ADONIAS AMORIM DE LIMA JUNIOR

ANÁLISE DE COMPLEXIDADE ASSINTÓTICA DE UM ALGORITMO EXATO PARA O CÁLCULO DO *NUCLEOLUS* DE UM JOGO COOPERATIVO

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora
Profª. Renata Couto Moreira

LAVRAS
MINAS GERAIS - BRASIL
2002

ADONIAS AMORIM DE LIMA JUNIOR

ANÁLISE DE COMPLEXIDADE ASSINTÓTICA DE UM ALGORITMO EXATO PARA O CÁLCULO DO *NUCLEOLUS* DE UM JOGO COOPERATIVO

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 16 de Dezembro de 2002

_____

Prof. José Monserrat Neto

_____

Prof. Ricardo Martins de Abreu

_____

Profª. Renata Couto Moreira

UFLA
(Orientadora)

LAVRAS
MINAS GERAIS – BRASIL

“O senhor é meu escudo, a minha glória, é o que me exalta e me fortalece.” Dedico a conquista a todos que me apoiaram para este objetivo.

Agradecimentos

Agradeço a Deus por ter me proporcionado a felicidade de ser filho do casal, Adonias e Maria Áurea, que formam a base e o apoio de toda minha estrutura. Esta que pôde ser desenvolvida graças a presença dos amigos e dos meus irmãos, que sempre me apoiaram na realização desta conquista. À minha orientadora Renata, que me incentivou e me ajudou no desenvolvimento deste trabalho. À minha irmã Neiry que sempre me ouviu nos momentos difíceis. À Patrícia e à Vaninha pela amizade e paciência e ao Alexandre, que juntos formamos uma equipe de grande aproveitamento nos estudos. Ao Décius pela amizade e companheirismo nos trabalhos. Ao Ricardo por formarmos uma equipe de estudo e discussão sobre este tema. À Giselle pela amizade. Enfim, o meu muito obrigado a todos que contribuíram direta ou indiretamente na conquista deste objetivo.

SUMÁRIO

LISTA DE FIGURAS

LISTAS DE TABELAS

Resumo

O problema atacado pelo projeto está em alocar os custos de uma rede de maneira justa entre os participantes e analisar a complexidade da solução para tal problema. A aplicação se encontra em situações onde determinadas entidades visam alcançar um objetivo comum. Visto que cada entidade possui um custo para obter o mesmo serviço, uma cooperação pode ser formada para a redução do custo. Para se obter uma divisão deste custo de maneira ótima entre os usuários, o valor de *Shapley* e o *Nucleolus* podem ser utilizados como solução. A solução do *nucleolus*, que é uma solução pertencente à Teoria dos Jogos Cooperativos, foi utilizada, pois este aloca os custos entre os usuários de maneira mais justa. Um algoritmo baseado na programação linear para a obtenção do *nucleolus*, proposto por [Grano81], foi desenvolvido, no entanto foi verificado que tal algoritmo necessitava de modificações para que a solução pudesse ser obtida, logo este foi estendido. Ao verificar os resultados, pode-se verificar que o algoritmo proposto resultava em resultados mais justos que o algoritmo encontrado na literatura, onde a complexidade se manteve a mesma.

Abstract

The problem attacked by the project is to allocate the costs of a net in a fair way among the participants and to analyze the complexity of the solution for the problem. The application is in situations where certain entities seek to reach a common objective. Because each entity possesses a cost to obtain the same service, a cooperation can be formed for the reduction of the cost. To obtain a division of this cost in a great way among the users, the value of Shapley and *Nucleolus* can be used as solution. The solution of the nucleolus, that is a solution belonging to the Cooperative Game Theory, it was used, therefore this allocates the costs among the users in a fairer way. An algorithm based on the lineal programming for the obtaining of the nucleolus, proposed for [Grano81], it was developed, however it was verified that such algorithm needed modifications so that the solution could be obtained, so this was extended. When verifying the results, it can be verified that the proposed algorithm resulted in fairer results than the algorithm found in the literature, where the complexity stayed the same.

Capítulo 1

Introdução

O trabalho tem como objetivo a realização de uma análise de complexidade assintótica do algoritmo exato para o cálculo do *nucleolus* de um jogo cooperativo, proposto por [Grano81], assim como seu desenvolvimento para que resultados sejam obtidos e avaliados.

O interesse por este tipo de problema, aplicado a redes de telecomunicações, aumentou no Brasil, isso devido à privatização de várias empresas de fornecimento de serviços, pois as empresas privadas necessitam de uma divisão de custos e de lucros de forma justa. Na situação de monopólio, as empresas eram subdivididas por todo o país, as vantagens que se obtinha em uma região, devido ao grande número de usuários, poderiam ser repassadas para outra região, onde este número não era tão grande, evitando assim a falência desta empresa nesta região específica. Verifica-se que a divisão de custos não se fazia necessário, visto que a fonte financiadora era a mesma, o governo. Com a privatização cada empresa deve arcar com seus gastos, gastos estes que podem estar relacionado à utilização de recursos públicos e como várias empresas podem estar utilizando este recurso, a divisão dos custos deve ser realizada de forma que nenhuma destas empresas se sinta prejudicada.

Para que o custo de um determinado serviço se torne ótimo, pode-se utilizar os vários algoritmos existentes para o cálculo do custo mínimo total de

uma rede, assim como mostra [Corme90]. No entanto estes algoritmos não são o suficiente para obter-se um custo justo para cada usuário da rede, ou seja, dividir o custo da rede entre os usuários não é uma tarefa fácil.

Com o intuito de exemplificar uma alocação mais justa de custos, um exemplo de transporte pode ser citado, onde os custos estão relacionados ao gasto com combustível:

Suponha que três empregados de uma empresa consideram a possibilidade deste tipo de otimização para reduzir o custo diário de suas viagens. O custo de conduzir um carro de um empregado para outro ou de um empregado para a empresa é determinado na Figura 1.

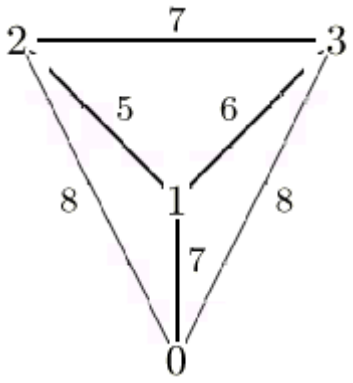

Figura 1: Custos de condução

Nesta figura os empregados são denotados pelos nós 1, 2 e 3 e a empresa pelo nó 0 do grafo. O custo mínimo total da rede é 18 e é formado pelos arcos {(0,1), (1,2), (1,3)}. Esta árvore corresponde ao plano de otimização das rotas de transporte na qual os empregados 2 e 3 conduzem o carro até o empregado 1, onde todos os empregados levam o carro à empresa. Supondo que estes custos sejam pagos pelos empregados, qual seria a maneira mais justa de dividir este custo total entre os empregados?

Suponha as duas soluções:

Primeira solução:

Empregado1= 7; Empregado2 = 5; Empregado3 = 6

Segunda solução:

Empregado1= 5; Empregado2 = 6; Empregado3 = 7

Veja que a primeira solução não forneceu nenhum desconto ao empregado 1, que tinha custo de sete para se conduzir à empresa e continuou com este custo. Este alocação não é justa, visto que este empregado forma a "ponte" de ligação dos empregados 2 e 3 à empresa.

A segunda solução fornece uma alocação de custos mais justa, visto que o empregado 1 obteve desconto de dois, aumentando para isso uma unidade para os demais empregados, que permanecem com desconto em relação ao custo de 8 que teriam se conduzissem o carro diretamente para a empresa.

Para que uma solução de alocação de custos de uma forma mais justa seja obtida, a solução de *nucleolus* é utilizada neste trabalho.

Ao realizar-se o estudo do algoritmo de obtenção do *nucleolus* de forma exata, verificou-se que este faltava informações para que a solução pudesse ser obtida. Para que os objetivos fossem alcançados, propostas de modificações foram adicionadas ao algoritmo, na qual foi possível obter uma solução mais justa para os participantes da cooperação, mantendo a mesma complexidade do algoritmo original. Estas melhoras foram comprovadas através de análises de resultados obtidos.

O trabalho está organizado de modo a apresentar um bom referencial teórico, para que assim se possa compreender o algoritmo proposto para a obtenção do nucleolus, assim como seus resultados e análises obtidos. Deste modo temos:

O capítulo 2 tem como objetivo definir formalmente uma rede de acesso e mostra como obter o custo mínimo total desta rede, visto que este será o custo alocado entre os usuários do serviço.

O capítulo 3 possui as definições da teoria dos jogos cooperativos, onde se enfatiza o *nucleolus*, que é a solução mais justa dentre todas as soluções possíveis para o jogo.

O capítulo 4 possui a definição de programação linear e também do método Simplex, que é o procedimento geral para resolver tais problemas de programação linear, que será utilizado pelo algoritmo na obtenção do *nucleolus* do jogo cooperativo.

No capítulo 5 foram propostas modificações no algoritmo exato encontrado na literatura, assim como a análise de sua complexidade teórica, na qual pode se observar a mesma complexidade em relação ao algoritmo original .

No capítulo 6 são mostrados os resultados obtidos pelo algoritmo proposto, pôde-se então obter a complexidade prática deste algoritmo em sua execução. Os resultados puderam ser analisados no decorrer deste capítulo, comprovando deste modo a melhora deste algoritmo.

Capítulo 2

Rede de Acesso

2.1 - Problemas de Otimização em Redes

Um exemplo muito freqüente de cooperação está relacionado ao acesso a algum tipo de serviço e a melhor forma de se obter este serviço é montar uma estrutura de menor custo total que viabilize este acesso, esta estrutura é então chamada de rede de acesso.

Uma rede de acesso é uma infra-estrutura comum que interliga os vários usuários de algum tipo de serviço (serviços de comunicação, de energia elétrica, de água, de esgoto entre outros) às centrais de fornecimento.

"Seja $G=(N,E)$ uma rede de acesso representado por um grafo não direcionado com um conjunto N de nós, representando os pontos a serem conectados, e um conjunto E de arcos, representando as possibilidades de ligações entre estes pontos. Deste modo, para cada nó $i \in N$ pode ser associada uma demanda d_i, que no caso mais geral pode ser maior que zero (nó preto ou nó cliente), menor que zero (nó fornecedor) ou igual a zero (nó bifurcação, transbordo ou nó branco). A presença deste tipo de nó surge nas situações em que a decisão de menor custo pode seguir inicialmente por uma trajetória comum e só a partir de determinado ponto bifurcar o caminho para ligar os

clientes. Desta forma, cada arco possui um custo c_{ij} embutido caso o arco $(i,j) \in E$ seja escolhido na solução ótima" [Morei01].

O custo mínimo total para a construção da rede será obtido neste trabalho através de árvore geradora mínima (AGM), pois existem vários algoritmos fáceis de se utilizar e bem difundidos, assim como o algoritmo de *Prim e Kruskal*, que podem ser vistos em [Corme90].

Como citado por [Norde01]**,** o objetivo em problemas de árvore geradora mínima é a construção de uma rede de custo mínimo que provê para todo nó na rede uma conexão com a fonte. Exemplos de problemas de árvore geradora mínima podem ser verificadas em várias situações desde o problema de construir uma rede de computadores que conectam todos computadores com algum servidor até o problema de construir um sistema de drenagem que conecta toda casa, em uma cidade, com o purificador de água.

Tendo uma AGM de um grafo, tem-se a forma ótima de se conectar o conjunto de nós de demanda ao fornecedor, surge então o problema do rateio deste custo mínimo entre os clientes de uma maneira justa, buscando conciliar duas metas, a de formação da rede ótima e a satisfação de todos.

Capítulo 3

Teoria dos Jogos Cooperativos

3.1 – Formulação de Um Jogo Cooperativo

A teoria dos jogos cooperativos tem sido freqüentemente utilizada para vários modelos de problemas de alocação de custos. Rateio de custo é a distribuição dos custos de um projeto entre seus participantes. Ela se aplica quando houver possibilidade de divisão da responsabilidade financeira de um projeto entre as entidades executoras, entre as entidades usuárias ou pagantes.

O rateio de custo tem dupla função. Por um lado é através dele que poderá ser assegurada uma necessária eqüidade na distribuição dos custos de um projeto. Por outro lado é através da conseqüente alocação de custos que poderá ser estimulada a eficiência econômica no uso que cada participante faz dos fatores de produção utilizados no projeto. Finalmente, esta alocação permitirá o estabelecimento de políticas de tarifação que igualmente estimulem a eficiência econômica no uso dos produtos e serviços gerados pelo projeto.

Em geral, um problema de alocação de custos tem as seguintes características. Existe um conjunto $N = \{1...n\}$ de usuários que querem cooperar para se obter um bem comum. A questão é como alocar o custo entre os usuários

deste conjunto, para que nenhum destes se sinta prejudicado e venha a desistir da cooperação [Geffa97].

Como citado por [Morei01], resolver o problema de como dividir o custo entre os usuários não é trivial. A solução para o problema deve ao mesmo tempo cobrir os custos e ser aceitável por todos os usuários. A Teoria dos Jogos Cooperativos deve então ser utilizada, já que esta possui resultados interessantes em vários problemas correlatos.

A Teoria dos Jogos lida com jogos cooperativos na forma de função característica, tendo os nós da rede como jogadores, formando o conjunto N e a função característica $c(S)$ definida para todo subconjunto $S \subseteq N$, onde S representa todos os subconjuntos de combinações possíveis de usuários pertencentes a N, essas combinações são denominadas coalizões, que é dado por $2^{|N|}$. O valor da função característica $c(S)$ para cada subconjunto $S \subseteq N$ corresponde ao custo da sub-rede ótima formada por este subconjunto.

Seja $N=\{1,2,...,n\}$ o conjunto finito de jogadores,
$c: 2^{|N|} \rightarrow IR$, será a função característica $\forall S \subseteq N$, onde $c(\varnothing)=0$
sendo $c(N)$ o custo que deve ser repartido entre os jogadores,
então o par $(N;c)$ é chamado jogo cooperativo

Na teoria dos jogos cooperativos várias aproximações para alocações de custos razoáveis têm sido sugeridas, entre elas, estão os conceitos de núcleo e *nucleolus*.

3.2 - Núcleo

É o espaço de todas as soluções que atendem à restrição de que a soma dos valores alocados a cada usuário seja igual ao custo da rede ótima e à restrição do valor alocado a cada jogador ser sempre menor ou igual ao custo que este teria ao se conectar de qualquer outra forma possível, ou seja, a soma das alocações atribuídas a um subconjunto dos jogadores da solução tem que ser menor ou igual ao custo determinado pela função característica deste subconjunto, logo as soluções do núcleo atendem as seguintes restrições:

Seja $X=\{x_1,...,x_n\}$ um vetor cujos elementos são os valores de alocações aos jogadores x_i, $1 \leq i \leq n$, sendo $n=|N|$.

$$\sum_{i=1}^{n} X_i = C(N) \longrightarrow$$ Custo mínimo total da rede

Esta equação estabelece que a soma dos custos alocados deve ser igual ao custo do projeto conjunto. Trata-se portanto de uma condição de efetividade, que assegura cobertura dos custos e garante a viabilidade financeira do projeto.

$$\sum_{i \in S} X_i \leq C(S)$$ Custo da sub-rede ótima usada para conectar o subconjunto S de jogadores

Pode-se ver que esta equação estabelece que a soma dos custos alocados a um subconjunto de jogadores é limitada pelo custo da melhor alternativa de atendimento exclusivo a esse subconjunto. Essas restrições asseguram que um subconjunto de usuários não terá custos alocados que superem os custos que terão em atuação isolada. Esta condição garante que os participantes do projeto terão atrativos para participarem do mesmo.

O núcleo é então formado por várias soluções, vetores de alocação, que se adaptam as restrições citadas, soluções estas que garantem aos jogadores que estes não teriam custos menores em outras coalizões. Contudo, assim como cita [Morei01], alguns vetores de alocação podem ser mais interessantes que outros,

para que esta escolha seja feita vários conceitos foram propostos, entre eles está o *nucleolus*.

3.3 - *Nucleolus*

Para que se obtenha uma escolha adequada dentre as diversas soluções possíveis, o *nucleolus* utiliza o conceito de excesso relativo, que é definido como:

Para um jogo $(N;c)$ e para um vetor de alocação de custos X,

Seja $e(X,S)= c(S) - X(S)$ o excesso de S relativo a X, $\forall \varnothing \neq S \subset N$,

e seja $e(X)$ um vetor em $\mathbb{R}^{(2^n -2)}$ cujos elementos são $e(X,S)$,

$\forall \varnothing \neq S \subset N$, em ordem não decrescente, ou seja:

$$e(X) = \begin{pmatrix} e(X, S_1) \\ \dots \\ e(X, S_2^n) \end{pmatrix}$$

onde $S_1, S_2, \dots, S_2^n$, estão arranjados de forma que $e(X, S_k) \leq e(X, S_{k+1})$

Através do vetor de excessos pode-se definir o núcleo do jogo como sendo o conjunto de vetores X que resultam num vetor $e(X)$, onde todos os elementos são excessos maiores ou iguais a zero.

O *nucleolus* do jogo é o conjunto de vetores X que maximizam $e(X)$ lexicograficamente. Assim como mostra [Morei01] a ordem lexicográfica pode ser definida como:

Suponha dois vetores x e y onde $x = \{x_1, x_2, \dots, x_n\}$ e $y = \{y_1, y_2, \dots, y_n\}$. Pode-se dizer que x é maior lexicograficamente que y, se existir um inteiro q, tal que $1 \leq q \leq n$, onde:

$x_i = y_i$ *para* $1 \leq i < q$

$x_q > y_q$

Logo $x \left[\begin{matrix} > \\ LO \end{matrix} \right] y$

A relação de maior ou igual lexicograficamente é definida se

$x \left[\begin{matrix} > \\ LO \end{matrix} \right] y$ ou $x = y$, logo $x \left[\begin{matrix} \geq \\ LO \end{matrix} \right] y$

Conclui-se que o *nucleolus* é formado pelo conjunto de vetores de alocação, pertencentes ao núcleo, tal que seu vetor *e(X)* seja maior ou igual lexicograficamente que qualquer outro vetor *e(X)* proveniente dos vetores de alocação do núcleo. Portanto, o *nucleolus* sobre o conjunto de alocações de entrada *X ={x| x pertença ao núcleo}*, é o conjunto definido por:

$$\left\{ x \;\middle|\; \begin{array}{l} x \in X \\ \text{Se } m \in X, \text{ então } e(x) \left[\begin{matrix} \geq \\ LO \end{matrix} \right] e(m) \end{array} \right\}$$

Como os elementos do vetor de excesso estão ordenados de forma crescente, e sabendo que o *nucleolus* é o conjunto de vetores de alocação que maximiza *e(X)* lexicograficamente, pode se definir que o *nucleolus* maximiza primeiramente o menor excesso dentre todas as coalizões possíveis, ou seja, seleciona o vetor de excessos que propõe uma melhor folga para as coalizões menos privilegiadas. Deste modo, como é citado por [Morei01], o *nucleolus* permite que o conjunto de membros mais ‘insatisfeitos’ fique o mais satisfeito possível. Feito isso, passa-se a tentar deixar o mais satisfeito possível o segundo conjunto de membros mais ‘insatisfeitos’, e assim por diante. Para que este processo seja realizado um algoritmo exato é utilizado neste trabalho.

O fato das condições de rateio serem lineares sugere a utilização de Programação Linear na solução do problema. As condições definiriam a região

das soluções viáveis e seria introduzido um critério de eqüidade que pudesse ser formulado como uma função objetiva linear.

O algoritmo que será utilizado, para se obter os objetivos relacionados, baseia-se numa seqüência de problemas de programação linear, onde o resultado do primeiro problema fornecerá uma solução com uma maior folga possível para uma ou mais coalizões mais 'insatisfeitas', folga esta que é cabível a todas as coalizões. Deste modo o resultado será a maximização do menor excesso dentre todas as coalizões possíveis.

O primeiro problema é montado com todas as restrições citadas na seção 3.2, ou seja, através da árvore geradora mínima obtém-se a restrição de efetividade, e para cada subconjunto de jogadores a soma de seus respectivos custos tem que ser menor ou igual à função característica para este subconjunto, isto é, o custo da sub-rede ótima formada por este subconjunto, qualquer solução que atenda a estas restrições faz parte do núcleo. Tendo a garantia de que a solução pertencerá ao núcleo, o próximo passo será maximizar lexicograficamente o vetor de excessos, obtendo então o *nucleolus*. Para que isto seja realizado este problema montado com as devidas restrições é passado ao algoritmo na qual encontra o máximo excesso, chamado de E_1, cabível a todas as coalizões, onde uma ou mais restrições atingem a igualdade com este excesso, maximizando assim o excesso mínimo.

O primeiro problema a ser resolvido será então:

$$LP(1)\begin{cases} Max\ E_1 \\ \sum_{i \in N} x_i = C(N) \\ \sum_{i \in S} x_i + E_1 \le C(S) \quad \begin{cases} \forall S \subset \{1...n\} \\ S \neq \varnothing;\ S \neq \{1...n\} \end{cases} \\ \forall i\ x_i \ge 0;\ E_1 \ge 0 \end{cases}$$

Depois de obtido o máximo excesso E_1 cabível a todos os subconjuntos de coalizões, onde uma ou mais restrições atingiram a igualdade com tal excesso, o próximo problema a ser resolvido será maximizar o ganho das mais 'infelizes' coalizões dentre todas aquelas coalizões que não obtiveram a igualdade com o excesso E_1. Assim sendo, o resultado de *LP(1)* será utilizado pelo segundo problema de programação linear *(LP(2))*, portanto o conjunto de coalizões mais 'insatisfeitos' em *LP(1)* que obtiveram a folga E_1, ou seja, restrições que alcançaram a igualdade no ótimo de *LP(1)*, será denominado por *OPT(1)* e será então utilizado como espaço de busca para *LP(2)*.

O segundo problema de programação linear fica então:

$$Max\ E_2$$

$$LP(2)\begin{cases} \sum_{i \in N} x_i = C(N) \\ \sum_{i \in S} x_i = C(S) - E_1 \quad \forall S \in OPT(1) \\ \sum_{i \in S} x_i + E_2 \le C(S) \begin{cases} \forall S \subset \{1...n\} \\ S \notin OPT(1) \\ S \neq \varnothing;\ S \neq \{1...n\} \end{cases} \\ \forall i\ x_i \ge 0;\ E_2 \ge 0 \end{cases}$$

Este segundo problema *LP(2)* maximiza o excesso E_2 para o segundo conjunto de coalizões mais 'insatisfeitos'. O processo continua com $(LP(K))_{k \ge 3}$ até que todos os subconjuntos de coalizões atinjam a igualdade com uma folga estabelecida pelos seus respectivos problemas de programação linear.

Assim como informa [Morei01], a obtenção do *nucleolus* é um processo de grande dificuldade, devido ao número exponencial de restrições. Isto

desanima o uso deste procedimento em instâncias de problemas com um número grande de jogadores, portanto na próxima seção será tratado o processo de decomposição que poderá ser utilizado, quando possível, para amenizar situações onde o número de jogadores for superior a um limite determinado de jogadores.

O exemplo abaixo mostra o algoritmo citado:

Dado um grafo, Figura 3, não direcional com quatro nós, cujos pesos nas ligações representam os custos das ligações entre os nós, onde o nó zero representa o único fornecedor e os demais nós sendo os usuários deste serviço, e cuja árvore geradora mínima pode ser verificada na Figura 3 através das ligações mais escuras , encontrar o *nucleolus* para este jogo.

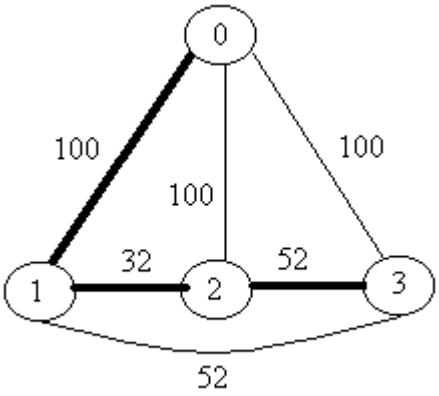

Figura 2: Grafo representando os jogadores e os custos

A formulação do jogo pode ser vista abaixo:

$N = \{1, 2, 3\}$

Tabela 1: Matriz de custos *C*

	1	2	3
0	100	100	100
1		32	52
2			52

$1)\ X_1 \leq 100$
$2)\ X_2 \leq 100$
$3)\ X_3 \leq 100$
$4)\ X_1 + X_2 \leq 132$
$5)\ X_1 + X_3 \leq 152$
$6)\ X_2 + X_3 \leq 152$
$7)\ X_1 + X_2 + X_3 = 184$

O primeiro problema de programação linear para este jogo obtém como resultado uma folga E_1 igual a 22,666667 para as restrições 4, 5 e 6, na qual alcançaram a igualdade no ótimo de *LP(1)*. O segundo problema de programação linear para o jogo obtém um excesso E_2 igual a 25,333333 para a restrição 3. O terceiro problema *LP(3)* obtém um excesso ótimo E_3 igual a 45,333333 para as restrições 1 e 2. Como todas as restrições de desigualdade atingiram a igualdade com seus respectivos ganhos, tornado-se, portanto, subconjuntos mais satisfeitos, o algoritmo finaliza retornando como *nucleolus* a alocação de custos (54,666667; 54,666667; 74,666667) correspondentes aos jogadores 1, 2 e 3.

3.4 - Técnica de Decomposição

Como num jogo cooperativo a função característica $C(S)$, $S \subseteq N$ é definida em todos os subconjuntos de usuários, ou seja $2^{|N|}$ restrições, assim com um número grande de jogadores o cálculo do *nucleolus* através do algoritmo exato se torna impraticável, deste modo podemos utilizar o recurso da decomposição, quando possível, para números maiores de usuários.

Suponha uma rede composta por um provedor comum de serviços conectado a vários clientes e o jogo $(N;c)$ referente a esta estrutura. Considere o caso na qual é produzida uma árvore geradora de custo mínimo com mais de uma aresta $(p > 1)$ incidindo no provedor comum 0. Isto eqüivale a existência de uma estrutura eficiente de coalizões $\{N_1,..., N_p\}$ no jogo $(N ; c)$ de modo que

$$c(N) = \sum_{i \in S}^{p} (N_i)$$

Granot e Huberman mostram que o *nucleolus* de um jogo original *(N;c)* é o produto cartesiano dos *nucleolus* dos jogos definidos sobre os componentes da estrutura eficiente de coalizões.

Desta forma são produzidos *p* jogos $(N_i ; c^i)$ $(i =1, ... ,p)$ tal que o *nucleolus* do jogo *(N;c)* é o produto cartesiano dos *nucleolus* dos jogos $(N_i;c^i)$ $(i=1, ... ,p)$, onde $\{ N_1, ... , N_p\}$ é uma estrutura eficiente de coalizões.

Dada uma matriz de custo $C = (c_{ij})$ representando o jogo *(N;c)* e supondo uma árvore geradora de custo mínimo com $p=2$ é possível construir uma matriz de custo $C'= (c'_{ij})$ de forma que

$$c'_{ij} \begin{cases} min\{c_{ij}, min_{k \in N2}\{c_{ik}\}\} & j = 0, i \in N_1 \\ min\{c_{ij}, min_{k \in N1}\{c_{ik}\}\} & j = 0, i \in N_2 \\ c_{ij} & caso\ contrário \end{cases}$$

Para exemplificar a decomposição suponha um jogo cooperativo com cinco jogadores *({1, ... ,5};c)* determinado pela matriz de custos *C* da Tabela 2.

Tabela 2: Matriz de custos c do jogo

		1	2	3	4	5
	0	2	5	7	2	3
	1		3	6	3	4
C =	2			4	4	3
	3				6	5
	4					2

Na Figura 4 temos uma árvore geradora mínima de custo mínimo para o jogo.

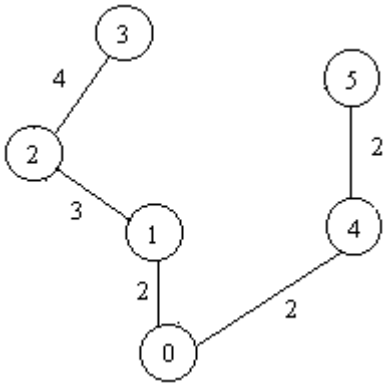

Figura 3: Árvore geradora mínima

Como o jogo possui cinco jogadores o número de subconjuntos seria (2^5 –1) = 32 –1=31, pois o conjunto vazio não faz parte das restrições. No entanto existe uma estrutura eficiente de coalizões sobre o jogo original ({1 , ... ,5}; c) sendo ela {{1,2,3},{4,5}}. Deste modo as novas matrizes de custo para os jogos ({1,2,3}; c^1}) e ({4,5}; c^2}) podem ser vista nas Tabelas 3 e 4 respectivamente

Tabela 3: Matriz de custos

	1	2	3
0	2	3	5
1		3	6
2			4

Tabela 4: Matriz de custos

	4	5
0	2	3
4		2

Nas Figuras 5 e 6 podem ser vistas suas respectivas árvores geradoras mínimas.

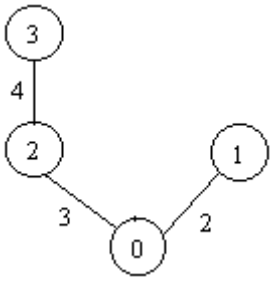

Figura 4: Árvore Geradora Mínima

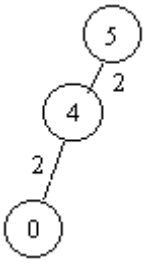

Figura 5: Árvore Geradora Mínima

O jogo ({1,2,3};c^1) ainda possui uma estrutura eficiente de coalizões {{1}, {2,3}}, logo este jogo é decomposto em ({1}; c^3) e ({2,3}; c^4). A Figura 7 mostra o resultado final da decomposição do jogo ({1 , ... ,5}; c) .

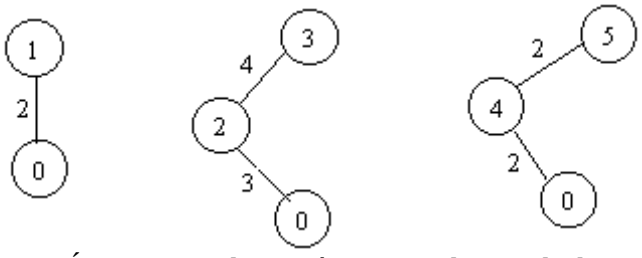

Figura 6: Árvores Geradoras mínimas resultantes da decomposição

O cálculo do *nucleolus* dos jogos resultantes da estrutura eficiente de coalizões do jogo original é calculado a seguir:

N_1 = {1}

Tabela 5: Matriz de custos c^3

	1
0	2

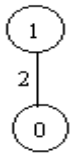

Figura 7: AGM

Nucleolus do jogo ({1}; c^3)
$X_1 = 2$

N_2 = {2, 3}

Tabela 6: Matriz de custos c^4

	2	3
0	3	5
2		4

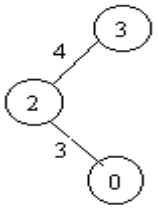

Figura 8: AGM

$X_2 \leq 3$
$X_3 \leq 5$
$X_2 + X_3 = 7$

Nucleolus do jogo *({2,3}; c^4)*
$X_2 = 2{,}5$
$X_3 = 4{,}5$

N_3 = {4, 5}

Tabela 7: Matriz de custos c^2

	4	5
0	2	3
4		2

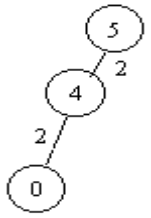

Figura 9: AGM

$X_4 \leq 2$	*Nucleolus* do jogo $(\{4,5\}; c^2)$
$X_5 \leq 3$	$X_4 = 1,5$
$X_4 + X_5 = 4$	$X_5 = 2,5$

Assim como Huberman e Granot mostraram, o *nucleolus* do jogo original $(\{1 , ... ,5\}; c)$ é o produto cartesiano dos *nucleolus* dos jogos $(\{1\}; c^3)$, $(\{2,3\}; c^4)$ e $(\{4,5\};c^2)$, onde $\{ N_1, N_2, N_3\}$ é uma estrutura eficiente de coalizões do jogo $(\{1 , ... ,5\}; c)$, logo seu *nucleolus* é (2; 2,5; 4,5; 1,5; 2,5), custo associado a cada um dos jogadores respectivamente, onde o somatório dos custos de cada jogador resulta num custo mínimo total igual a 13.

Verifica-se que a decomposição fez com que o número de restrições, que seriam 31 no jogo original ($2^5 - 1$), caísse para um total de 7 restrições na decomposição, o que facilita muito o processo de elaboração do jogo, onde em casos de um número grande de jogadores a elaboração de todas as restrições se torna impraticável.

Capítulo 4

Programação Linear

4.1 – Definição de Programação Linear

"A programação linear é considerada hoje como um instrumento-padrão que poupou muitos milhares ou milhões de dólares de companhias ou negócios de tamanho médio nos países industrializados do mundo, e seu uso em outros setores da sociedade está se expandindo rapidamente" [Hill88].

Sumariamente, a programação linear trata tipicamente com o problema de alocação de recursos limitados a atividades em competição, da melhor maneira possível. Este problema de alocação pode aparecer toda vez que alguém precise selecionar o nível de certas atividades que competem por recursos escassos necessários para desempenhá-las. A variedade de situações a que esta descrição se aplica é realmente diversificada. Entretanto, o ingrediente comum a cada uma destas situações é a necessidade de alocação de recursos a atividades.

"A programação linear usa um modelo matemático para descrever o problema em questão. O adjetivo "linear" significa que é requerido que todas as funções matemáticas neste modelo sejam funções lineares. A palavra "programação" aqui não se refere á programação de computadores; ao contrário, trata-se essencialmente de um sinônimo de planejamento. Assim, a programação

linear faz o planejamento de atividades para obter um resultado "ótimo", um resultado que alcance a melhor meta especificada (de acordo com o modelo matemático) entre as alternativas viáveis" [Hill88].

4.2 - Método Simplex

O Método Simplex é o procedimento geral para resolver problemas de programação linear, portanto este método será utilizado para resolver tais problemas, assim sendo o algoritmo de obtenção do *nucleolus* irá utilizá-lo seqüencialmente até atingir este objetivo.

O simplex parte de uma solução viável inicial, com alguma solução qualquer que satisfaz todas as restrições, solução básica possível, e sucessivamente obtém soluções básicas viáveis, cada uma melhor que a sua predecessora, até que seja alcançada uma solução ótima. Cada nova solução viável é obtida a partir de sua predecessora, transformando uma variável não-básica (variável que não estava na solução) em variável básica e transformando uma variável básica em uma variável não-básica. Uma variável básica viável é tida como ótima quando nenhuma das soluções básicas viáveis adjacentes é melhor que ela.

Estabelecimento do método:

É muito mais conveniente lidar com equações do que com relações de desigualdade. Por isso, o primeiro passo para estabelecermos o método simplex é converter as restrições funcionais de desigualdade em restrições equivalentes de igualdade. Isto é feito introduzindo-se variáveis de folga. Para ilustrar, temos a restrição

$$X_1 \leq 4.$$

A variável de folga para esta restrição é

$$X_3 = 4 - X_1,$$

a qual é exatamente a folga entre os dois lados da desigualdade. Portanto,

$$X_1 + X_3 = 4.$$

a constante original $X_1 \leq 4$ se mantém sempre que $X_3 \geq 0$. Consequentemente, $X_1 \leq 4$ é inteiramente equivalente ao conjunto de restrições

$$X_1 + X_3 = 4 \text{ com } X_3 \geq 0 \text{ e } X_1 \geq 0$$

assim, estas restrições, mais convenientes, são usadas em seu lugar.

Deste modo para cada restrição de desigualdade é inserida uma variável de folga, e o método simplex inicializa com estas variáveis de folga na base, pois todas as variáveis fora da base possuem valor zero. Assim sendo, verifica-se se a solução é ótima através da regra da parada, que finaliza se todos os coeficientes da função objetivo são menores ou iguais a zero.

Caso a solução não seja ótima, escolhe a coluna que tenha o maior valor negativo para ser a coluna pivô (variável entrando na base) e o menor valor positivo da relação B_i / a_{ij} , onde B_i é o valor que restringe a equação e a_{ij} é o coeficiente correspondente a linha *i* da variável que vai entrar na base, para ser a linha pivô, correspondente a variável que irá sair da base. Ao testar o método pode se ver que o valor zero dividido por um número negativo não é escolhido como menor relação B_i/a_{ij}.

A nova linha pivô será:

$$\text{Nova linha pivô} = \frac{\text{antiga linha pivô}}{\text{Número pivô}}$$

As demais linhas serão:

Nova linha = antiga linha – (coeficiente da coluna pivô)x (Nova linha pivô)

O método prossegue até que a regra da parada finalize o procedimento, e então se obtém a solução ótima.

O método compreenderá, portanto, dos seguintes passos:

1. Passo de inicialização:
   Identificar uma solução básica viável inicial.
2. Regra de parada:
   Verificar se a solução atual é ótima. Se for, pare. Caso contrário, siga para o passo 3.
3. Passo iterativo:
   3.1 Determinar a variável básica entrando.
   3.2 Determinar a variável básica saindo.
   3.3 Achar a nova solução compatível básica, e voltar ao passo2.

Restrições de igualdade serão tratadas através do método do Grande *M*, assim como cita [Hill88]. Este método baseia-se na introdução de uma variável artificial para restrições de igualdade, na qual seu valor na função objetivo corresponde a um número negativo muito grande, já que se pretende maximizar *Z*, esta variável artificial será obrigatoriamente expulsa da base, ficando portanto com o valor igual a zero, garantindo assim a igualdade da restrição, visto que toda restrição de igualdade não possui folga.

Entretanto, o método simplex exige que toda variável básica tenha um coeficiente zero na função objetivo, e a variável artificial é uma variável básica inicial. Para que este problema seja resolvido, o método procede como se a coluna para a variável artificial fosse a coluna pivô e sua restrição de igualdade fosse a linha pivô. Isto completa o trabalho adicional requerido no passo de inicialização para esse tipo de problema, e o método simplex prossegue como descrito anteriormente.

Suponha o exemplo abaixo:

Maximizar $Z = E_1$, onde

$E_1+X_1 \leq 100$
$E_1+X_2 \leq 100$
$E_1+X_3 \leq 100$
$E_1+X_1 + X_2 \leq 132$
$E_1+X_1 + X_3 \leq 152$
$E_1+X_2 + X_3 \leq 152$
$X_1 + X_2 + X_3 = 184$

Este problema é representado no tabela 8 e o método simplex irá utilizá-la para obter a solução ótima.

Grande M

Tabela 8: Primeiro Tableau do método simplex

	Coeficientes de												
VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	-1	0	0	0	0	0	0	0	0	0	M	0
X_4	0	1	1	0	0	1	0	0	0	0	0	0	100
X_5	0	1	0	1	0	0	1	0	0	0	0	0	100
X_6	0	1	0	0	1	0	0	1	0	0	0	0	100
X_7	0	1	1	1	0	0	0	0	1	0	0	0	132
X_8	0	1	1	0	1	0	0	0	0	1	0	0	152
X_9	0	1	0	1	1	0	0	0	0	0	1	0	152
X_{10}	0	0	1	1	1	0	0	0	0	0	0	1	184

Linha Pivô

Coluna Pivô

O primeiro passo será resolver o problema da equação de igualdade que completa o trabalho adicional requerido no passo de inicialização, logo o novo quadro pode ser visto na tabela 9.

Tabela 9: Segundo Tableau do método simplex

VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	-1	-M	-M	-M	0	0	0	0	0	0	0	-184M
X_4	0	1	1	0	0	1	0	0	0	0	0	0	100
X_5	0	1	0	1	0	0	1	0	0	0	0	0	100
X_6	0	1	0	0	1	0	0	1	0	0	0	0	100
X_7	0	1	1	1	0	0	0	0	1	0	0	0	132
X_8	0	1	1	0	1	0	0	0	0	1	0	0	152
X_9	0	1	0	1	1	0	0	0	0	0	1	0	152
X_{10}	0	0	1	1	1	0	0	0	0	0	0	1	184

Coluna Pivô Número Pivô Linha Pivô

Tabela 10: Terceiro Tableau do método simplex

VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	M-1	0	-M	-M	M	0	0	0	0	0	0	-84M
X_1	0	1	1	0	0	1	0	0	0	0	0	0	100
X_5	0	1	0	1	0	0	1	0	0	0	0	0	100
X_6	0	1	0	0	1	0	0	1	0	0	0	0	100
X_7	0	0	0	1	0	-1	0	0	1	0	0	0	32
X_8	0	0	0	0	1	-1	0	0	0	1	0	0	52
X_9	0	1	0	1	1	0	0	0	0	0	1	0	152
X_{10}	0	-1	0	1	1	-1	0	0	0	0	0	1	84

Tabela 11: Quarto Tableau do método simplex

VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	M-1	0	0	-M	0	0	0	M	0	0	0	-52M
X_1	0	1	1	0	0	1	0	0	0	0	0	0	100
X_5	0	1	0	0	0	1	1	0	-1	0	0	0	68
X_6	0	1	0	0	1	0	0	1	0	0	0	0	100
X_2	0	0	0	1	0	-1	0	0	1	0	0	0	32
X_8	0	0	0	0	1	-1	0	0	0	1	0	0	52
X_9	0	1	0	0	1	1	0	0	-1	0	1	0	120
X_{10}	0	-1	0	0	1	0	0	0	-1	0	0	1	52

Tabela 12: Quinto Tableau do método simplex

VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	-1	0	0	0	0	0	0	0	0	0	M	0
X_1	0	1	1	0	0	1	0	0	0	0	0	0	100
X_5	0	1	0	0	0	1	1	0	-1	0	0	0	68
X_6	0	2	0	0	0	0	0	1	1	0	0	-1	48
X_2	0	0	0	1	0	-1	0	0	1	0	0	0	32
X_8	0	1	0	0	0	-1	0	0	1	1	0	-1	0
X_9	0	2	0	0	0	1	0	0	0	0	1	-1	68
X_3	0	-1	0	0	1	0	0	0	-1	0	0	1	52

Tabela 13: Sexto Tableau do método simplex

VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	0	0	0	0	-1	0	0	1	1	0	M-1	0
X_1	0	0	1	0	0	2	0	0	-1	-1	0	1	100
X_5	0	0	0	0	0	2	1	0	-2	-1	0	1	68
X_6	0	0	0	0	0	2	0	1	-1	-2	0	1	48
X_2	0	0	0	1	0	-1	0	0	1	0	0	0	32
E_1	0	1	0	0	0	-1	0	0	1	1	0	-1	0
X_9	0	0	0	0	0	3	0	0	-2	-2	1	1	68
X_3	0	0	0	0	1	-1	0	0	0	1	0	0	52

Tabela 14: Sétimo Tableau do método simplex

VB	Z	E_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	B_i
Z	1	0	0	0	0	0	0	0	1-2/3	1-2/3	1/3	M-1+1/3	22,666
X_1	0	0	1	0	0	0	0	0	4/3-1	4/3-1	-2/3	1-2/3	54,666
X_5	0	0	0	0	0	0	1	0	4/3-2	4/3-1	-2/3	1-2/3	22,666
X_6	0	0	0	0	0	0	0	1	4/3-1	4/3-2	-2/3	1-2/3	2,6666
X_2	0	0	0	1	0	0	0	0	1-2/3	2/3	1/3	1/3	54,666
E_1	0	1	0	0	0	0	0	0	1-2/3	1-2/3	1/3	1/3-1	22,666
X_4	0	0	0	0	0	1	0	0	-2/3	-2/3	1/3	1/3	22,666
X_3	0	0	0	0	1	0	0	0	-2/3	1-2/3	1/3	1/3	74,666

A regra da parada mostra que o resultado é ótimo, finalizando, portanto, o método. Obtendo a solução ótima:

$X_1 = 54{,}666;\ X_2 = 54{,}666;\ X_3 = 74{,}666;\ E_1 = 22{,}666$

$X_4 = 22{,}666;\ X_5 = 22{,}666;\ X_6 = 2{,}666;\ X_7 = 0;\ X_8 = 0;\ X_9 = 0;\ X_{10} = 0$, portanto $Z = 22{,}666$

Como explica [Hill88], a existência de coeficientes zero na função objetivo correspondentes as variáveis não básicas, indicam que o problema possui mais que uma solução básica viável ótima, pois a inserção destas variáveis na solução não altera a função objetivo, já que o coeficiente é zero. Neste exemplo não se verifica a existência de múltiplas soluções.

Capítulo 5

Algoritmo Proposto

5.1 - Algoritmo Proposto

Ao realizar-se o estudo da literatura citada, verificou-se que o método simplex possuía a característica de retirar variáveis de folga da base, para que assim as variáveis do problema pudessem entrar na base e obter, portanto, seus respectivos valores, contudo essas variáveis de folga passavam a ter zero como valor, ou seja, todas as restrições correspondentes a estas variáveis de folga atingiam a igualdade, e o número dessas restrições correspondia ao número de variáveis do problema menos o número de variáveis artificiais, já que estas devem ser removidas obrigatoriamente da base, pois equações de igualdade não possuem folga. Tendo analisado tal característica, pode se concluir que se tais variáveis de folga não fossem variáveis artificiais, estas estavam sendo retiradas obrigatoriamente da base sem um motivo lógico, apenas para dar lugar para as variáveis do problema na base, ficando estas variáveis com valores grandes para que tais equações atingissem a igualdade.

Quando o simplex obtém como entrada o primeiro problema de programação linear do algoritmo, onde apenas uma restrição é de igualdade e o número de variáveis do problema é *n* (numero de jogadores + variável E_1), *(n-1)*

inequações estarão atingindo a igualdade, visto que, suas variáveis de folga estarão fora da base, portanto *(n-1)* subconjuntos estarão atingindo a igualdade, ficando as variáveis destas equações com valores grandes, para que a folga seja zero, o que não condiz com o problema em questão, onde se tem o objetivo de maximizar o excesso para estas equações.

Como o objetivo do algoritmo está em maximizar primeiramente o menor excesso dentre todas as coalizões possíveis, ou seja, seleciona o vetor de excesso que propõe uma melhor folga para as coalizões menos privilegiadas, ao fixar todas estas restrições na igualdade com o mesmo valor E_1 obtido não condiz com o objetivo do algoritmo, visto que, fixando uma ou mais destas restrições e não todas na igualdade, pode-se obter uma folga maior para as demais restrições, pois agora existem mais variáveis artificiais, Deste modo os subconjuntos se tornam mais satisfeitos.

O algoritmo exato foi então modificado da seguinte forma:

Pode-se verificar que estas *(n-1)*, restrições que atingiram a igualdade, são as que restringem a região de solução do problema, ou seja, formam o poliedro de soluções possíveis para o núcleo, onde *(n-1)* corresponde ao número total de jogadores. Dentre estas soluções possíveis, o algoritmo deve obter a solução que maximize lexicograficamente o vetor de excessos proveniente destas soluções. Deste modo o algoritmo, que será citado em seguida, irá maximizar o vetor de excessos resultante destas soluções, pois estas *(n-1)* coalizões são as que possuem o menor excesso dentre todas as $(2^{|N|} - 1)$ restrições, ao encontrar este vetor de excessos todas as variáveis já estarão fixadas com seus respectivos valores, logo o vetor de excessos para as $(2^{|N|} - 1)$ restrições já estará formado, portanto este resultado será o *nucleolus* obtido.

Para encontrar o vetor de excessos maior lexicograficamente dentre todos aqueles resultantes das soluções do núcleo, no primeiro passo o algoritmo fixa separadamente na igualdade cada uma destas *(n-1)* restrições com o excesso

E_1 obtido e executa o simplex com cada um destes problemas, verifica-se então se algum destes problemas de programação linear retorna um excesso E_2 maior que o E_1, para as *(n-2)* restrições, então escolhe o problema que obtiver o maior excesso E_2, se existir. Se obtiver este excesso E_2 maior que E_1 significa que o problema de programação linear que se tem no momento possui apenas uma restrição fixada na igualdade com E_1, portanto em seu próximo passo o algoritmo verifica se tem alguma restrição fixada na igualdade com o excesso E_1. Se existir o algoritmo irá fixar na igualdade separadamente cada uma destas *(n-2)* restrições com o excesso E_2 *obtido*, para assim verificar se algum destes problemas retornará um excesso E_3 maior que E_2 para as *(n-3)* restrições. Se não existir restrição fixada na igualdade com o excesso E_1, ou seja, não obteve E_2 maior que E_1 no primeiro passo, o algoritmo fixa separadamente cada combinação de duas das *(n-1)* restrições na igualdade com o excesso E_1 e verifica se o simplex retorna um excesso E_2 maior que E_1 para as *(n-3)* restrições. Escolhe-se portanto o problema que retornar o melhor excesso E_2, se existir. No terceiro passo verifica se existem duas restrições fixadas na igualdade com seus respectivos excessos, se existir fixa cada uma das *(n-3)* restrições restantes com o melhor excesso obtido para tais coalizões e escolhe o problema que retornar o maior excesso, que seja maior que o anterior, para as *(n-4)* restrições, se existir. Os próximos passos seguem o mesmo raciocínio, até que se obtenha as *(n-1)* coalizões mais satisfeitas com os excessos obtidos.

Suponha o exemplo de um jogo de cinco jogadores, onde o simplex é executado com as $(2^{|N|} - 1)$ restrições e obtém um excesso E_1, com cinco restrições atingindo a igualdade com este excesso. Supondo que as restrições 16, 17, 22, 27, 29 compõem estas cinco restrições. A Tabela 15 mostra os passos seguidos pelo algoritmo na tentativa de obter melhores excessos para as *(n-1)* restrições.

Tabela 15: Combinações

Combinações entre as *(n-1)* restrições
16
17
22
27
29
16 17
16 22
16 27
16 29
17 22
17 27
17 29
22 27
22 29
27 29
16 17 22

No primeiro passo o algoritmo tenta encontrar um melhor excesso para as *(n-2)* restrições fixando separadamente cada uma das *(n-1)* restrições com o excesso E_1 encontrado, contudo o algoritmo não obteve excesso maior que E_1 para as quatro demais restrições. Portanto o simplex foi executado *(n-1)* vez neste passo, sem nenhuma melhora obtida para estas restrições.

Se o algoritmo tivesse obtido um excesso melhor no passo citado anteriormente, este passo iria executar o simplex apenas 4 vezes, que seria a combinação de cada uma das *(n-2)* restrições com a restrição já fixada. Mas como não obteve melhor excesso o simplex foi executado mais dez vezes que é total de combinações das cinco restrições tomadas dois a dois ($C_{(n-1),\ 2} = O(n^2)$), na tentativa de obter um melhor excesso.

Um melhor excesso E_2 foi obtido para as *(n-3)* restrições fixando as restrições 16 e 17 na igualdade com E_1, este é o melhor pois o simplex não retornou um excesso maior que este ao fixar as demais combinações na igualdade com o excesso E_1 .

Como o passo anterior obteve um melhor excesso para as *(n-3)* restrições, então o algoritmo tenta fixar cada uma destas *(n-3)* restrições com o melhor excesso obtido para elas, na tentativa de obter uma folga maior para as *(n-4)* restrições, no entanto esta folga não foi obtida. Este passo exigiu que o simplex fosse rodado *(n-3)* vezes, que é o número de combinações das *(n-3)* restrições com as duas restrições que estão fixas na

16 17 27
16 17 29
16 17 22 27
16 17 22 29
16 17 27 29

Como o passo anterior não obteve melhor folga para as *(n-4)* restrições, o algoritmo fixa então duas das *(n-3)* restrições com o melhor excesso obtido para elas, que até então é o excesso E_2, para tentar obter um melhor excesso para as *(n-5)* restrições, mas este melhor excesso também não é obtido. O simplex é rodado três vezes neste passo, que é o número de combinações de *(n-3)* variáveis tomadas dois a dois, onde *n* é o total de variáveis do jogo $(E_1 + 5 \text{ jogadores}=6)$.

Como se pode verificar o algoritmo proposto irá obter um vetor de excessos maior lexicograficamente para as *(n-1)* coalizões, onde *n* é o número total de variáveis do problema. Isto já garante a obtenção de um vetor de excessos maior lexicograficamente para as $(2^{|N|} - 1)$ restrições do problema, isto pois, ao resolver o problema para as *(n-1)* restrições, que são as restrições que limitam o núcleo, todas as variáveis do problema já estarão com seus valores fixados, portanto o vetor total de excessos maior lexicograficamente já estará definido.

Para que sejam definidos quais são as *(n-1)* restrições, o simplex é executado com todas as $(2^N - 1)$ restrições do problema, retornando então o

máximo excesso E_1 e as *(n-1)* restrições que atingem a igualdade com este excesso, , ou seja, as *(n-1)* restrições mais insatisfeitas.

5.2 - Análise de Complexidade Teórica

O algoritmo exato avaliado se baseia em iterações do método simplex até que todas as restrições atinjam a igualdade, ou seja todas os $2^{|N|}$ subconjuntos de usuários.

Deste modo se apenas uma restrição por vez atinge a igualdade temos o método simplex executando $2^{|N|}$ vezes, tornando o algoritmo $O(2^N)$, portanto impraticável para um número grande de jogadores, no entanto se todas as restrições atingem a igualdade ao mesmo tempo, temos o método simplex executando uma única vez, mas se a cada vez que o método simplex é executado várias restrições atingem a igualdade juntas, a complexidade se torna polinomial.

O método simplex parte de um vértice inicial, ou seja pontos de interseção entre as restrições, à procura da solução ótima, no pior caso a solução ótima é o último vértice a ser procurado. No pior caso o simplex é exponencial, tomando como *n* o número de jogadores, tem-se o problema de programação linear com todas as combinações destas restrições. Logo tem-se:

$$C_2{}^n{}_{,2} + C_2{}^n{}_{,3} + ... + C_2{}^n{}_{,2}{}^n = 2^{2^n} - 2^n - 1 = 4^n - 2^n - 1 = O(4^n)$$

No caso médio o simplex é polinomial, pois encontra o melhor vértice no princípio.

Como este algoritmo exato, encontrado na literatura, não deixava clara a política utilizada na fixação das restrições com o excesso ótimo, uma proposta foi desenvolvida na seção anterior. Comprovou-se no capítulo 6 com os testes realizados, que sua complexidade modificou-se e ficou da seguinte forma:

O melhor caso ocorreria se cada passo do algoritmo obtivesse um excesso melhor para as demais restrições, sendo assim, no primeiro passo o simplex seria rodado *(n-1)* vezes, sendo este o número total de restrições que formam o poliedro de soluções do núcleo, e seria escolhido o problema que obtivesse o maior E_2. O segundo passo do algoritmo executaria o simplex *(n-2)* vezes, visto que uma das restrições já estaria fixa com E_1. O terceiro passo executaria o simplex *(n-3)* vezes, já que duas restrições já estariam fixas na igualdade com seus respectivos excessos, e assim sucessivamente até *((n-1) -(n-3))*, pois a última folga é uma conseqüência do problema anterior. Conclui-se então que a complexidade no melhor caso é polinomial da forma:

$$(n-1) + (n-2) + (n-3) + ... + 2 = \sum_{i=2}^{x} = (2 + x).(x-1) / 2 = O(x^2)$$

O pior caso ocorre quando nenhum dos passos do algoritmo obtém melhor folga para as restrições, sendo assim este executa o simplex em todas as combinações possíveis das *(n-1)* restrições, ou seja, no primeiro passo executa o simplex $C_{(n-1),1}$, como não obteve melhor excesso no primeiro passo, no segundo o simplex é executado $C_{(n-1),2}$, como não obtém melhor folga para as *(n-3)* restrições, o terceiro passo roda o simplex $C_{(n-1),3}$, e assim segue até $C_{(n-1),(n-2)}$.

A complexidade no pior caso é exponencial da forma:

$$C_{(n-1),1} + C_{(n-1),2} + C_{(n-1),3} + ... + C_{(n-1),(n-2)} = 2^{|N|}$$

Tendo em vista a complexidade de ordem exponencial, conclui-se que o algoritmo é impraticável para um número grande de jogadores. Neste caso, resolver jogos com número grande de jogadores com este algoritmo só se torna viável se existir uma estrutura eficiente de coalizões permitindo assim a decomposição do jogo, e como decompor um jogo é um processo fácil e rápido, este pode ser utilizado com êxito se possível.

Capítulo 6

Resultados e Análises

6.1 - Recursos Utilizados

O algoritmo exato para o cálculo do *nucleolus* foi implementado no ambiente Scilab, ambiente este utilizado no desenvolvimento de programas para a resolução de problemas numéricos, criado e mantido por um grupo de pesquisadores do *Institut de Recherche en Informatique et en Automatique*, INRIA, França. O Scilab é gratuito e distribuído com o código fonte, possui linguagem própria, e se caracteriza pela facilidade de desenvolvimento de protótipos, por estes motivos optou-se por sua utilização.

O desenvolvimento do algoritmo, assim como os testes, foram realizados em um computador com processador Athlon, 128Mb de memória RAM, 1,3 GHz e sistema operacional Windows 98.

6.2 - Resultados e Análise da Melhora Destes

Usando como exemplo de jogo cooperativo a topologia da rede UFLA da Figura12 em anexo, levantada por [Lopes02], visto que, esta cabe muito bem

nesta formulação de problemas, pois é uma situação em que vários usuários estão cooperando entre si, para poderem participar da rede de informação da melhor maneira possível. O significado das siglas desta topologia pode ser visto na Tabela 19 da seção 1 do anexo.

Como se pode verificar na Figura 12 do anexo o número de 17 jogadores, 2^{17} restrições, é um valor grande para ser tratado pelo algoritmo proposto. Tendo em vista a dificuldade, foi verificada a possibilidade de se decompor o jogo. A existência de uma estrutura eficiente de coalizões no jogo permite afirmar que o *nucleolus* do jogo original será o produto cartesiano dos *nucleolus* dos jogos definidos sobre os componentes da estrutura eficiente de coalizões assim como mostram [Grano81].

A decomposição obtida do jogo da topologia da UFLA e a matriz de custos *C* do jogo, podem ser verificadas respectivamente na Seção 2 do Anexo e na Tabela 20 da Seção 1 do Anexo.

O algoritmo foi executado para cada sub-jogo resultante da decomposição, dentre estes vale a pena destacar o sub-jogo $N_{6.2}$, em anexo na Seção 2, página 56, na qual possui um número de cinco jogadores. O algoritmo ao receber o jogo $N_{6.2}$ ele executa o método simplex, que retorna um excesso ótimo E_1 igual a 11, como o número de variáveis do jogo é igual a 6 (o número de jogadores + a variável E_1) e devido à existência de uma restrição de igualdade, o simplex retorna o máximo E_1 com 5 restrições atingindo a igualdade com esta folga, abaixo segue o resultado:

$E_1 = 11$

$X_{10} = 304; X_{11} = 236; X_{15} = 123; X_{16} = 304; X_{17} = 187$

Tal excesso fixava na igualdade as cinco restrições:

$X_{16} + X_{17} \leq 502 \quad \Rightarrow \quad X_{16} + X_{17} + E_1 = 502$

$X_{10} + X_{11} + X_{15} \leq 674 \Rightarrow X_{10} + X_{11} + X_{15} + E_1 = 674$

$X_{10} + X_{16} + X_{17} \leq 806 \Rightarrow X_{10} + X_{16} + X_{17} + E_1 = 806$

$$X_{10} + X_{11} + X_{15} + X_{16} \leq 978 \Rightarrow \quad X_{10} + X_{11} + X_{15} + X_{16} + E_1 = 978$$
$$X_{10} + X_{11} + X_{16} + X_{17} \leq 1042 \Rightarrow \quad X_{10} + X_{11} + X_{16} + X_{17} + E_1 = 1042$$

Como o algoritmo não obteve melhor excesso que E_1 ao fixar apenas uma das cinco restrições, citadas a cima, com este excesso, ele então fixou duas destas restrições com o excesso E_1, sendo elas $(X_{16} + X_{17} + E_1 = 502)$ e $(X_{10} + X_{11} + X_{15} + E_1 = 674)$, e então obteve um excesso E_2 maior que E_1 para as três demais restrições. A nova solução ótima obtida foi:

$$E_2 = 39{,}5$$
$$X_{10} = 275{,}5;\ X_{11} = 236;\ X_{15} = 151{,}5;\ X_{16} = 275{,}5;\ X_{17} = 215{,}5$$

As restrições ficaram fixadas na igualdade com seus respectivos excessos:

$$X_{16} + X_{17} \leq 502 \quad \Rightarrow \quad X_{16} + X_{17} + E_1 = 502$$
$$X_{10} + X_{11} + X_{15} \leq 674 \Rightarrow \quad X_{10} + X_{11} + X_{15} + E_1 = 674$$
$$X_{10} + X_{16} + X_{17} \leq 806 \Rightarrow \quad X_{10} + X_{16} + X_{17} + E_2 = 806$$
$$X_{10} + X_{11} + X_{15} + X_{16} \leq 978 \Rightarrow \quad X_{10} + X_{11} + X_{15} + X_{16} + E_2 = 978$$
$$X_{10} + X_{11} + X_{16} + X_{17} \leq 1042 \Rightarrow \quad X_{10} + X_{11} + X_{16} + X_{17} + E_2 = 1042$$

Como pode se ver o melhor excesso foi obtido no segundo passo do algoritmo. Os demais passos do algoritmo não forneceram excesso maior que 39,5, portanto o resultado obtido fornece um vetor de excessos maior lexicograficamente que qualquer outro vetor proveniente das possíveis soluções do núcleo.

Com o objetivo de comprovar a melhora através das modificações propostas para o algoritmo exato, este foi desenvolvido sem estas propostas de correções e executado com o jogo da topologia da rede UFLA, onde o *nucleolus*

para o jogo $N_{6.2}$ obtido pode ser visto na Tabela 16, onde está também representado o *nucleolus* obtido por [Lopes02], e o *nucleolus* obtido pelo algoritmo proposto.

Tabela 16: *Nucleolus* do jogo

	***Nucleolus* do sub-jogo $N_{6.2}$ obtido pelo algoritmo proposto**	***Nucleolus* do sub-jogo $N_{6.2}$ obtido pelo algoritmo exato encontrado na literatura**	***Nucleolus* do sub-jogo $N_{6.2}$ obtido por [Lopes02]**
X_{10}	275,5	304	280
X_{11}	236	236	190
X_{15}	151,5	123	190
X_{16}	275,5	304	314
X_{17}	215,5	187	180

A solução obtida pelo algoritmo exato é um resultado melhor que o encontrado por [Lopes02]. Esta melhora pode ser vista através da comparação entre os vetores de excesso obtido pelos dois resultados na Tabela 17.

Como se pode verificar na Tabela 16 os resultados obtidos pelo algoritmo proposto são mais justos, em relação ao conceito de *nucleolus*, que os resultados obtidos pelo algoritmo exato original. Veja que o jogador X_{10}, representado na Figura 11 pelo nó 10, possui um custo alocado mais justo em relação ao valor obtido pelo algoritmo exato, visto que, todos os demais jogadores dependem dele para se conectarem ao fornecedor, o X_{16} ficou com um custo muito alto determinado pelo algoritmo exato, no algoritmo proposto este custo diminuiu, o que é justo, já que o X_{17} depende dele para se conectar, com isso o X_{17} tem seu valor aumentado pelo algoritmo proposto. Como o X_{11} possui um custo da ligação próximo ao custo do X_{16}, que está em seu mesmo nível na árvore geradora mínima da Figura 11, o algoritmo proposto o mantém com este custo. Como o X_{15} possui um custo de ligação muito menor que o custo

alocado ao jogador X_{17} que está no mesmo nível que o X_{15}, seu valor é aumentado, esta diferença é utilizada para minimizar a sobrecarga do X_{10}.

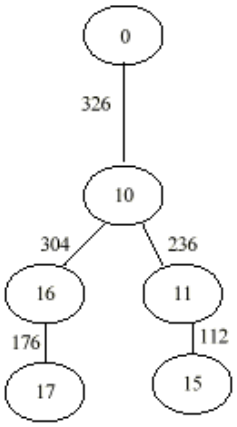

Figura 10: AGM do jogo $N_{6.2}$

O vetor de excessos para o jogo $N_{6.2}$ proveniente do algoritmo proposto, assim como os vetores de excessos obtidos pelo algoritmo exato e por [Lopes02] estão representados na Tabela 17.

Tabela 17: Vetores de excessos

Vetor de excessos a partir do algoritmo proposto	**Vetor de excesso a partir do algoritmo exato**	**Vetor de excesso a partir de [Lopes02]**
11	11	4
11	11	8
39,5	11	12
39,5	11	14
39,5	11	32
39,5	22	36
50,5	22	44
50,5	22	46
50,5	22	48
50,5	22	58

79	68	78
79	79	78
79	79	82
79	79	82
90	90	92
99,5	90	92
110,5	124	122
118,5	128	122
124	135	126
135	135	126
139	139	136
139	139	136
139	139	138
163,5	192	146
163,5	203	170
174,5	203	182
178,5	207	216
203	252	216
223,5	296	260
263	320	260

Verificando-se os vetores de excessos resultantes do algoritmo exato e por [Lopes02] e sabendo que o *nucleolus* é o conjunto de alocações de entrada que primeiro maximiza o menor excesso dentre todas as soluções possíveis, fazendo assim com que os jogadores mais 'insatisfeitos' se tornem mais 'satisfeitos', conclui-se que a solução obtida pelo algoritmo exato para o sub-jogo $N_{6.2}$ é melhor que a solução obtida em [Lopes02], já que o jogador mais 'infeliz', correspondente ao jogador que possui o menor excesso, obteve uma folga maior no resultado do algoritmo (11>4).

Pode-se ver que as modificações no algoritmo fizeram com que os subconjuntos correspondentes à linha 3 a 5 na Tabela 17, que antes eram fixados na igualdade com a folga 11, obtivessem uma maior folga deixando-os mais satisfeitos, logo este novo vetor de excesso é maior lexicograficamente que o vetor gerado pelo algoritmo exato.

O algoritmo proposto obteve o vetor de excessos maior ou igual lexicograficamente que qualquer outro vetor de excessos proveniente deste jogo. Isto foi possível, pois o algoritmo tentou fixar na igualdade apenas uma das restrições, dentre aquelas que tinham obtido a igualdade com o excesso 11, no entanto não obteve melhor excesso para as quatro demais restrições. O algoritmo tentou então fixar na igualdade de duas em duas restrições com este excesso 11, e então obteve um excesso 39,5 para as demais três restrições, o que tornou o vetor de excessos maior lexicograficamente, e assim prosseguiu mas não obteve maior excesso que 39,5 para as duas demais restrições. Os valores alocados a cada jogador resultam no vetor de excessos da Tabela 17.

O *nucleolus* do jogo cooperativo da topologia da rede UFLA obtido a partir do produto cartesiano dos *nucleolus*, encontrados através do algoritmo proposto, dos jogos definidos sobre os componentes da estrutura eficiente de coalizões, mostradas na Seção 2 do Anexo, pode ser visto na Tabela 18.

Tabela 18: *Nucleolus* do jogo da topologia da rede UFLA

Valor do *Nucleolus* obtido para cada departamento através do algoritmo exato com as modificações especificadas	**Valor pago por cada departamento igual ao custo da fibra que chega nele, núcleo ≠ ∅ (primeira proposta)**
$X_1 = 274$	$X_1 = 274$
$X_2 = 1300$	$X_2 = 1300$
$X_3 = 54.666667$	$X_3 = 100$
$X_4 = 54.666667$	$X_4 = 32$
$X_5 = 74.666667$	$X_5 = 52$
$X_6 = 304$	$X_6 = 304$
$X_7 = 169$	$X_7 = 288$
$X_8 = 169$	$X_8 = 50$
$X_9 = 144$	$X_9 = 144$
$X_{10} = 275.5$	$X_{10} = 326$
$X_{11} = 236$	$X_{11} = 236$

$X_{12} = 158$	$X_{12} = 326$
$X_{13} = 188$	$X_{13} = 104$
$X_{14} = 158$	$X_{14} = 74$
$X_{15} = 151.5$	$X_{15} = 112$
$X_{16} = 275.5$	$X_{16} = 304$
$X_{17} = 215.5$	$X_{17} = 176$

Como pode se ver, os jogadores X_1, X_2 *e* X_9 permaneceram com os seus respectivos valores em relação à primeira proposta, visto que, estes estão ligados diretamente ao fornecedor e não existe nenhum jogador dependente deles para se conectar a fonte 0, logo têm que arcar com seus próprios custos .

O X_4 e o X_5 estão pagando mais, pois estes dependem do X_3 para se ligarem à fonte 0, e o custo do X_3 é alto para se conectar ao fornecedor. Se não houvesse uma divisão deste custo o X_3 poderia se sentir insatisfeito de pagar o custo sozinho. O X_6 permaneceu com o mesmo valor, para ele este valor é satisfatório, pois nenhum jogador depende dele para se conectar, mas ele depende do X_3.

O X_7, tem o custo diminuído, custo que era alto para se ligar à fonte, mas como o X_8 depende deste para se conectar, os custos são divididos e o X_8 tem seu custo aumentado.

O X_{13} e o X_{14} dependem do X_{12} para se conectarem, logo o custo do X_{12}, que era alto diminui e os custos do X_{13} e do X_{14} aumentam. O X_{16} tem o seu custo diminuído , visto que, o X_{17} depende dele para se conectar. O X_{15} que possuía custo baixo, teve este aumentado, já que ele depende de outros jogadores, que possuem altos custos, para se conectar ao fornecedor. Logo o X_{10}, que permite a ligação destes jogadores X_{11}, X_{12} , X_{13}, X_{14}, X_{15}, X_{16} *e* X_{17} ao fornecedor, tem o custo diminuído.

6.3 - Análise da Complexidade Prática

Ao executar o algoritmo com o jogo $N_{6.2}$ de cinco jogadores o tempo necessário para a obtenção do resultado foi de 1 minuto e 93 segundos. Como o tempo de execução do método simplex foi na média de 5 segundos para este problema, pode-se concluir que o método simplex foi executado vinte duas vezes, que foi o número de vezes avaliado teoricamente. Executando um exemplo com seis jogadores foi avaliado um tempo de processamento igual a 21 minutos, o tempo de execução do método simplex foi na média de 33 segundos para este problema. Um jogo com número maior de jogadores não foi testado devido à dificuldade de montar as $(2^{|N|} - 1)$ restrições, assim como a complexidade exponencial do jogo.

6.4 – Instâncias Possíveis de Solução

Tendo verificado uma complexidade de ordem exponencial, e confirmado esta através da complexidade prática, pode-se dizer que a utilização do algoritmo proposto é recomendado apenas para um pequeno número de usuários, pois além da complexidade, a obtenção das $(2^{|N|} - 1)$ restrições é uma tarefa difícil e trabalhosa em situações de jogos com um grande número de usuários, portanto nestes casos o algoritmo só será utilizado se existir uma estrutura eficiente de coalizões, permitindo então a decomposição do jogo.

7 - Conclusão

Ao realizar-se o estudo do algoritmo exato para o cálculo do *nucleolus*, verificou-se que este necessitava de modificações para que o *nucleolus* pudesse ser obtido. Um algoritmo foi então proposto e testes foram realizados. Ao analisar estes testes pôde-se comprovar que realmente faltavam informações no algoritmo exato encontrado na literatura, completando então este algoritmo, o *nucleolus* pôde então ser obtido.

Estas correções, no entanto não melhoraram a complexidade do algoritmo, já que tais correções não tinham este objetivo, mas a complexidade se manteve.

Ao verificar uma complexidade exponencial no pior caso para o algoritmo proposto, conclui-se que este deve ser utilizado em jogos cooperativos com um número pequeno de jogadores, pois não só pela complexidade do algoritmo, como também pela dificuldade em formular todas as $(2^{|N|} - 1)$ restrições do jogo. Para um número grande de jogadores o algoritmo poderá ser utilizado se existir uma estrutura eficiente de coalizões que viabilize a decomposição do jogo, onde o *nucleolus* do jogo original será formado pelo produto cartesiano dos *nucleolus* dos sub-jogos provenientes da estrutura eficiente de decomposição.

7.1 – Trabalhos Futuros

Propõe-se para um futuro trabalho a melhora da complexidade do algoritmo proposto, e a utilização de políticas de escolha das $(n-1)$ restrições dentre as $(2^{|N|} - 1)$, através de heurísticas, pois formular estas restrições para um número grande de jogadores é uma tarefa difícil, assim como a complexidade, já

que somente estas restrições são responsáveis pela obtenção do *nucleolus*, visto que elas restringem a região n-dimensional (poliedro) de soluções possíveis do núcleo do jogo.

REFERÊNCIAS BIBLIOGRÁFIAS

[Corme90]CORMER, T.C.; LEISERSON, C.E.; RIVEST, R.L. **Introduction to Algorithms.** McGraw- Hill Book Company, 1990.

[Grano81]GRANOT, D.; HUBERMAN, G.. **Minimum Cost Spanning Tree Games**. Mathematical Programming, 1981, 21: 1-18.

[Geffa97]GEFFARD, J.; LUTTON, J.; PREMAOR, F. **An Algorithm to Compute the *Nucleolus* of the Minimum Cost Spanning Tree Game.** France, 1997.

[Hill88]HILLIES, F.S.; Lieberman, G.J. **Introdução à Pesquisa Operacional.** 3.ed. São Paulo: Ed. Campus LTDA, 1988. 805p.

[Lopes02]LOPES, J.M. **Alocação de Custos e Lucros em Redes de Informação.** Lavras: UFLA, 2002. 55p. (Monografia – Graduação em Ciência da Computação)

[Morei01]MOREIRA, R.C. **Estudo dos Problemas de alocação de custos em Redes de Acesso**. Belo Horizonte: UFMG, 2001. 80 p. (Dissertação – Mestrado em Ciência da Computação, Instituto de Ciências Exatas).

[Norde01]NORDE H.; MORETTI S.; TIJS, S. **Minimum Cost Spanning Tree Games and Population Monotonic Allocation Schemes**: ISSN 0924-7815, 2001. 18p.

ANEXO

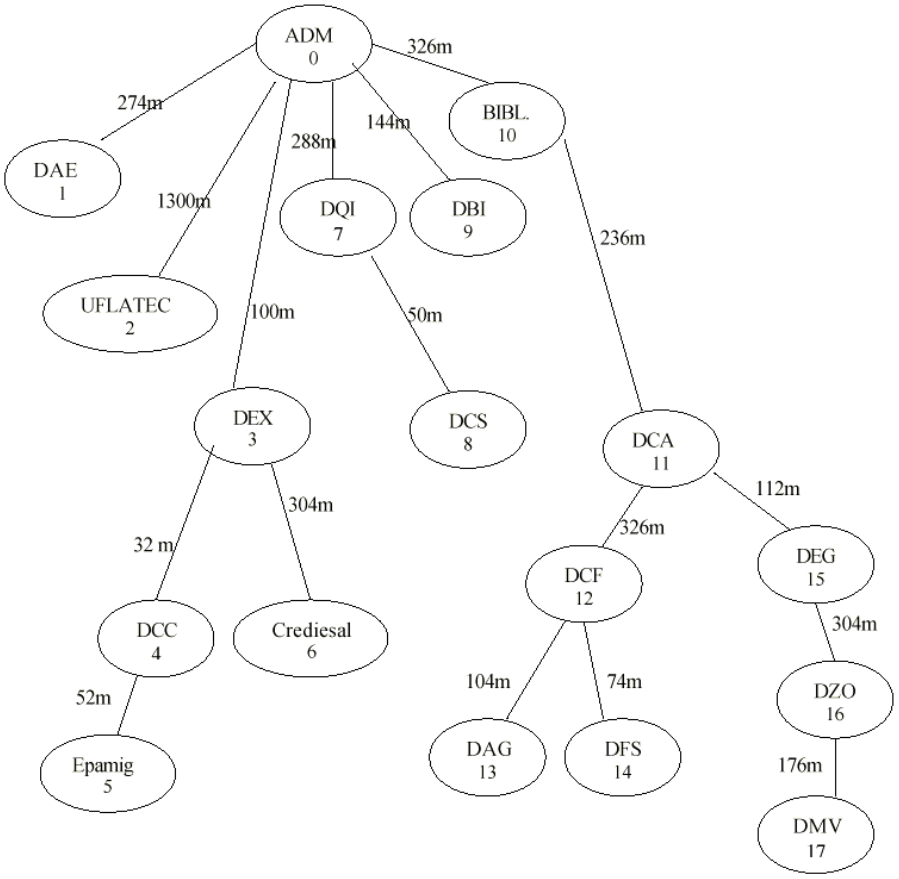

Figura 11: Topologia da Rede UFLA

1 - Complemento da Topologia da Rede UFLA

Tabela 19: Siglas para a Topologia da Rede UFLA

ADM: Prédio de Administração
BIBL: Biblioteca
CREDESAL: Banco Credesal
DAE: Departamento de Administração e Economia
DAG: Departamento de Agricultura
DBI: Departamento de Biologia
DCA: Departamento de Ciência Alimentos
DCC: Departamento de Ciência da Computação
DCF: Departamento de Ciências Florestais
DCS: Departamento de Solos
DEG: Departamento de Engenharia
DEX: Departamento de Exatas
DFS: Departamento de Fitopatologia e Entomologia
DMV: Departamento de Medicina Veterinária
DQI: Departamento de Química
DZO: Departamento de Zootecnia
EPAMIG: Empresa de Pesquisa Agropecuária de Minas Gerais
UFLATEC: Centro de Tecnologia e Informática

Tabela 20: Matriz de Custo do Grafo da Rede UFLA

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	274	1300	100	100	100	304	288	288	144	326	326	326	326	326	326	326	326
1		1300	274	274	274	304	288	288	274	326	326	326	326	326	326	326	326
2			1300	1300	1300	1300	1300	1300	1300	1300	1300	1300	1300	1300	1300	1300	1300
3				32	52	304	288	288	144	326	326	326	326	326	326	326	326
4					52	304	288	288	144	326	326	326	326	326	326	326	326
5						304	288	288	144	326	326	326	326	326	326	326	326
6							304	304	304	326	326	326	326	326	326	326	326
7								50	288	326	326	326	326	326	326	326	326
8									288	326	326	326	326	326	326	326	326
9										326	326	326	326	326	326	326	326
10											236	326	326	326	236	304	304
11												380	326	326	112	304	176
12													104	74	326	326	326
13														104	326	326	326
14															326	326	326
15																304	304
16																	176

2 - Decomposição do Jogo da Topologia da Rede UFLA

$N_1 = \{1\}$

C^1

Tabela 21: matriz de custos de N_1

	1
0	274

Nucleolus do jogo (N_1, C^1)
X_1=274

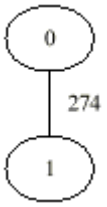

Figura 12: AGM N_1

$N_2 = \{2\}$

C^2

Tabela 22: matriz de custos de N_2

	2
0	1300

Nucleolus do jogo (N_2, C^2)
X_2=1300

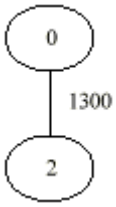

Figura 13: AGM N_2

$N_3 = \{3, 4, 5, 6\}$

C^3

Tabela 23: matriz de custos de N_3

	3	4	5	6
0	100	100	100	304
3		32	52	304
4			52	304
5				304

Decomposto em $N_{3.1}$={6}
$N_{3.2}$={3, 4, 5}

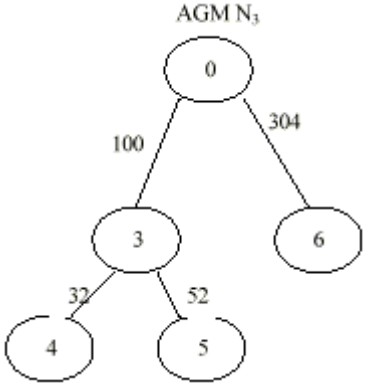

Figura 14: AGM N_3

$N_{3.1} = \{6\}$

$C^{3.1}$

Tabela 24: matriz de custos de $N_{3.1}$

	6
0	304

Nucleolus do jogo $(N_{3.1}, C^{3.1})$
X_6=304

Figura 15: AGM $N_{3.1}$

$N_{3.2} = \{3, 4, 5\}$

$C^{3.2}$

Tabela 25: matriz de custos de $N_{3.2}$

	3	4	5
0	100	100	100
3		32	52
4			52

$X_3 \leq 100$
$X_4 \leq 100$
$X_5 \leq 100$
$X_3 + X_4 \leq 132$
$X_3 + X_5 \leq 152$
$X_4 + X_5 \leq 152$
$X_3 + X_4 + X_5 = 184$

Nucleolus do jogo $(N_{3.2}, C^{3.2})$
$X_3 = 54{,}666667$
$X_4 = 54{,}666667$
$X_5 = 74{,}666667$

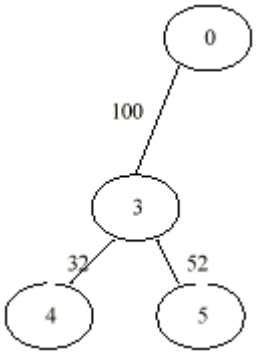

Figura 16: AGM $N_{3.2}$

$N_4 = \{7, 8\}$

C^4

Tabela 26: matriz de custos de N_4

	7	8
0	288	288
7		50

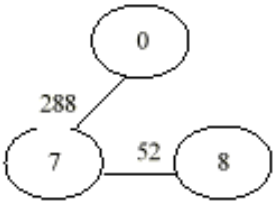

Figura 17: AGM N_4

$X_7 \leq 288$
$X_8 \leq 288$
$X_7 + X_8 = 338$

Nucleolus do jogo (N_4, C^4)
$X_7 = 169$
$X_8 = 169$

$N_5 = \{9\}$

C^5

Tabela 27: matriz de custos de N_5

	9
0	144

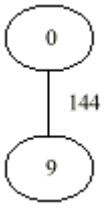

Figura 18: AGM N_5

Nucleolus do jogo (N_5, C^5)
$X_9 = 144$

$N_6 = \{10, 11, 12, 13\}$

C^6

Tabela 28: matriz de custos de N_6

	10	11	12	13	14	15	16	17
0	326	326	326	326	326	326	326	326
10		236	326	326	326	236	304	304
11			380	326	326	112	304	176
12				104	74	326	326	326
13					104	326	326	326
14						326	326	326
15							304	304
16								176

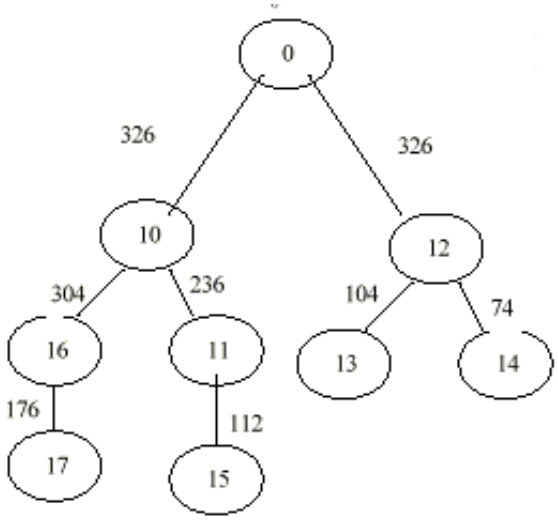

Figura 19: AGM N_6

Decomposto em $N_{6.1} = \{12, 13, 14\}$ e $N_{6.2} = \{10, 11, 15, 16, 17\}$

$N_{6.1} = \{12, 13, 14\}$

$C^{6.1}$

Tabela 29: matriz de custos de $N_{6.1}$

	12	13	14
0	326	326	326
12		104	74
13			104

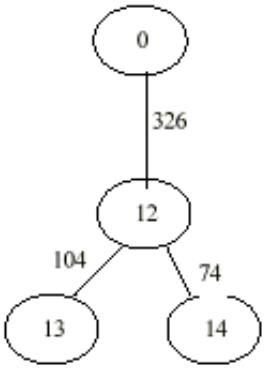

Figura 20: AGM $N_{6.1}$

$X_{12} \leq 326$
$X_{13} \leq 326$
$X_{14} \leq 326$
$X_{12} + X_{13} \leq 430$
$X_{12} + X_{14} \leq 400$
$X_{13} + X_{14} \leq 430$
$X_{12} + X_{13} + X_{14} = 504$

Nucleolus do jogo $(N_{6.1}, C^{6.1})$
$X_{12} = 158$
$X_{13} = 188$
$X_{14} = 158$

$N_{6.2}$ = {10, 11, 15, 16, 17}

$C^{6.2}$

Tabela 30: matriz de custos de $N_{6.2}$

	10	11	15	16	17
0	326	326	326	326	326
10		236	236	304	304
11			112	304	304
15				304	304
16					176

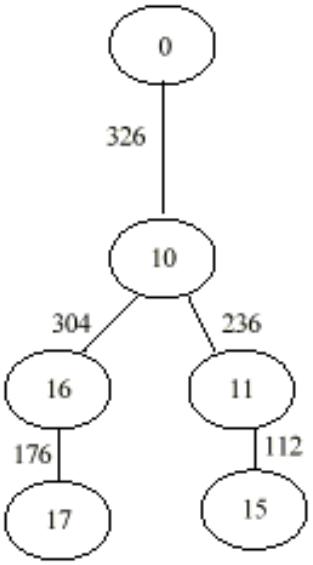

Figura 21: AGM $N_{6.2}$

$X_{10} \leq 326$
$X_{11} \leq 326$
$X_{15} \leq 326$
$X_{16} \leq 326$
$X_{17} \leq 326$
$X_{10} + X_{11} \leq 562$
$X_{10} + X_{15} \leq 562$
$X_{10} + X_{16} \leq 630$
$X_{10} + X_{17} \leq 630$
$X_{11} + X_{15} \leq 438$
$X_{11} + X_{16} \leq 630$
$X_{11} + X_{17} \leq 630$
$X_{15} + X_{16} \leq 630$
$X_{15} + X_{17} \leq 630$
$X_{16} + X_{17} \leq 502$
$X_{10} + X_{11} + X_{15} \leq 674$
$X_{10} + X_{11} + X_{16} \leq 866$
$X_{10} + X_{11} + X_{17} \leq 866$
$X_{10} + X_{15} + X_{16} \leq 866$
$X_{10} + X_{15} + X_{17} \leq 866$
$X_{10} + X_{16} + X_{17} \leq 806$
$X_{11} + X_{15} + X_{16} \leq 742$
$X_{11} + X_{15} + X_{17} \leq 742$
$X_{11} + X_{16} + X_{17} \leq 806$
$X_{15} + X_{16} + X_{17} \leq 806$
$X_{10} + X_{11} + X_{15} + X_{16} \leq 978$
$X_{10} + X_{11} + X_{15} + X_{17} \leq 978$
$X_{10} + X_{11} + X_{16} + X_{17} \leq 1042$
$X_{10} + X_{15} + X_{16} + X_{17} \leq 1042$
$X_{11} + X_{15} + X_{16} + X_{17} \leq 918$

$X_{10} + X_{11} + X_{15} + X_{16} + X_{17} = 1154$

Nucleolus do jogo $(N_{6.2}, C^{6.2})$:

$X_{10} = 275{,}5$
$X_{11} = 236$
$X_{15} = 151{,}5$
$X_{16} = 275{,}5$
$X_{17} = 215{,}5$