochastic linear programming. *Parallel Computing*, v. 23, n. 8, p. 1069–1088, 1997.

O'KELLY, M. E. The location of interacting hub facilities. *Transportation Science*, v. 20, n. 2, p. 92–106, 1986.

O'KELLY, M. E. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, v. 32, p. 393–404, 1987.

O'KELLY, M. E.; MILLER, H. J. The hub network design problem. *Transportation Geography*, v. 2, n. 1, p. 31–40, 1994.

O'KELLY, M. E.; SKORIN-KAPOV, D.; SKORIN-KAPOV, J. Lower bounds for the hub location problem. *Management Science*, v. 41, p. 283–302, 1995.

PIRKUL, H.; SCHILLING, D. A. An efficient procedure for designing single allocation hub and spoke systems. *Management Science*, v. 44, n. 12, p. 235–242, 1998.

QUINN, M. J. *Parallel Computing: Theory and Pratice*. 2st edition. ed. [S.l.: s.n.], 1994. ISBN 0-07-051294-9.

QUINN, M. J. *Parallel Programming in C with MPI and OpenMP*. 1st edition. ed. [S.l.: s.n.], 2004. ISBN 0072822562.

SASAKI, M.; SUZUKI, A.; DREZNER, Z. On the selection of hub airports for an airline hub-and-spoke system. *Computers & Operations Research*, v. 26, p. 1411–1422, 1999.

SKORIN-KAPOV, D. Hub network games. *Networks*, v. 31, n. 4, p. 293–302, Dezembro 1998.

SKORIN-KAPOV, D.; SKORIN-KAPOV, J. On tabu search for the location of interacting hub facilities. *European Journal of Operational Research*, v. 73, p. 501–508, 1994.

SKORIN-KAPOV, D.; SKORIN-KAPOV, J.; O'KELLY, M. Tight linear programming relaxations of uncapacitated p-hub median problems. *European Journal of Operational Research*, v. 94, p. 584–593, 1996.

SKORIN-KAPOV, D.; SKORIN-KAPOV, J.; O'KELLY, M. Tight linear programming relaxations of uncapacitated p-hub median problems. *European Journal of Operational Research*, v. 94, p. 584–593, 1996.

TAYLOR, G. D. et al. Hub and spoke networks in truckload trucking: configuration, testing and operacional concerns. *Logistics & Transportation Review*, v. 31, n. 3, p. 209–238, 1995.