FREDERICO PAIVA QUINTÃO

O PROBLEMA DA ÁRVORE DE CUSTO MÍNIMO COM K ARESTAS: REFORMULAÇÕES E RELAXAÇÃO LAGRANGEANA

Belo Horizonte
09 de julho de 2008

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

O PROBLEMA DA ÁRVORE DE CUSTO MÍNIMO COM K ARESTAS: REFORMULAÇÕES E RELAXAÇÃO LAGRANGEANA

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

FREDERICO PAIVA QUINTÃO

Belo Horizonte
09 de julho de 2008

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

O Problema da Árvore de Custo Mínimo com k arestas: Reformulações e Relaxação Lagrangeana

FREDERICO PAIVA QUINTÃO

Dissertação defendida e aprovada pela banca examinadora constituída por:

Prof. Alexandre Salles da Cunha – Orientador
Departamento de Ciência da Computação/UFMG

Prof. Geraldo Robson Mateus
Departamento de Ciência da Computação/UFMG

Prof. Carlos Roberto Venâncio de Carvalho
Departamento de Engenharia de Produção/UFMG

Prof. Rodney Rezende Saldanha
Departamento de Engenharia Elétrica/UFMG

Prof. Abílio Lucena
Departamento de Administração/UFRJ

Belo Horizonte, 09 de julho de 2008

Resumo

Dado um grafo $G = (V, E)$, com custos nos vértices de V e nas arestas de E, o *Problema da Árvore de Custo Mínimo com k Arestas* (k-ACM) consiste em encontrar uma sub-árvore T em G com exatas k arestas, com o objetivo de que o custo de T seja mínimo. Este problema possui aplicações nas áreas de arrendamento de campos de petróleo, telecomunicações e roteamento, dentre outras.

Neste trabalho, são propostas reformulações de Programação Inteira que se baseiam em uma transformação realizada sobre o grafo de entrada do problema. Após uma análise empírica do comportamento de algoritmos *Branch-and-Bound* que empregam as reformulações, foi proposta uma Relaxação Lagrangeana para uma delas. Esta Relaxação, que visa obter limites inferiores para o problema, foi integrada a uma heurística que se mostrou capaz de gerar limites superiores de qualidade para o mesmo.

Palavras-chave: k-ACM, Programação Inteira, Heurística Lagrangeana, Árvore Geradora Mínima

Abstract

Given a graph $G = (V, E)$ with costs associated to the vertices of V and to the edges of E, the k-Cardinality Tree Problem (KCT) seeks a minimal cost sub-tree T of G with exactly k edges. This problem has applications in, among others, oil field leasing, telecommunications and routing.

In this dissertation, we propose Integer Programming reformulations for the KCT, based upon a tranformation accomplished over the input graph. After an empirical analysis of our formulations, we implemented a Lagrangian Relaxation approach for the strongest of them. The aim of this approach is to yield lower bounds for the problem. A Lagrangian Heuristic, that was embedded in the method, was able to generate upper bounds that compare favourably to the best in the literature.

Keywords: KCT, Integer Programming, Lagrangian Heuristic, Minimum Spanning Tree

My life is such a mess, let's have a Brahma.

(Eterno maestro Antônio Carlos Brasileiro de Almeida Jobim)

Agradecimentos

A lista de pessoas para agradecer é enorme, mas obviamente começa pelos meus pais, Carlos e Marília, e meu irmão Rodrigo, que me deram muito apoio, em todos os níveis imagináveis, para que eu pudesse realizar o **nosso sonho** de terminar um curso de mestrado.

Em seguida, gostaria de mencionar aqueles que sempre me incentivaram e ajudaram do ponto de vista acadêmico. Seguindo cronologicamente para não ser injusto, gostaria de agradecer aos seguintes pesquisadores: Geraldo Robson Mateus (meu orientador por quase 5 anos), Antonio Alfredo F. Loureiro, Fabíola Guerra Nakamura (eterna chefe que me deu a oportunidade de trabalhar com as coisas que eu gostava), Gustavo Campos Menezes, Flávio Cruzeiro, Martin Gomez Ravetti, Wagner M. Aioffi e Gleicy Aparecida Cabral. Um agradecimento especial ao prof. Alexandre Salles da Cunha, que me mostrou a beleza da Otimização Combinatória, me apresentou um problema maravilhoso, me ensinou como escrever um bom texto científico, foi um orientador altamente profissional, um grande amigo, e não se importou com a minha demora para entregar resultados, textos, revisões, etc.

Finalmente, gostaria de agradecer a todos os amigos que fiz dentro e fora da UFMG nestes últimos 6 anos. Dentre os amigos de classe, nunca irei me esquecer do apoio de Fernando A. Fernandes Júnior, Anísio Mendes Lacerda, Cristiano Bola Gato, Bruno Pontes, Rodrygo Teodoro, Felipe B3, Erickson Rangel e Felipe Louback. Na área de pesquisa, os amigos do Lapo: Wagner M. Aioffi, Pedro Djavan, João Sarubbi, Martin, Fabíola, Gustavo Ice-Man, Cristiano Arbex, Tais Marina, Leonardo Conegundes, Flávio Cruzeiro e tantos outros. Fora do DCC, tive a oportunidade de trabalhar e fazer muitos amigos na ATAN Automation: Vinícius Paiva, Juliano Simões, Marcelo Szuster, Marconi Arruda, Rafael Alexandre, Raphael Petronilho, Giovanni Maia, Kênia Carolina. Ainda preservo excelentes amigos do tempo do Programa de Residência Google: Ivan Conti, Anísio, Cássia, Iuri Chaer, Helen, David Reis. *Last but not least* agradeço aos amigos do Google Brasil: Davi Reis, Francisco Matos, Artur, Débora, Davi++, Han-Wen e Deca por todo o apoio nesses últimos 9 meses.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Otimização Combinatória é um dos ramos da Pesquisa Operacional que se dedica ao estudo de problemas que estão entre os mais difíceis da Ciência da Computação. Para uma grande variedade de problemas desta área, até o momento nenhum algoritmo eficiente (isto é, polinomial) foi encontrado. Assim sendo, para se resolver os problemas mais difíceis nesta classe, o melhor algoritmo pode requerer um número de operações elementares da mesma ordem que o número de operações requeridas por um algoritmo puramente enumerativo.

Problemas de Otimização Combinatória, embora de fundamental importância em Teoria da Computação, modelam também diversos problemas práticos. Por exemplo, o planejamento e dimensionamento de redes de telecomunicações, a alocação de recursos (pessoas, máquinas e insumos) em empresas, dentre outros problemas (veja Wolsey (1998) para uma descrição de alguns destes e outros problemas).

Graças ao aperfeiçoamento dos algoritmos e ao desenvolvimento de sistemas computacionais de maior porte, observamos nos últimos anos um maior interesse das empresas nas vantagens competitivas que podem ser obtidas através do uso do ferramental matemático dessa área. Notamos, por parte da mídia, uma maior divulgação de notícias que relatam como empresas melhoram os seus negócios a partir do uso das técnicas de otimização; por exemplo, reportagem do Portal Exame (EXAME (2006)) destaca o uso de técnicas de otimização em empresas nacionais. Já Nemhauser (2007) apresentou recentemente como técnicas de Otimização Combinatória foram usadas para resolver um complexo problema de alocação de aeronaves e tripulação para um seleto mercado nos Estados Unidos da América. Uma outra maneira de constatarmos o sucesso da área é através dos *cases* das empresas de otimização. Por exemplo, é fácil reconhecer grandes grupos corporativos mundiais, muitos deles líderes de mercado em suas respectivas áreas de atuação, dentre a lista de clientes da ILOG (ILOG (2008)), empresa de destaque no setor de sistemas computacionais para otimização. Finalmente, uma coleção de centenas de exemplos de casos de sucesso decorrentes da aplicação de técnicas de otimização é apresentada em INFORMS (2008).

Via de regra, os problemas da área de Otimização Combinatória podem ser formulados através de um grafo. Nesta dissertação, mais especificamente, discutiremos algoritmos para uma classe especial de grafos, que são as **árvores**, grafos conexos e acíclicos que possuem

importância fundamental em Ciência da Computação, devido às suas diversas aplicações (Ahuja et al. (1993); Kruskal (1956); Prim (1957)) e ao fato de possuírem relacionamento estreito com diversos problemas importantes, por exemplo o *Problema do Caixeiro Viajante* (Dantzig et al. (1954); Held e Karp (1970)). As árvores são ainda mais interessantes e instigantes pois, em muitos casos, elas estão na fronteira entre o que é fácil e o que é difícil de ser resolvido, do ponto de vista da Teoria da Computação.

O problema que tratamos nesta dissertação é conhecido como *Problema da Árvore de Custo Mínimo com k arestas*, ou simplesmente, k-ACM, que teve suas principais propriedades estudadas por Fischetti et al. (1994). De uma maneira informal, o k-ACM consiste em obter, a partir de um grafo valorado nos vértices e nas arestas, uma árvore de custo mínimo que contenha exatamente k arestas (e portanto, $k + 1$ vértices). Este problema foi demonstrado ser NP-Difícil, o que significa que é muito pouco provável que existam algoritmos eficientes (polinomiais) para resolvê-lo. A despeito de sua dificuldade, o problema já foi amplamente estudado na literatura, principalmente devido às suas aplicações no contexto industrial. Por exemplo, o primeiro trabalho a estudar o k-ACM (Hamacher e Joernsten (1993)) discute exatamente a sua aplicação mais conhecida, que está relacionada ao arrendamento de campos de petróleo de um país no norte da Europa. Além disso, há aplicações nas áreas de telecomunicações, planejamento de facilidades, dentre outras. Apesar de sua importância prática, até o presente momento, poucos trabalhos exploraram algoritmos exatos para a resolução deste problema, sendo esta umas das principais motivações desta dissertação.

1.1 Organização da dissertação

O restante do texto está organizado da seguinte forma: no Capítulo 2 formalizamos a definição do k-ACM, discutimos sua complexidade computacional e apresentamos uma revisão bibliográfica, destacando também suas aplicações mais conhecidas.

No Capítulo 3 apresentamos reformulações de Programação Inteira para o k-ACM. Resultados computacionais com um algoritmo exato do tipo *Branch-and-bound* que emprega estas reformulações são também apresentados. Estes resultados estimularam o desenvolvimento de uma Relaxação Lagrangeana para o k-ACM, que é tratada no Capítulo 4. Os algoritmos e reformulações apresentados nos Capítulos 3 e 4 constituem as nossas principais contribuições para a solução do k-ACM com garantia de otimalidade.

Finalmente, encerramos o texto no Capítulo 5, apresentando as principais conclusões desta dissertação e os trabalhos futuros que pretendemos conduzir.

Capítulo 2

O Problema da Árvore de Custo Mínimo com k arestas

Neste capítulo definimos o *Problema da Árvore de Custo Mínimo com k Arestas*, ou abreviadamente k-ACM, e discutimos os seus principais aspectos de complexidade computacional. Além disso, destacamos as variações do problema comumente encontradas na literatura e suas aplicações em diferentes contextos. Também fornecemos uma revisão dos principais trabalhos que previamente trataram do problema, com o objetivo de fornecer ao leitor uma visão do estado-da-arte das técnicas usadas para resolver o k-ACM. Concluímos este capítulo destacando a motivação principal desta dissertação.

2.1 O k-ACM e suas propriedades

Dado um grafo não-direcionado $G = (V, E)$, com um conjunto de vértices V $(n = |V|)$ e um conjunto de arestas E $(m = |E|)$, custos $\{d_v \geq 0 : v \in V\}$ associados aos vértices e custos $\{c_e \geq 0 : e \in E\}$ associados às suas arestas, o custo de uma árvore $T = (V_T, E_T)$ com exatas k arestas $(|E_T| = k)$ é dado por $\sum_{v \in V_T} d_v + \sum_{e \in E_T} c_e$. No k-ACM, desejamos encontrar uma árvore de G com k arestas, de mínimo custo.

Figura 2.1: Exemplo para o k-ACM – Grafo de entrada e uma árvore com $k = 4$ arestas.

Na literatura, diferentes versões do k-ACM são propostas, em geral diretamente relacionadas ao tipo da aplicação que está sendo estudada. Por exemplo, diversos trabalhos tratam da versão k-ACM-CA, onde apenas o custo das arestas é considerado (veja Ehrgott et al. (1997);

Blum e Ehrgott (2003); Jornsten e Lokketangen (1997)). Esta versão é mais adequada para modelar problemas onde o custo de instalação de uma aresta é mais significativo do que o custo de instalação de uma facilidade no vértice. Em contrapartida, outros trabalhos, como Brimberg et al. (2006), consideram a versão conhecida como k-ACM-PV, na qual apenas o custo dos vértices é considerado. Nesse caso, modela-se um problema no qual o custo de instalação de uma facilidade no vértice é muito maior do que o custo associado à aresta. Neste trabalho, entretanto, trataremos o caso geral onde custos são respectivamente associados às arestas e aos vértices de G.

2.1.1 Complexidade computacional

Independentemente da versão considerada, o k-ACM é em geral um problema de difícil resolução, sendo que diferentes trabalhos (por exemplo, Fischetti et al. (1994); Coimbra (1995)) mostraram que, dado um grafo arbitrário $G = (V, E)$ e um escalar $k \in N$, a versão de decisão do k-ACM é NP-Completo.

Para demonstrar que um problema é NP-Completo, em geral usa-se o conceito de *transformação polinomial* (ver T. H. Cormen (2001), Ziviani (2004)). Dados dois problemas Π_1 e Π_2, suponha que exista um algoritmo α_2 para resolver o problema Π_2. Se for possível transformar Π_1 em Π_2 e se for possível transformar a solução de Π_2 em uma solução para Π_1, então o algoritmo α_2 pode ser utilizado para resolver Π_1. Além disso, se as duas transformações puderem ser realizadas em tempo polinomial (com relação ao tamanho do problema), então Π_1 é polinomialmente transformável em (ou redutível a) Π_2.

No caso particular do k-ACM, sua NP-Completude foi demonstrada por Fischetti et al. (1994) e Coimbra (1995) através de uma transformação polinomial do Problema de Steiner em Grafos (Maculan (1987)) para o k-ACM. Dado que é possível verificar em tempo polinomial se um determinado sub-grafo $G' = (V', E')$, $V' \subseteq V$ e $E' \subseteq E$, induz uma árvore de k arestas em G, podemos dizer que a versão de otimização do k-ACM é, portanto, NP-Difícil.

Apesar de sua dificuldade intrínseca, alguns casos particulares do problema são polinomialmente solúveis. Quando $k = 1$, o problema pode ser resolvido em tempo polinomial, com exatas $m - 1$ operações de comparação. Por outro lado, se $k = n - 1$, o k-ACM é equivalente ao problema da Árvore Geradora de Custo Mínimo (AGM), podendo ser facilmente resolvido em $O(m \log n)$ pelos conhecidos algoritmos de Kruskal (1956) e Prim (1957).

Outro caso de fácil resolução bastaste interessante ocorre quando o grafo de entrada do problema é uma árvore, conforme foi provado por Fischetti et al. (1994) e Blum (2007b). Para esta classe específica de grafos, o problema pode ser resolvido utilizando-se a técnica de Programação Dinâmica. Fischetti et al. (1994) propuseram o algoritmo para a versão k-ACM-CA, que foi estendido por Blum (2007b) para o problema geral, com custos nas arestas e nos vértices, resultando em um algoritmo de complexidade de tempo igual a $O(k^2 n)$. Este algoritmo, denominado nesta dissertação por `DynamicTree`, será explicado em detalhes no Apêndice A, pois foi utilizado para geração de limites superiores em uma Heurística Lagrangeana que apresentamos para o k-ACM.

Devemos observar que a estratégia gulosa de se aplicar n vezes as k primeiras iterações do

algoritmo de Prim, tomando a cada execução um vértice do grafo como raiz, não garante a otimalidade nem mesmo quando o grafo de entrada é uma árvore. Como exemplo, considere o grafo G para o 4-ACM-CA dado pela árvore indicada na Figura 2.2, onde os custos das arestas são apresentados na figura. Observe que a sub-árvore com exatas 4 arestas de mínimo custo em G possui custo igual a 2 e que a melhor solução encontrada ao se executar o algoritmo de Prim n vezes possui custo 3.

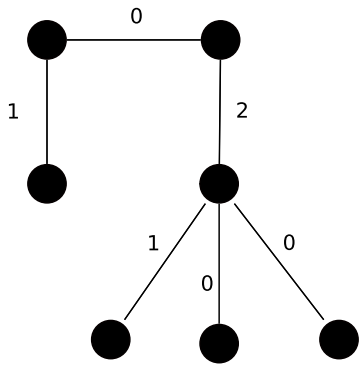

Figura 2.2: Exemplo para o qual a aplicação por repetidas vezes do algoritmo de Prim falha ao encontrar uma sub-árvore ótima para o 4-ACM-CA.

2.2 Aplicações do k-ACM

Uma das aplicações do k-ACM mais conhecidas na literatura consiste no *Problema de Arrendamento de Poços de Petróleo*. Em alguns países, por exemplo, na Noruega (Hamacher e Joernsten (1993)), o governo cede a uma empresa o direito de exploração de campos de petróleo, por um determinado período de tempo (por exemplo, 5 anos). Ao final deste período de tempo, a empresa arrendatária deve devolver alguns campos petrolíferos da região, enquanto outros ela ainda poderá explorar. Tipicamente, as regras contratuais impõem que um percentual do número de campos (por exemplo, 25% deles) deverá ser devolvido, sob a condição de que exista um caminho entre quaisquer pares de campos i e j, $i \neq j$. A Figura 2.3 ilustra o modelo em um grafo associado ao problema. Neste modelo, cada vértice do grafo representa um campo de petróleo da região de exploração e as arestas representam possíveis equipamentos de transporte/conexão entre eles. A empresa dispõe do tempo de exploração para avaliar uma estimativa do valor d_v associado à reserva residual em cada campo $v \in V$, ao final do período contratual. Assim sendo, a empresa eventualmente pode escolher os k campos conectados de menor valor comercial para serem devolvidos ao governo, caso deseje maximizar os seus lucros. Neste cenário, é bem provável que o valor econômico dos campos seja muito superior ao custo dos equipamentos empregados. Portanto, é razoável admitir que $\{c_e = 0, \forall e \in E\}$, e assim sendo esta aplicação é melhor representada pela versão k-ACM-PV.

A lista de outros problemas práticos que podem ser modelados através do k-ACM é bastante extensa. Bruglieri et al. (2006) destacaram as seguintes aplicações em:

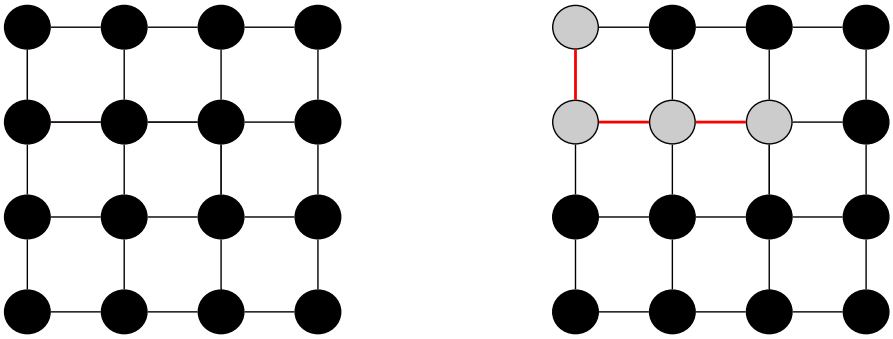

Figura 2.3: Aplicação do k-ACM no Problema de Arrendamento de Campos de Petróleo. À esquerda, um grafo com os vértices representando os campos de petróleo e as arestas representando os equipamentos de conexão entre eles. À direita, em cinza, os vértices que representam os campos a serem devolvidos para o governo após o término do período de arrendamento.

- *Telecomunicações* (Garg e Hochbaum (1997)): Considera-se o problema de uma empresa de telecomunicações que obteve uma concessão governamental para explorar quaisquer k cidades em um estado de n cidades. Nesta aplicação, os vértices do grafo representam as cidades, cada uma com um potencial econômico d_v, e as arestas representam os cabos do sistema telefônico, que possuem o custo de implantação c_e, proporcional à distância entre as cidades que conectam. Eventualmente, a empresa está interessada em minimizar o custo de implantação do parque telefônico ao passo que deseja obter maior lucratividade, escolhendo as k cidades com potencial econômico mais significativo.

- *Roteamento* (Cheung e Kumar (1994)): Em redes de computadores, é muito comum a utilização de uma técnica de roteamento chamada *multicasting*, cujo objetivo é enviar uma mensagem para um sub-conjunto de p vértices previamente definidos de uma rede de n vértices. A generalização desta idéia dá origem à uma técnica conhecida como *quorumcasting*, que consiste em enviar uma mensagem para *qualquer* sub-conjunto de k vértices dentre os p vértices pré-definidos. Nesse cenário, os vértices do grafo são os dispositivos computacionais que fazem parte da rede e as arestas modelam medidas de Qualidade de Serviço (QoS – *Quality of Service*), tais como distância, atraso, latência, dentre outros. O objetivo consiste em construir uma rede de comunicação entre qualquer sub-conjunto com k vértices que maximize a QoS. A estratégia de *quorumcasting* é muito usada em diversas aplicações em sistemas distribuídos, tais como sincronização ou atualização de recursos replicados.

- *Mineração*(Philpott e Wormald (1997)): Os autores consideram o problema de realizar a mineração de uma lavra de ouro a céu aberto, que pode ser modelado através do k-ACM com restrições adicionais. Nesse problema, os vértices são os blocos de solo, cada qual com seu respectivo valor econômico d_v, proporcional à quantidade de ouro existente no bloco. As arestas, por sua vez, são usadas para mapear as relações de precedência entre os blocos; um bloco u só pode ser extraído se todos os blocos que o precedem já

tiverem sido extraídos. Neste problema, deseja-se extrair uma quantidade k de blocos, de modo a satisfazer a demanda imposta pelos acordos comerciais da mineradora.

Além das aplicações acima apresentadas, existem outras menos conhecidas, por exemplo o projeto do *layout* de prédios de escritórios discutido por Fischetti et al. (1994).

2.3 Trabalhos relacionados e técnicas previamente empregadas para solução do k-ACM

O k-ACM foi introduzido por Fischetti et al. (1994), quando além de uma formulação de Programação Inteira baseada nas Desigualdades de Eliminação de Sub-rotas Generalizadas (Lucena (1991); Goemans (1994); Margot et al. (1994)), foram apresentados um estudo poliedral, um estudo de complexidade computacional e um esboço de um algoritmo *Branch-and-cut* para o problema.

O trabalho de Coimbra (1995) discute a versão roteada do k-ACM, na qual um vértice $r \in V$ previamente definido deve fazer parte da solução. Este trabalho apresenta duas formulações e algumas desigualdades válidas para o problema.

Dentre os pacotes de *software* dedicados ao k-ACM, destacamos aquele desenvolvido por Ehrgott e Freitag (1996). Este pacote inclui heurísticas baseadas em adaptações do algoritmo de Prim (Prim (1957)) e em Programação Dinâmica, bem como uma implementação do algoritmo *Branch-and-cut* sugerido por Fischetti et al. (1994). Entretanto, a referência não menciona aspetos cruciais de um algoritmo *Branch-and-cut*, tais como os procedimentos de separação, as políticas de *branching*, de seleção de nós, etc.

Já Ehrgott et al. (1997) apresentam duas diferentes heurísticas para a versão k-ACM-CA. Na primeira delas, executa-se o algoritmo de Prim n vezes, cada qual tendo um vértice de V como raiz. Em cada execução, as k primeiras iterações do método de Prim são realizadas. A segunda heurística, por sua vez, pode ser vista como uma adaptação do algoritmo de Borůvka (Borůvka (1926a,b)) para o problema. Em cada passo do algoritmo, remove-se a aresta mais cara do grafo, desde que as arestas remanescentes contenham pelo menos um componente conexo com $k+1$ ou mais vértices. A remoção de arestas prossegue até que o grafo remanescente induza uma árvore com k arestas em G.

Ehrgott et al. (1997) também apresentam limites inferiores para o k-ACM-CA, dados pela soma dos custos das k primeiras arestas selecionadas pelo algoritmo de Kruskal. Este limite inferior foi utilizado em conjunto com as duas heurísticas acima explicadas, em testes computacionais com grafos de até 30 vértices. A partir desta experiência, os autores concluíram que problemas definidos em grafos com mais de 40 vértices seriam de difícil solução exata.

Jornsten e Lokketangen (1997), por sua vez, propuseram um algoritmo de Busca Tabu (*Tabu Search*) para o k-ACM, que foi testado com grafos de 100 vértices e $k = 50$.

Já Kataoka et al. (2000) discutem a versão roteada do k-ACM-CA, mostrando que o problema de decisão associado a esta versão do k-ACM também é NP-Completo. Além disso, os autores propuseram diferentes métodos para obtenção de limites inferiores e superiores para

o problema. Um dos procedimentos propostos para a obtenção de limites inferiores baseia-se em uma modificação do algoritmo de Kruskal. Inicialmente, computa-se para cada vértice v o número de arestas no caminho mínimo entre v e o vértice definido como raiz. A fase seguinte é similar à fase iterativa do algoritmo de Kruskal, com a exceção de que na i-ésima iteração, somente os vértices que estejam a no máximo i arestas da raiz são candidatos a entrar na floresta em construção. Os autores mostram que esta estratégia gera um limite inferior válido para o k-ACM.

Kataoka et al. (2000) também apresentaram uma heurística baseada no algoritmo de Prim seguido de uma busca local. O algoritmo de Prim é utilizado para gerar uma solução viável $T = (V_T, E_T)$ para o grafo de entrada $G = (V, T)$. Os movimentos da busca local são dados por trocas de vértices em $V_T \setminus \{r\}$ por outros em $V \setminus V_T$. As trocas são realizadas até que nenhum par (v', v''), $v' \in V_T \setminus \{r\}$, $v'' \in V \setminus V_T$, melhore o custo da solução corrente.

Os experimentos computacionais realizados por Kataoka et al. (2000), envolvendo grafos *grid* com até 1000 vértices, permitiram estabelecer empiricamente que as instâncias mais difíceis do problema possuem valores de k em torno de $\frac{n}{3}$. Para a versão não-roteada do k-ACM, entretanto, outros trabalhos, por exemplo Blum (2007b), observaram que as instâncias mais difíceis possuem $k \approx \frac{n}{2}$.

Blum e Ehrgott (2003) também propuseram métodos de solução aproximada para k-ACM-CA, baseados em Busca Local, em Algoritmos Genéticos (Holland (1992); Goldberg (1989)) paralelos e em Busca Tabu (Glover e Laguna (1997)).

Já Blum e Blesa (2005) apresentam meta-heurísticas para o k-ACM: uma Busca Tabu, um algoritmo de Computação Evolucionária e um algoritmo de Otimização por Colônia de Formigas (*Ant Colony Optimization*). Os autores também consolidaram as instâncias usadas nos testes em uma biblioteca de instâncias, conhecida como KCTLib (ver Blum (2007a))[1]. Desde então, contribuições de diferentes autores fizeram da KCTLib o principal repositório *on-line* de informações sobre o k-ACM, onde, atualmente encontra-se mais de uma centena de instâncias com graus variados de dificuldade.

Brimberg et al. (2006) apresentam uma meta-heurística VNS (*Variable Neighborhood Search* – veja Hansen e Mladenovic (2001)) desenvolvida exclusivamente para o k-ACM-PV. Na referência, argumenta-se não ser possível resolver instâncias com mais de 20 vértices na otimalidade através de algoritmos *Branch-and-bound*, justificando assim a escolha da técnica VNS. São apresentados resultados computacionais para instâncias *grid* com número de vértices variando entre 900 e 2500. Mostra-se que os resultados do VNS superam, em termos da qualidade das soluções encontradas e tempo computacional, aqueles obtidos pelos algoritmos de Blum e Ehrgott (2003) discutidos anteriormente.

De fundamental importância para o trabalho desenvolvido no Capítulo 4 desta dissertação é o algoritmo de Programação Dinâmica apresentado por Blum (2007b), que é uma extensão do algoritmo proposto por Fischetti et al. (1994). O algoritmo original se aplicava apenas à versão k-ACM-CA e extraía uma k-ACM a partir de uma árvore de entrada. A versão de Blum (2007b), por sua vez, pode ser usada para grafos valorados nos vértices e nas arestas.

[1] `http://iridia.ulb.ac.be/~cblum/kctlib/`

Blum (2007b) também realizou uma série de experimentos computacionais, incluindo grafos *grid* e instâncias da KCTLib, visando comparar o seu algoritmo com os melhores algoritmos disponíveis na literatura até então.

Na linha de algoritmos aproximados, destacam-se os trabalhos de Garg (1996), que apresenta um algoritmo com fator de aproximação igual a 3, e, mais recentemente, o trabalho de Arora e Karakostas (2006), que apresenta um algoritmo de custo $O(n^{1/\epsilon})$ com fator de aproximação igual a $2 + \epsilon$.

Como pode ser observado, apenas um trabalho apresentou, até o momento, algoritmos exatos para a resolução do problema aqui tratado (Ehrgott e Freitag (1996)). Talvez por isto, em alguns trabalhos, identificamos observações bastante céticas quanto as dimensões das instâncias que podem ser resolvidas com garantia de otimalidade em *tempos de computação razoáveis* (ver por exemplo, Ehrgott et al. (1997) e Brimberg et al. (2006)). Esta observação nos motivou a propor novas formulações e algoritmos para solução exata do k-ACM. Estes temas são discutidos nos capítulos que seguem.

Capítulo 3

Reformulações para o k-ACM

Neste capítulo, em adição à formulação de Programação Inteira baseada em *Desigualdades de Eliminação de sub-rotas Generalizadas* (*GSEC*[1]), apresentamos duas reformulações para o k-ACM. Ambas baseiam-se em uma transformação do grafo de entrada e, diferentemente da formulação que emprega as GSEC, possuem número polinomial de variáveis e restrições. Ao final do capítulo, apresentamos resultados computacionais obtidos com um algoritmo do tipo *Branch-and-bound* (B&B), no qual limites duais são obtidos através de Relaxações Lineares das reformulações propostas.

3.1 A Formulação baseada em Desigualdades de Eliminação de Sub-rotas Generalizadas

Antes de apresentarmos a formulação baseada em GSEC, vamos introduzir a notação utilizada neste e nos próximos capítulos. Dado um grafo $G = (V, E)$, denotamos uma aresta $e \in E$ conectando i e j por (i, j). Dado um conjunto de vértices S, denotamos por $E(S) := \{(i, j) \in E : i, j \in S\}$ o conjunto de arestas com ambas as extremidades em S. Por outro lado, dado um digrafo (ou seja, um grafo direcionado) $D = (V, A)$, vamos denotar por $[i, j]$ um arco $a \in A$ que parte de i e chega a j. De maneira similar ao caso não direcionado, $A(S) := \{[i, j] \in A : i, j \in S\}$ é o conjunto de arcos com ambas as extremidades em S. Vamos também definir $\delta^+(S) := \{[i, j] \in A : i \in S, j \notin S\}$ e $\delta^-(S) := \{[i, j] \in A : i \notin S, j \in S\}$, respectivamente, como os conjuntos de arcos que partem e que chegam em S. Para os casos em que S possui apenas um vértice i, usaremos a notação $\delta^+(i)$ e $\delta^-(i)$ em substituição a $\delta^+(\{i\})$ e $\delta^-(\{i\})$.

Fischetti et al. (1994) apresentaram a primeira formulação de Programação Inteira para o k-ACM, que emprega as desigualdades de eliminação de sub-rotas generalizadas (GSEC), propostas independentemente por Lucena (1991), Goemans (1994) e Margot et al. (1994) no estudo do Problema de Steiner em Grafos.

Em sua formulação, Fischetti et al. (1994) empregam dois conjuntos de variáveis de decisão binárias:

[1] *Generalized Subtour Elimination Constraints.*

$$x_e = \begin{cases} 1, \text{ se aresta } e \text{ pertence à solução ótima} \\ 0, \text{ caso contrário} \end{cases}$$

e

$$y_i = \begin{cases} 1, \text{ se o vértice } i \text{ pertence à solução ótima} \\ 0, \text{ caso contrário.} \end{cases}$$

As variáveis acima, aliadas aos parâmetros c_e e d_i definidos na Seção 2.1, são utilizadas na seguinte formulação:

$$z = \min \left\{ \sum_{e \in E} c_e x_e + \sum_{i \in V} d_i y_i : (x, y) \in P_{GSEC} \cap (R^{|E|}, B^{|V|}) \right\} \tag{3.1}$$

onde a região poliédrica P_{GSEC} é dada por:

$$\sum_{e \in E} x_e = k, \tag{3.2}$$

$$\sum_{i \in V} y_i = k + 1, \tag{3.3}$$

$$\sum_{e \in E(S)} x_e \leq \sum_{i \in (S \setminus \{j\})} y_i, \ \forall j \in S, \forall S \subseteq V, |S| \geq 2, \tag{3.4}$$

$$0 \leq x_e \leq 1, \ \forall e \in E, \tag{3.5}$$

$$0 \leq y_i \leq 1, \ \forall i \in V. \tag{3.6}$$

As restrições (3.2) e (3.3) são restrições que garantem, respectivamente, que a solução deverá envolver k arestas e $k + 1$ vértices. Observe que este par de restrições é redundante – podemos seguramente remover uma delas. As GSEC (3.4), por sua vez, garantem que a solução é livre de ciclos. Observe que se $y_i = 1, \forall i \in S$, as GSEC se reduzem às Desigualdades de Eliminação de Sub-rotas[2]:

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \forall S \subseteq V, S \neq \emptyset, \tag{3.7}$$

propostas originalmente por Dantzig et al. (1954).

Esta formulação possui $m + n$ variáveis de decisão e exponencialmente muitas restrições (3.4). Conforme mencionamos anteriormente, o único trabalho que explorou a formulação (3.1) foi Ehrgott e Freitag (1996), através de um algoritmo *Branch-and-cut*, onde (3.4) são empregadas como planos de corte. Entretanto, não são disponíveis na literatura resultados computacionais deste algoritmo.

[2]SEC: *Subtour Elimination Constraints.*

Ao contrário da formulação proposta por Fischetti et al. (1994), nesta dissertação optamos por trabalhar com formulações compactas, isto é, formulações que possuem um número polinomial de variáveis e restrições. As duas formulações que propusemos são baseadas em uma re-interpretação do problema, decorrente de uma transformação do grafo de entrada, que será apresentada a seguir.

3.2 Reformulações para o k-ACM baseadas em uma transformação do grafo de entrada

O k-ACM é definido originalmente em termos de um grafo não direcionado $G = (V, E)$. Entretanto, as reformulações que propomos fazem uso de um digrafo $D = (\overline{V}, A)$, obtido a partir de $G = (V, E)$ através da introdução de:

- (passo 0) Dois vértices artificiais, $n + 1$ e $n + 2$;
- (passo 1) Um arco artificial $[n + 1, n + 2]$, com $c_{[n+1,n+2]} = 0$;
- (passo 2) Arcos artificiais $\{[n + 1, i] : i \in V\}$ com custos $c_{[n+1,i]} = 0$;
- (passo 3) Arcos artificiais $\{[n + 2, i] : i \in V\}$ com custos: $c_{[n+2,i]} = d_i$;
- (passo 4) Um conjunto de pares de arcos $A_E = \{[i, j], [j, i] : (i, j) \in E\}$ representando o conjunto de arcos obtidos pela duplicação das arestas de E, cujos custos são:

$$c_{[i,j]} = c_{(i,j)} + d_j \tag{3.8}$$

e

$$c_{[j,i]} = c_{(i,j)} + d_i. \tag{3.9}$$

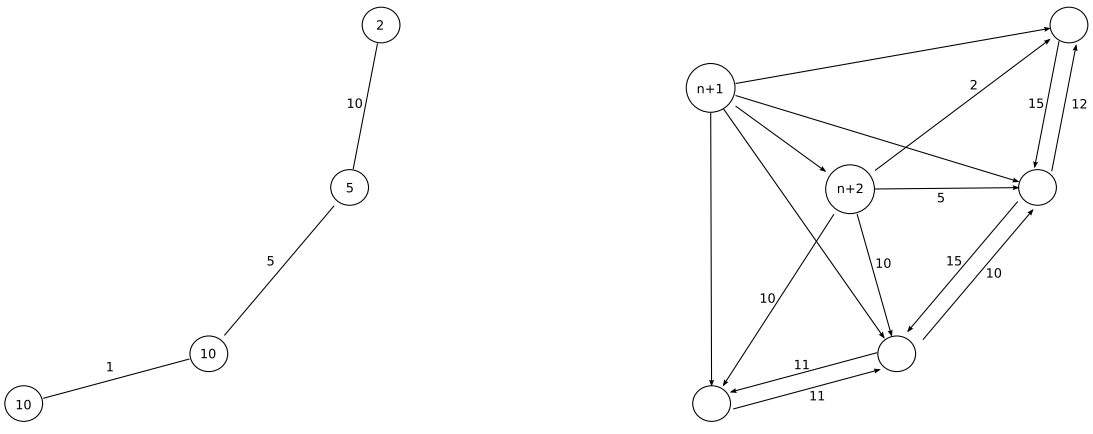

Figura 3.1: Exemplo de transformação para o k-ACM – Grafo de original e o digrafo correspondente.

A Figura 3.1 apresenta uma ilustração da transformação proposta. Observe que incorporamos os custos dos vértices aos custos dos arcos em A_E que incidem a vértices em V. Desta forma, o digrafo D emprega os seguintes conjuntos:

- $V' = V \cup \{n+2\}$
- $\overline{V} = V' \cup \{n+1\}$
- $A = \{[n+1, n+2]\} \cup \{[n+1, i] : i \in V\} \cup \{[n+2, i] : i \in V\} \cup A_E$. Observe que os arcos em A possuem custo c_a, calculados de acordo com os passos 1, 2, 3 e 4 da transformação acima.

Pode-se observar que é possível mapear toda árvore $T = (V_T, E_T)$, viável para o k-ACM em G, em uma arborescência $T_D = (\overline{V}, A_D)$[3], roteada em $n+1$, que satisfaz algumas restrições adicionais (veja a Figura 3.2):

- os vértices $i \notin V_T$ encontram-se conectados ao vértice $n+1$, por meio de arcos do tipo $[n+1, i]$;
- para todo vértice $i \in V_T$, existe um único caminho (direcionado) entre $n+1$ e i no sub-grafo de D induzido por T_D, que necessariamente envolve o arco $[n+1, n+2]$;
- dentre os vértices em V_T, apenas um deles, digamos r, se conecta diretamente a $n+2$, por meio de $[n+2, r]$. Se este arco for removido de T_D, passamos a ter dois sub-grafos não conectados. O primeiro contém os vértices $\{n+1, n+2\}$ e $\{j \in V : j \notin V_T\}$. O segundo consiste em uma arborescência roteada em r, que, se desconsiderada a orientação dos arcos, induz uma solução viável para o k-ACM.

Observe também que, pelo modo como definimos os custos dos arcos (passos 1, 2, 3 e 4 da transformação), o custo da aborescência $T_D = (V_D, A_D)$, roteada em $n+1$, equivale exatamente ao custo de uma k-ACM em G.

Assim sendo, podemos apresentar reformulações de Programação Inteira para o k-ACM, agora definido sobre o digrafo D, nas quais a conectividade das soluções viáveis pode ser assegurada por meio de:

- (exponencialmente muitas) Restrições denominadas de *directed cut* (Chopra et al. (1992); Koch e Martin (1998)). Estas restrições impõem que, em qualquer arborescência viável, pelo menos um arco deve incidir a qualquer sub-conjunto de vértices $S \subset \overline{V}$, $n+1 \notin S$.
- Restrições de salto, que empregam variáveis para mapear o número de arcos no caminho entre dois vértice i e j. Devemos observar que uma solução viável para o k-ACM em D é uma arborescência na qual o número de arcos (saltos) entre $n+1$ e i deve ser não

[3]Uma arborescência é um grafo direcionado no qual, para um vértice raiz r e qualquer outro vértice v, existe exatamente um caminho direcionado entre r e u. Em outras palavras, uma arborescência é um árvore direcionada e roteada, na qual todas as arestas apontam para a direção aposta à raiz. Observe, portanto, que toda arborescência é um grafo acíclico direcionado.

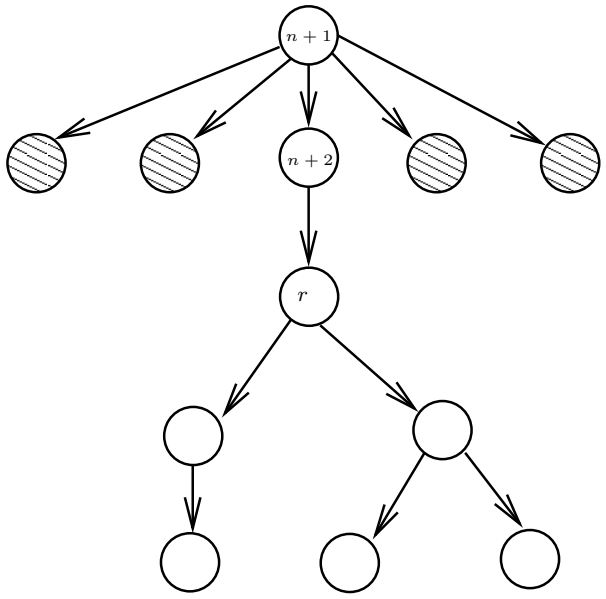

Figura 3.2: Um exemplo de solução viável em D.

superior a $k+2$. Além disso, quando $i \notin V_T$, o número de arcos entre $n+1$ e i deve ser exatamente 1, caso em que o arco $[n+1, i]$ é utilizado. O mesmo ocorre quando $i = n+2$, onde se utiliza o arco $[n+1, n+2]$.

- Restrições de balanço de fluxo, por exemplo em formulações de fluxo que empregam uma única ou múltiplas mercadorias, como aquelas propostas por Maculan (1987) para o Problema de Steiner em Grafos.

- Restrições de nível, que empregam variáveis de nível em adição àquelas associadas à seleção de arcos (ver, por exemplo, Miller et al. (1960)).

Nesta dissertação, desenvolvemos métodos de solução para o k-ACM que empregam reformulações baseadas em restrições de balanço de fluxo e de nível, que serão apresentadas nas seções a seguir.

3.2.1 Uma Reformulação Multi-Fluxo para o k-ACM

A idéia central desta reformulação é a seguinte: para cada vértice $q \in V'$, criamos uma demanda unitária de uma mercadoria q e uma oferta unitária da mesma mercadoria em $n+1$. Assim sendo, cada mercadoria q deve ser entregue ao seu respectivo vértice de destino através de um caminho que se inicia em $n+1$ e termina em q. A união de todos os caminhos necessários para satisfazer as demandas criadas deve induzir uma arborescência viável para o k-ACM em D.

Nesta abordagem, o problema de se determinar uma árvore com k arestas de mínimo custo é transformado em um *Problema de Fluxo de Custo Mínimo* com múltiplas mercado-

rias (Ahuja et al. (1993)), sujeito a um conjunto de restrições complicantes (envolvendo a cardinalidade dos conjuntos de arcos e vértices selecionados em D).

Para estabelecermos esta reformulação, definimos dois conjuntos de variáveis de decisão:

1. Variáveis binárias $\{x_a : a \in A\}$, sendo que x_a assume valor 1 caso o arco a seja empregado para transportar uma unidade de mercadoria na rede;
2. Variáveis de fluxo reais f_a^q positivas que denotam a quantidade de fluxo da mercadoria destinada a q transportada pelo arco a.

Uma formulação Multi-Fluxo para o k-ACM é então dada por:

$$z = \min \left\{ \sum_{a \in A} c_a x_a : (x, f) \in P_{MF} \cap \left(B^{2(m+n)+1}, R_+^{n(2(m+1)+n)+1} \right) \right\} \tag{3.10}$$

onde a região poliédrica P_{MF} é dada por:

$$f^q_{[n+1,q]} + f^q_{[n+1,n+2]} = 1, \ \ \forall q \in V', \tag{3.11}$$

$$\sum_{a \in \delta^+(i)} f_a^q - \sum_{a \in \delta^-(i)} f_a^q = 0, \ \ \forall q, i \in V', \ \ i \neq q, \tag{3.12}$$

$$\sum_{a \in \delta^+(q)} f_a^q - \sum_{a \in \delta^-(q)} f_a^q = -1, \ \ \forall q \in V', \tag{3.13}$$

$$\sum_{q \in V'} f^q_{[n+1,n+2]} = k + 2, \tag{3.14}$$

$$\sum_{a \in A_E} x_a = k, \tag{3.15}$$

$$\sum_{a \in \delta^+(n+2)} x_a = 1, \tag{3.16}$$

$$\sum_{a \in \delta^+(n+1)} x_a = n - k, \tag{3.17}$$

$$f_a^q \leq x_a, \ \ \forall q \in V, \ \ \forall a \in A. \tag{3.18}$$

As restrições de balanço de fluxo (3.11), (3.12) e (3.13) garantem que as demandas são atendidas. Observe que as restrições (3.11) garantem que uma unidade de fluxo $q, \forall q \in V'$, deve sair de $n + 1$; por sua vez, as restrições (3.12) garantem que um vértice $i \neq q$ não retém a mercadoria q e as restrições (3.13) garantem que a mercadoria q é retida pelo vértice destino correspondente. Observe qeue esta reformulação não associa variáveis de fluxo $f^q_{[n+1,i]}$, para o caso em que $i \neq q$ (ver Equação (3.11)), evitando assim que os arcos $[n + 1, i], [i, q]$

sejam empregados simultaneamente em uma arborescência. Desta maneira, eliminamos a necessidade de introduzir as restrições

$$x_{[n+1,i]} + x_{[i,j]} \leq 1, \forall [i,j] \in A_E,$$

nesta reformulação.

A restrição (3.14) garante que $k+2$ unidades de fluxo são enviadas a partir de $n+1$ para o vértice $n+2$: o vértice $n+2$ retém a unidade de fluxo da mercadoria a ele associada e encaminha as demais para os $k+1$ vértices de uma sub-árvore viável com k arestas. Por sua vez, a restrição (3.16) garante que somente um vértice $r \in V$ estará diretamente conectado a $n+2$.

Por outro lado, a restrição (3.17) garante que os $n-(k+1)$ vértices de V que não fazem parte da arborescência com k arcos roteada em r serão conectados diretamente ao vértice $n+1$, bem como o vértice $n+2$.

Finalmente, as restrições (3.18) garantem que só pode haver fluxo em um arco a que for selecionado para fazer parte da arborescência. Em conjunto com as restrições (3.11), (3.12), (3.13) e (3.17), estas restrições também asseguram que a solução é livre de ciclos.

3.2.2 Uma reformulação baseada em um *lifting* das desigualdades de Miller, Tucker e Zemlin

Nesta seção, apresentamos uma reformulação para o k-ACM onde a conectividade das soluções é garantida através do uso das restrições de eliminação de sub-circuitos propostas por Miller et al. (1960), no contexto do Problema do Caixeiro Viajante (PCV).

Para formular o PCV como um Problema Inteiro, Miller et al. (1960) empregaram variáveis de nível $u_v \in R$, que denotam o número de arcos entre o vértice 1 (depósito ou ponto de partida do caixeiro) e v. Observe que não mais de $n-1$ arcos são necessários em um caminho que inicia em 1 e termina em v. Por este motivo, a diferença de nível entre quaisquer dois vértices i e j é limitada superiormente por $n-2$. Se a rota do caixeiro for tal que j é visitado imediatamente após i, a diferença de nível entre i e j é -1.

Assim sendo, Miller et al. (1960) propuseram as seguintes restrições de eliminação de sub-circuitos:

$$u_i - u_j + (n-1)x_{[i,j]} \leq n-2, \;\; 2 \leq i,j \leq n. \tag{3.19}$$

Observe que se $x_{[i,j]} = 1$, então:

$$u_i - u_j \leq -1, \tag{3.20}$$

ou seja, o nível de i deve ser uma unidade menor que o nível de j em uma solução viável. No caso em que $x_{[i.j]} = 0$, a desigualdade é trivialmente satisfeita. Para verificar que (3.20) elimina sub-circuitos que não envolvem o vértice depósito 1, basta somar (3.20) para os arcos no sub-circuito e verificar que chegamos a uma contradição.

Desrochers e Laporte (1991) mostraram ser possível fortalecer (3.19) com o seguinte *lifting*:

$$u_i - u_j + (n-1)x_{[i,j]} + (n-3)x_{[j,i]} \leq n-2, \forall [i,j] \in A. \tag{3.21}$$

A reformulação que apresentamos nesta seção, denominada MTZ, utiliza uma adaptação de (3.21) para o caso do k-ACM. Para estabelecermos a reformulação MTZ, utilizamos as seguintes variáveis de decisão:

1. Variáveis binárias $\{x_a : a \in A\}$, sendo que x_a assume valor 1 caso o arco a seja parte de uma solução viável.
2. Variáveis de nível $u_v \in R$, conforme Miller et al. (1960). No caso do k-ACM, estas variáveis denotam o número de arcos no caminho que conecta a raiz $n+1$ a $v \in V'$ em uma arborescência. Isto significa que um vértice v diretamente conectado a $n+1$ tem nível 1, por exemplo $u_{n+2} = 1$, e, pela transformação apresentada, um vértice i diretamente conectado a $n+2$ tem nível $u_i = 2$. Além disso, por definição, $u_{n+1} = 0$.

A reformulação para o k-ACM é dada por:

$$\min \left\{ \sum_{a \in A} c_a x_a : (x,u) \in P_{MTZ} \cap (B^{2(m+n)+1}, R^n_+) \right\} \tag{3.22}$$

onde a região poliédrica P_{MTZ} é dada pela interseção de (3.15)-(3.17) e:

$$\sum_{a \in \delta^-(j)} x_a = 1, \forall j \in V', \tag{3.23}$$

$$x_{[n+1,i]} + x_{[i,j]} \leq 1, \forall [i,j] \in A_E, \tag{3.24}$$

$$(k+3)x_{[i,j]} + u_i - u_j + (k+1)x_{[j,i]} \leq (k+2), \forall [i,j] \in A, \tag{3.25}$$

$$(k+3)x_{[i,j]} + u_i - u_j \leq (k+2), \forall [i,j] \in A \setminus A_E, \tag{3.26}$$

$$x_{[n+1,n+2]} = 1, \tag{3.27}$$

$$u_{n+1} = 0. \tag{3.28}$$

A restrição de atribuição (3.23) garante que existe exatamente um arco chegando aos vértices do conjunto V'. Por sua vez, a restrição (3.24) garante que para um vértice $v \in V$, ou v está diretamente conectado a $n+1$ ou v pertence à arborescência com raiz r que induz uma solução viável para o k-ACM em G.

As restrições (3.25) e (3.26) são adaptações da restrição (3.21) para o k-ACM e garantem que a solução é livre de ciclos.

Finalmente, (3.27) e (3.28) indicam os domínios das variáveis e (3.28) inicializa o nível da raiz $n+1$ como zero.

Na próxima seção apresentamos os experimentos computacionais obtidos com um algoritmo *Branch-and-bound* que emprega as duas reformulações propostas.

3.3 Experimentos Computacionais

Nesta seção, investigamos as duas reformulações propostas para o k-ACM através de experimentos computacionais. Com estes experimentos, avaliamos empiricamente a qualidade do *gap* de Relaxação Linear de cada uma, assim como o tempo gasto para o cálculo deste limite. Além disso, consideramos o tempo total gasto pelo algoritmo para obtenção de soluções inteiras (e eventualmente, a solução ótima do problema). Discutimos o consumo de memória do algoritmo, e como este fator restringe o uso das reformulações. Comparamos também os resultados obtidos com aqueles disponíveis na literatura, principalmente no que diz respeito à qualidade das soluções inteiras encontradas.

A nossa base de testes é composta por 3 conjuntos de instâncias:

1. Conjunto `g`: proposto por Blesa e Xhafa (2000), contém 35 instâncias para o k-ACM-CA formadas por grafos 4-regulares (isto é, cada vértice possui grau 4), com número de vértices variando entre 25 e 1000 e $k = 20$. Nesta seção, entretanto, apresentamos resultados computacionais apenas para as 30 instâncias que possuem até 400 vértices, devido às limitações de tempo computacional e memória.

2. Conjunto `d`: proposto por Blum e Blesa (2005), este conjunto contém quatro grupos de instâncias. O primeiro deles inclui instâncias cujos nomes iniciam-se com o prefixo `bb`, correspondentes a grafos *grid* k-ACM-CA com $n \in [225, 1089]$ e diferentes valores de k. O segundo grupo é formado por algumas das maiores instâncias do conjunto `g`, todas com $k \neq 20$. As instâncias cujos nomes iniciam-se com o prefixo `steiner` são oriundas das instâncias para o *Problema de Steiner em Grafos* da *OR-Library* (Beasley (1990)) possuem $n \in [500, 1000]$ e custos nulos nos vértices. O quarto e último grupo consiste em apenas uma instância do tipo `Leighton`, proveniente dos conjuntos de testes do *Problema de Coloração de Grafos* (Beasley (1990)), que também possui apenas custo nas arestas.

3. Conjunto `NWG`: as instâncias deste conjunto foram geradas conforme sugerido no trabalho de Brimberg et al. (2006). Tratam-se de grafos *grid* com custos nulos nas arestas, com dimensões variando entre 10×10 a 20×20. Os vértices possuem custos inteiros, aleatoriamente escolhidos com distribuição de probabilidade uniforme no intervalo $[10, 1000]$. Para as instâncias deste conjunto, fixamos $k \approx \lfloor \frac{n}{2} \rfloor$, pois a nossa experiência computacional mostrou que este é um valor ao qual se associam instâncias bastante difíceis do k-ACM. Este fato também foi observado em outros estudos, como por exemplo em Blum (2007b).

Todos os experimentos foram executados em um computador Pentium Dual Processor XEON 3.0GHz 64 bits, 2 GB de memória principal e 2 MB de memória cache por processador, rodando o sistema operacional Debian GNU/Linux – 2.6.18-4-amd64. Utilizamos o pacote de otimização CPLEX (ILOG (2006)), versão 10.2.0, com parâmetros *default*.

3.3.1 Análise do limite de Programação Linear

As Tabelas 3.1, 3.2 e 3.3 e 3.4 apresentam, respectivamente, os resultados obtidos para cada um dos conjuntos de instâncias descritos anteriormente. As primeiras 4 colunas de cada uma das tabelas indicam o nome da instância e os valores de n, m e k. Nas próximas duas colunas, apresentamos o limite de Programação Linear para a reformulação Multi-Fluxo (LP_{MF}) e o tempo gasto (em segundos), para avaliá-lo. Resultados similares para a reformulação MTZ são apresentados nas colunas seguintes. Finalmente, a última coluna apresenta a razão entre LP_{MF} e LP_{MTZ}, o limite dual fornecido pela reformulação MTZ.

É possível observar que, no geral, os limites LP_{MF} são mais fortes do que aqueles dados por LP_{MTZ}. Considerando apenas as instâncias para as quais conseguimos avaliar LP_{MF}, a razão LP_{MF}/LP_{MTZ} foi de 1.70, 2.54 e 1.04, respectivamente, para os conjuntos `g`, `d` e `NWG`.

Para as instâncias pequenas (25 vértices) e moderadas (400 vértices) do conjunto `g`, o *gap* médio da reformulação Multi-fluxo foi de 0.55%, sendo que 30% das instâncias foram resolvidas na raiz da árvore de enumeração. Entretanto, para as instâncias de tamanho moderado, mesmo a avaliação da Relaxação Linear exige horas de processamento. Por outro lado, a reformulação MTZ apresentou *gaps* maiores, mas o tempo necessário para o cálculo da Relaxação Linear é praticamente desprezível.

Para diversas instâncias do conjunto `d` (Tabelas 3.2 e 3.3), não foi possível nem mesmo avaliar a Relaxação Linear (campos marcados com um traço nas tabelas). Nestes casos, a memória disponível não foi suficiente nem mesmo para carregar o modelo. Isto acontece pois o número de variáveis e restrições do problema, $O(nm)$, dificulta o uso desta formulação para instâncias muito grandes.

Na Tabela 3.4 podemos observar o comportamento de ambas as reformulações para instâncias k-ACM-PV. Os *gaps* associados a LP_{MF} são, na média, de 0.03%, sendo que para 10 instâncias a Relaxação Linear obteve soluções inteiras. Entretanto, o tempo requerido para o cálculo da relaxação é muito alto: por exemplo, para GRID_20_200_3, a avaliação da Relaxação Linear consome quase dois dias de CPU. A reformulação MTZ, por sua vez, apresenta *gap* médio de 4.29%, e o tempo de computação da Relaxação Linear é, novamente, praticamente desprezível.

Em linhas gerais, observamos que a reformulação Multi-Fluxo apresentou limites de programação linear sistematicamente mais fortes do que aqueles gerados pela reformulação MTZ, com maior custo computacional. Este resultado era esperado, pois formulações multi-fluxo tendem a ser fortes (ver, por exemplo, Maculan (1987); Bahiense et al. (2003)). Por outro lado, de acordo com Desrochers e Laporte (1991), formulações baseadas em restrições de Miller-Tucker-Zemlin tendem a gerar limites de Relaxação Linear fracos. Naquele estudo, os autores mostraram que as restrições (3.19) não induzem facetas da envoltória convexa do

Dados da instância				MF		MTZ		
	n	m	k	LP_{MF}	$t(s)$	LP_{MTZ}	$t(s)$	$\frac{LP_{MF}}{LP_{MTZ}}$
g25-4-01	25	50	20	219.000	0.81	188.000	0.01	1.16
g25-4-02	25	48	20	607.000	0.16	592.000	0	1.03
g25-4-03	25	50	20	462.000	0.25	449.000	0.01	1.03
g25-4-04	25	48	20	618.000	0.56	610.000	0.01	1.01
g25-4-05	25	50	20	573.000	0.84	537.000	0	1.07
g50-4-01	50	98	20	459.000	5.97	357.000	0	1.29
g50-4-02	50	99	20	421.000	2.21	343.000	0	1.23
g50-4-03	50	99	20	565.000	3.89	427.000	0.01	1.32
g50-4-04	50	100	20	434.000	3.70	364.000	0.01	1.19
g50-4-05	50	100	20	387.000	3.97	290.000	0.01	1.33
g75-4-01	75	149	20	364.250	17.57	263.000	0.01	1.38
g75-4-02	75	150	20	290.000	12.96	229.000	0.01	1.27
g75-4-03	75	149	20	411.000	18.66	292.000	0.01	1.41
g75-4-04	75	150	20	427.750	16.43	286.000	0.01	1.50
g75-4-05	75	150	20	284.000	12.81	199.000	0.01	1.43
g100-4-01	100	200	20	362.800	44.36	242.000	0.01	1.50
g100-4-02	100	200	20	334.428	43.99	198.000	0.01	1.69
g100-4-03	100	200	20	408.666	38.17	263.000	0.02	1.55
g100-4-04	100	199	20	439.500	35.32	323.000	0.01	1.36
g100-4-05	100	199	20	382.764	44.82	220.000	0.01	1.74
g200-4-01	200	400	20	305.454	416.05	165.000	0.03	1.85
g200-4-02	200	400	20	297.500	374.34	140.000	0.03	2.13
g200-4-03	200	400	20	300.000	222.86	117.000	0.02	2.56
g200-4-04	200	399	20	295.000	406.20	143.000	0.01	2.06
g200-4-05	200	399	20	353.571	287.05	169.000	0.03	2.09
g400-4-01	400	800	20	252.857	835.00	103.000	0.04	2.45
g400-4-02	400	799	20	327.352	6794.77	97.000	0.05	3.37
g400-4-03	400	799	20	299.333	884.19	127.000	0.05	2.36
g400-4-04	400	800	20	302.200	2459.71	123.000	0.05	2.46
g400-4-05	400	800	20	314.583	9076.43	103.000	0.06	3.05

Tabela 3.1: Limites de Relaxação Linear - Conjunto g

conjunto de soluções viáveis para o *Problema do Caixeiro Viajante*, ao contrário das desigualdades de eliminação de sub-circuitos propostas por Dantzig et al. (1954). Além disso, também verificaram que as restrições (3.21), apesar de serem mais fortes do que (3.19), ainda são mais fracas do que as apresentadas por Dantzig et al. (1954).

3.3.2 Análise dos algoritmos *Branch-and-bound*

Devido às dificuldades de se avaliar o limite dual LP_{MF}, apresentadas na Seção anterior, o algoritmo B&B que emprega a reformulação Multi-Fluxo só pôde ser empregado para tentar resolver instâncias de dimensões pequenas e moderadas. Por este motivo, nesta seção daremos ênfase aos resultados do algoritmo B&B que emprega a reformulação MTZ.

As Tabelas 3.5, 3.6 e 3.7 e 3.8 apresentam os resultados computacionais de tal algoritmo. As primeiras colunas fornecem os dados das instâncias, como anteriormente. A quinta coluna apresenta o número de nós abertos na árvore de enumeração e a sexta coluna apresenta o tempo gasto pelo algoritmo. A sétima coluna apresenta o melhor limite superior encontrado e a última coluna apresenta o *status* do algoritmo ao final da execução. Uma entrada "OPT" nesta coluna indica que a instância foi resolvida na otimalidade. Já uma entrada "OFM" indica que o algoritmo utilizou toda a memória disponível, e por esse motivo sua execução foi interrompida. A indicação (+) à direita de OPT ou OFM significa que o limite superior encontrado ao longo do algoritmo B&B é melhor que o melhor limite superior conhecido até este estudo, de acordo com as entradas mais recentes da KCTLib[4].

Conforme pode ser verificado, com a reformulação MTZ são fornecidos 67 novos certificados de otimalidade e 16 novos melhores limites superiores, considerando apenas as instâncias listadas na KCTLib.

A baixa qualidade do limite de programação linear da reformulação MTZ levou o algoritmo B&B a abrir muitos nós na árvore de enumeração, em grande parte dos casos. Como sabemos, bons limites duais e primais são fundamentais para o desempenho de um algoritmo desse tipo, uma vez que permitem cortes significativos na árvore de enumeração (Wolsey (1998)). O elevado número de nós abertos na árvore de enumeração fez com que, em muitas execuções, a quantidade de memória disponível não fosse suficiente para a execução do programa. Por este motivo, o algoritmo foi interrompido pelo sistema operacional. Isto significa que, na prática, também não conseguimos resolver instâncias grandes com a reformulação MTZ. Assim sendo, ambas as reformulações propostas apresentam dificuldades em relação ao consumo de memória, já que as $O(nm)$ variáveis e restrições da reformulação Multi-Fluxo também dificultam o uso do algoritmo B&B que a emprega.

Para a reformulação Multi-Fluxo, damos destaque aos resultados obtidos para as instâncias `NWG`, apresentadas na Tabela 3.9,. Para este conjunto de instâncias, o algoritmo B&B que emprega a reformulação Multi-Fluxo é, na maioria dos casos, mais rápido do que o correspondente algoritmo que emprega da reformulação MTZ. Comparando os resultados da Tabela 3.9 com aqueles obtidos pela reformulação MTZ, apresentados na Tabela 3.8, observarmos que o algoritmo baseado na reformulação Multi-Fluxo conseguiu resolver as instâncias 20×20, enquanto que o algoritmo B&B que emprega a reformulação MTZ não conseguiu.

[4] `http://iridia.ulb.ac.be/~cblum/kctlib/` – Acessado no dia 08/06/2008.

3.4 Comentários

Neste capítulo, apresentamos duas reformulações para o k-ACM. Na primeira delas, a conectividade das soluções viáveis é garantida por meio de restrições de balanço de fluxo. Na segunda, as desigualdades de eliminação de sub-rotas de Miller et al. (1960) foram empregadas. Os experimentos computacionais que realizamos confirmaram que a reformulação baseada em fluxos fornece limites de Relaxação Linear mais fortes que aqueles fornecidos pela reformulação que emprega as restrições de Miller et al. (1960).

A reformulação baseada em fluxos requer, entretanto, uma maior capacidade de recursos computacionais, apresentando alto consumo de tempo e memória. Este aspecto limita, na prática, o seu uso em algoritmos B&B para resolver instâncias de tamanho moderado. Na próxima seção apresentamos um procedimento baseado em Relaxação Lagrangeana para o k-ACM, onde, pelo menos parcialmente, contornamos as dificuldades associadas ao uso da reformulação Multi-Fluxo.

Dados da instância				MF		MTZ		
	n	m	k	LP_{MF}	$t(s)$	LP_{MTZ}	$t(s)$	$\frac{LP_{MF}}{LP_{MTZ}}$
bb15x15_1	225	420	20	257.000	790.39	59.000	0.02	4.36
			40	613.170	2427.98	198.000	0.03	3.09
			60	973.596	2230.31	431.000	0.04	2.26
			80	1335.000	915.18	779.000	0.05	1.71
bb15x15_2	225	420	20	233.583	809.50	47.000	0.02	4.97
			40	578.091	314.86	162.000	0.03	3.57
			60	926.576	335.60	349.000	0.03	2.65
			80	1289.737	1163.30	614.000	0.06	2.10
bb45x5_1	225	400	20	295.500	390.89	48.000	0.03	6.16
			40	682.882	669.35	180.000	0.04	3.79
			60	1098.176	1055.67	415.000	0.04	2.65
			80	1289.297	892.37	740.000	0.04	1.74
bb45x5_2	225	400	20	272.947	412.14	71.000	0.02	3.84
			40	644.785	462.78	260.000	0.02	2.94
			60	1110.750	2903.91	521.000	0.04	2.13
			80	1616.050	2379.05	894.000	0.05	1.80
bb33x33_1	1089	2112	100	-	-	316.000	0.23	-
			200	-	-	1144.000	0.37	-
			300	-	-	2370.000	0.41	-
			500	-	-	6175.000	0.62	-
b33x33_2	1089	2112	100	-	-	274.000	0.31	-
			200	-	-	1007.000	0.38	-
			300	-	-	2234.000	0.37	-
			500	-	-	6140.0000	0.53	-
g400-4-01	400	800	40	562.833	2281.68	301.000	0.08	1.87
			80	1304.000	3656.95	889.000	0.09	1.47
			120	2132.000	2871.22	1639.000	0.08	1.30
			160	3061.500	12296.15	2564.000	0.10	1.19
g400-4-05	400	800	40	667.361	2905.24	287.000	0.06	2.33
			80	1443.182	2974.70	845.000	0.06	1.71
			120	2290.250	4626.19	1584.000	0.10	1.45
			160	3192.000	4217.41	2473.000	0.10	1.29
			200	4156.000	3651.64	3530.000	0.22	1.18
g1000-4-01	1000	2000	200	-	-	2092.000	0.34	-
			300	-	-	3881.000	0.38	-
			400	-	-	6100.000	0.43	-
			500	-	-	8724.000	0.55	-
g1000-4-05	1000	2000	200	-	-	2291.000	0.32	-
			300	-	-	4267.000	0.48	-
			400	-	-	6674.000	0.48	-
			500	-	-	9401.000	0.55	-

Tabela 3.2: Limites de Relaxação Linear - Conjunto `d`, Parte I

Dados da instância				MF		MTZ		
	n	m	k	LP_{MF}	$t(s)$	LP_{MTZ}	$t(s)$	$\frac{LP_{MF}}{LP_{MTZ}}$
steinc15	500	2500	50	-	-	71.000	0.13	-
			100	-	-	233.000	0.24	-
			200	-	-	878.000	0.27	-
			250	-	-	1353.000	0.29	-
steinc5	500	625	50	767.261	8124.35	197.000	0.07	3.89
			100	1709.400	3359.50	746.000	0.08	2.29
			200	4271.000	17132.52	3123.000	0.11	1.37
			250	-	28784.53	4859.000	0.11	-
steind5	1000	1250	100	-	-	378.000	0.22	-
			200	-	-	1460.000	0.26	-
le450_15a	450	8168	45	-	-	45.000	0.58	-
			135	-	-	179.000	0.66	-
			225	-	-	425.000	0.87	-
			405	-	-	1369.000	1.25	-

Tabela 3.3: Limites de Programação Linear - Conjunto `d`, Parte II

Dados da instância				MF		MTZ		
	n	m	k	LP_{MF}	$t(s)$	LP_{MTZ}	$t(s)$	$\frac{LP_{MF}}{LP_{MTZ}}$
GRID_10_50_1	100	180	50	16050.000	280.40	15270.385	0.02	1.05
GRID_10_50_2	100	180	50	14026.000	38.88	13785.071	0.02	1.02
GRID_10_50_3	100	180	50	13117.000	234.85	12591.333	0.03	1.04
GRID_10_50_4	100	180	50	14240.000	362.40	13386.500	0.02	1.06
GRID_10_50_5	100	180	50	17078.750	469.26	16204.750	0.03	1.05
GRID_15_112_1	225	420	112	38321.500	8161.55	37140.354	0.07	1.03
GRID_15_112_2	225	420	112	32061.000	8192.85	30937.000	0.05	1.04
GRID_15_112_3	225	420	112	31321.000	14822.86	30433.250	0.07	1.03
GRID_15_112_4	225	420	112	30681.000	7247.34	29068.520	0.06	1.06
GRID_15_112_5	225	420	112	29869.000	10541.46	28660.217	0.06	1.04
GRID_20_200_1	400	760	200	55443.000	113885.98	53126.764	0.09	1.04
GRID_20_200_2	400	760	200	54894.000	63061.88	52526.017	0.10	1.05
GRID_20_200_3	400	760	200	53912.000	161523.95	51150.139	0.13	1.05
GRID_20_200_4	400	760	200	53512.000	106490.39	51366.667	0.14	1.04
GRID_20_200_5	400	760	200	62364.000	81925.95	60524.319	0.12	1.03

Tabela 3.4: Limites de Relaxação Linear - Conjunto `NWG`

Dados da instância				Resultados do B&B			
	n	m	k	Nós B&B	$t(s)$	Melhor LS	Status
g25-4-01	25	50	20	6565	4.24	219	OPT
g25-4-02	25	48	20	44	0.11	607	OPT
g25-4-03	25	50	20	97	0.19	464	OPT
g25-4-04	25	48	20	4	0.12	620	OPT
g25-4-05	25	50	20	193	0.39	573	OPT
g50-4-01	50	98	20	1007	2.43	460	OPT
g50-4-02	50	99	20	33	0.59	421	OPT
g50-4-03	50	99	20	259	1.06	565	OPT
g50-4-04	50	100	20	0	0.14	434	OPT
g50-4-05	50	100	20	863	1.78	387	OPT
g75-4-01	75	149	20	8	0.50	366	OPT
g75-4-02	75	150	20	740	2.64	295	OPT
g75-4-03	75	149	20	22	0.48	412	OPT
g75-4-04	75	150	20	432	2.30	430	OPT
g75-4-05	75	150	20	0	0.12	284	OPT
g100-4-01	100	200	20	0	0.25	363	OPT
g100-4-02	100	200	20	1064	3.70	335	OPT
g100-4-03	100	200	20	11	0.70	412	OPT
g100-4-04	100	199	20	9	1.20	442	OPT
g100-4-05	100	199	20	2034	8.88	388	OPT
g200-4-01	200	400	20	3041	20.63	308	OPT
g200-4-02	200	400	20	1447	11.20	299	OPT
g200-4-03	200	400	20	343	5.88	300	OPT
g200-4-04	200	399	20	838	11.96	304	OPT
g200-4-05	200	399	20	859	8.26	357	OPT
g400-4-01	400	800	20	4650	59.14	253	OPT
g400-4-02	400	799	20	81000	1060.75	328	OPT
g400-4-03	400	799	20	34385	315.11	302	OPT
g400-4-04	400	800	20	17129	198.92	306	OPT
g400-4-05	400	800	20	93225	1060.32	320	OPT

Tabela 3.5: Resultados B&B – MTZ, Conjunto g.

Dados da instância				Resultados do B&B			
	n	m	k	Nós B&B	$t(s)$	Melhor LS	Status
bb15x15_1	225	420	20	12600	81.85	257	OPT
			40	1916955	24425.45	642	OPT
			60	6497787	153320.69	977	OPT
			80	15423298	328839.66	1335	OPT
bb15x15_2	225	420	20	49727	273.16	253	OPT
			40	767714	6508.41	585	OPT
			60	3008220	31317.61	927	OPT
			80	1977285	36628.68	1290	OPT
bb45x5_1	225	400	20	120889	1067.70	306	OPT
			40	2507388	53752.90	695	OPT
			60	7594785	262528.78	1109	OFM+
			80	4047449	166184.40	1656	OFM
bb45x5_2	225	400	20	119539	732.20	302	OPT
			40	882105	7658.12	654	OPT
			60	10675871	233609.23	1122	OFM
			80	6051216	156046.94	1617	OFM
bb33x33_1	1089	2112	100	1267297	53100.21	-	OFM
			200	1205046	36779.94	-	OFM
			300	1193644	41915.70	-	OFM
			500	1173298	29093.88	-	OFM
bb33x33_2	1089	2112	100	1280980	69437.97	-	OFM
			200	1205510	50291.80	-	OFM
			300	1209131	36352.32	-	OFM
			500	1186953	23279.84	-	OFM
g400-4-01	400	800	40	13423	155.50	563	OPT
			80	83835	979.42	1304	OPT
			120	259250	3322.27	2132	OPT+
			160	7924988	132412.57	3062	OPT
g400-4-05	400	800	40	110399	1709.59	670	OPT+
			80	220245	3774.46	1445	OPT
			120	319431	6083.98	2291	OPT+
			160	1800750	32882.25	3192	OPT+
			200	1131667	13962.67	4156	OPT

Tabela 3.6: Resultados B&B – MTZ, Conjunto `d`, Parte I.

Dados da instância				Resultados do B&B			
	n	m	k	Nós B&B	$t(s)$	Melhor LS	Status
g1000-4-01	1000	2000	200	2669519	215106.66	3308	OPT
			300	2776041	234027.90	5325	OFM
			400	2041469	171327.76	7572	OFM+
			500	1195424	37014.39	10067	OFM
			600	1272881	48141.14	-	OFM
			700	1281941	39393.53	15675	OFM
g1000-4-05	1000	2000	200	1624054	148251.04	3618	OPT+
			300	1651553	133471.52	5798	OFM+
			400	2819112	215221.11	8195	OFM+
			500	1263768	75442.48	10786	OFM+
			600	1348737	66647.21	13589	OFM
			700	1330346	50055.13	16674	OFM+
steinc15	500	2500	50	54781	959.16	208	OPT
			100	217092	4577.04	481	OPT
			150	169429	5542.21	802	OPT
			200	1372350	39944.70	1182	OPT
			250	1811433	54607.31	1625	OFM
steinc5	500	625	50	143032	1597.12	772	OPT
			100	232186	2783.31	1712	OPT
			150	429772	4641.91	2865	OPT
			200	1070420	13695.98	4271	OPT+
			250	1755504	20226.13	5942	OPT+
steind15	500	5000	100	829100	108768.77	454	OFM+
			300	703451	63368.52	1674	OFM
			400	816800	80688.36	2446	OFM
			500	673241	39391.24	-	OFM
steind5	1000	1250	100	514228	27822.98	1503	OPT
			200	1221944	72865.63	3440	OPT+
			300	3590487	210450.37	5817	OFM
			400	2392740	169768.88	8685	OFM+
			500	1710306	52064.24	12136	OFM
le450_15a	450	8168	45	47399	4982.81	59	OPT
			135	76999	5501.04	226	OPT
			225	571075	99599.15	471	OPT
			405	540482	20132.95	1388	OFM

Tabela 3.7: Resultados B&B – MTZ, Conjunto `d`, Parte II.

Dados da instância				Resultados do B&B			
	n	m	k	Nós B&B	$t(s)$	Melhor LS	Status
GRID_10_50_1	100	180	50	3703560	16794.61	16052	OPT
GRID_10_50_2	100	180	50	467843	1142.42	14026	OPT
GRID_10_50_3	100	180	50	216741	535.55	13117	OPT
GRID_10_50_4	100	180	50	3307053	14087.91	14254	OPT
GRID_10_50_5	100	180	50	6027894	32373.64	17093	OPT
GRID_15_112_1	225	420	112	6287542	55682.19	38322	OPT
GRID_15_112_2	225	420	112	5645983	69518.57	32062	OFM
GRID_15_112_3	225	420	112	6103011	70307.49	31357	OFM
GRID_15_112_4	225	420	112	5674373	90554.05	30745	OFM
GRID_15_112_5	225	420	112	5635347	59829.79	29869	OFM
GRID_20_200_1	400	760	200	2885918	48282.72	56512	OFM
GRID_20_200_2	400	760	200	2888788	31041.96	-	OFM
GRID_20_200_3	400	760	200	2825857	34204.25	-	OFM
GRID_20_200_4	400	760	200	2803262	39177.20	-	OFM
GRID_20_200_5	400	760	200	2780369	43301.99	64839	OFM

Tabela 3.8: Resultados B&B, MTZ, Conjunto NWG.

Dados da instância				Resultados do B&B			
	n	m	k	Nós B&B	$t(s)$	Melhor LS	Status
GRID_10_50_1	100	180	50	9	1247.08	16052	OPT
GRID_10_50_2	100	180	50	0	39.90	14026	OPT
GRID_10_50_3	100	180	50	0	236.43	13117	OPT
GRID_10_50_4	100	180	50	24	10579.82	14254	OPT
GRID_10_50_5	100	180	50	10	2904.40	17093	OPT
GRID_15_112_1	225	420	112	6	61371.59	38322	OPT
GRID_15_112_2	225	420	112	0	8733.51	32061	OPT
GRID_15_112_3	225	420	112	0	28753.73	31321	OPT
GRID_15_112_4	225	420	112	0	7715.17	30681	OPT
GRID_15_112_5	225	420	112	0	11297.77	29869	OPT
GRID_20_200_1	400	760	200	0	122720.93	55443	OPT
GRID_20_200_2	400	760	200	0	71718.50	54894	OPT
GRID_20_200_3	400	760	200	0	510935.61	53912	OPT
GRID_20_200_4	400	760	200	0	109211.57	53512	OPT
GRID_20_200_5	400	760	200	0	96651.52	62364	OPT

Tabela 3.9: Resultados B&B - Multi-Fluxo, Conjunto NWG.

Capítulo 4

Uma Relaxação Lagrangeana para o k-ACM

Neste capítulo apresentamos um algoritmo baseado em Relaxação Lagrangeana que explora a reformulação Multi-Fluxo, proposta para o k-ACM no Capítulo 3. Com esta abordagem, pretendemos avaliar o limite inferior LP_{MF} sem explicitamente resolver programas lineares. No procedimento aqui proposto, utilizamos informação dual (custos modificados conforme a solução Lagrangeana) para guiar uma heurística, que gera soluções viáveis para o k-ACM.

4.1 Relaxação Lagrangeana da Reformulação Multi-Fluxo

Para aplicarmos a técnica de Relaxação Lagrangeana para resolver um problema de Programação Inteira, devemos inicialmente identificar um conjunto de restrições convenientes para serem relaxadas e dualizadas. Idealmente, o problema relaxado deve ser de fácil resolução. Para o caso específico de aplicação considerado nesta dissertação, duas opções de decomposição se destacam:

- Relaxar e dualizar as restrições de cardinalidade e acoplamento (3.14)-(3.18). Neste caso, preserva-se a estrutura de rede e os problemas Lagrangeanos envolvem a solução de um problema de caminho mínimo com origem em $n+1$ e múltiplos destinos ($v \in V'$).
- Relaxar e dualizar as restrições de balanço e fluxo (3.11)-(3.13). Neste caso, como veremos, o PRL pode ser facilmente resolvido por inspeção.

Alternativas similares de decomposição foram estudadas para uma formulação multi-fluxo apresentada para o Problema de Projeto de Redes com Múltiplas Mercadorias Capacitado (PPRMMC, do inglês *Multicommodity Capacitated Network Design Problem*) por Gendron e Crainic (1994). Os experimentos computacionais realizados para o PPRMMC naquela referência revelaram que, na prática, limites inferiores de pior qualidade foram obtidos ao se empregar a segunda alternativa.

Holmberg e Yuan (1998), por sua vez, optaram pelo segundo esquema ao propor uma Relaxação Lagrangeana para o PPRMMC. A sua escolha foi justificada por dois argumentos

principais: *(i)* os sub-problemas associados ao segundo esquema podem ser resolvidos mais facilmente, com menor esforço computacional e *(ii)* os limites duais teóricos associados aos dois esquemas são os mesmos. Outro trabalho que empregou a segunda alternativa de decomposição foi a Relaxação Lagrangeana proposta por Bahiense et al. (2003), para o Problema de Steiner em Grafos.

Nesta dissertação, optamos pela segunda alternativa de decomposição do problema ao propor um procedimento baseado em Relaxação Lagrangeana para o k-ACM. Assim sendo, associando multiplicadores Lagrangeanos reais, irrestritos em sinal, $\{\alpha_q \in R : q \in V'\}$, $\{\beta_{qi} \in R : i, q \in V', i \neq q\}$, $\{\gamma_q \in R : q \in V'\}$ e $\{\lambda \in R\}$ respectivamente às restrições (3.11), (3.12), (3.13) e (3.14), limites inferiores válidos para o k-ACM podem ser obtidos ao se resolver o Problema de Relaxação Lagrangeana (PRL):

$$\begin{aligned} z(\alpha,\beta,\gamma,\lambda) = \min \Bigg\{ & \sum_{a\in A} c_a x_a + \sum_{q\in V'} \alpha_q \left(\sum_{a\in\delta^+(n+1)} f_a^q - 1 \right) + \\ & \sum_{q\in V'} \sum_{i\in V', i\neq q} \beta_{q,i} \left(\sum_{a\in\delta^+(i)} f_a^q - \sum_{a\in\delta^-(i)} f_a^q \right) + \\ & \sum_{q\in V'} \gamma_q \left(\sum_{a\in\delta^+(q)} f_a^q - \sum_{a\in\delta^-(q)} f_a^q + 1 \right) + \\ & \lambda \left(\sum_{q\in V'} f^q_{[n+1,n+2]} - (k+2) \right) : (x,f) \in (B^{2(m+n)+1}, R_+^{n(2(m+1)+n)+1}) \cap P_{LAGR} \Bigg\}, \end{aligned} \tag{4.1}$$

onde P_{LAGR} é dado pela interseção de (3.15) - (3.18).

Após procedermos a uma reorganização de seus termos, o PRL pode ser reescrito como:

$$\begin{aligned} z(\alpha,\beta,\gamma,\lambda) = \min \left\{ \sum_{a\in A} c_a x_a + \sum_{q\in V'} \sum_{a\in A} \overline{c}_a^q f_a^q : (x,f) \in \left(B^{2(m+n)+1}, R_+^{n(2(m+1)+n)+1} \right) \cap P_{LAGR} \right\} \\ + \text{const}(\alpha,\beta,\gamma,\lambda). \end{aligned} \tag{4.2}$$

Em (4.2), $\text{const}(\alpha,\beta,\gamma,\lambda) = \sum_{q\in V'}(\gamma_q - \alpha_q) - (k+2)\lambda$ é um valor independente de x e f e os custos Lagrageanos $\overline{c}^q_{[i,j]}$ são dados por:

$$\overline{c}^q_{[i,j]} = \begin{cases} \beta_{qi} - \beta_{qj}, & i,j \in V', i,j \neq q \\ \gamma_q - \beta_{qj}, & i,j \in V', i = q, j \neq q \\ \beta_{qi} - \gamma_q, & i,j \in V', j = q, i \neq q \\ \alpha_q - \gamma_q, & i = n+1, j = q \neq n+2 \\ \alpha_q - \gamma_q + \lambda, & i = n+1, j = q = n+2 \\ \alpha_q - \beta_{n+2,q} + \lambda, & i = n+1, j = n+2, q \neq n+2. \end{cases} \tag{4.3}$$

Como de costume em Relaxação Lagrangeana, desejamos conhecer o conjunto de multiplicadores $(\alpha,\beta,\gamma,\lambda)$ que maximiza (4.2), isto é, que resolvem o Problema Dual Lagrangeano (DL):

$$\textbf{(DL)} \quad z_d = \max_{\alpha,\beta,\gamma,\lambda}\{\min z(\alpha,\beta,\gamma,\lambda)\}. \tag{4.4}$$

A função DL é côncava, linear por partes e não-diferenciável (Nemhauser e Wolsey (1988)), e seu máximo pode ser obtido por diferentes técnicas, dentre as quais podemos citar o Método do Volume (Barahona e Anbil (2000); Bahiense et al. (2003)), o Método de Feixes (Lemarechal (1974)) e o Método dos Subgradientes (Held et al. (1974)). Por sua simplicidade de implementação e por apresentar bons resultados na prática, neste trabalho empregamos o Método dos Subgradientes.

Antes de descrever como resolver o PRL, deve-se notar que se um arco $[i,j]$ pertence à solução ótima de (4.2), então todas as variáveis de fluxo $\{f^q_{[i,j]} : q \in V', \overline{c}^q_{[i,j]} < 0\}$ devem assumir seus limites superiores. Por outro lado, as variáveis $\{f^q_{[i,j]} : q \in V', \overline{c}^q_{[i,j]} > 0\}$ devem assumir seus limites inferiores. Estas observações dão origem a um procedimento que resolve o PRL. Inicialmente, calculamos o *custo Lagrangeano equivalente* s_a, associado a cada arco $[i,j] \in A$:

$$s_a = c_a + \sum_{q\in V': \overline{c}^q_a < 0} \overline{c}^q_a. \tag{4.5}$$

Em seguida, listamos os arcos de A em ordem não-decrescente de seus custos equivalentes e escolhemos:

- (passo 0) o arco $[n+1, n+2]$;
- (passo 1) os k arcos em A_E com os menores valores de s_a;
- (passo 2) os $n-(k+1)$ arcos com os menores valores de s_a dentre aqueles em $\{[n+1,i] : i \in V\}$;
- (passo 3) o arco que sai de $n+2$ com menor valor de s_a.

Assumindo que $A(\alpha, \beta, \gamma, \lambda)$ denota o conjunto de arcos escolhidos nos passos acima, para finalizar a resolução do PRL fazemos $f_a^q = 1$, $\forall q \in V'$, $\forall a \in A(\alpha, \beta, \gamma, \lambda)$ se $\overline{c}_a^q \leq 0$. Todas as demais variáveis de fluxo assumem valores nulos.

Observe que o Problema de Relaxação Lagrangeana (4.2) possui a Propriedade da Integralidade e que, assim sendo, $z_d = LP_{MF}$. O nosso objetivo ao propor este procedimento para o k-ACM consiste em tentar aproximar z_d, empregando para isto esforço computacional significativamente inferior àquele requerido para avaliar LP_{MF}.

4.1.1 Reforçando a estrutura de rede no PRL

Uma vez que as restrições de balanço de fluxo (3.11)-(3.13) são relaxadas, uma solução do PRL (4.2) pode incluir mais de um arco incidente a um mesmo vértice $i \in V$. Este tipo de solução viola a propriedade de que, em qualquer arborescência, apenas um arco deve incidir sobre cada vértice $v \neq n + 1$.

Para contornar esta indesejável característica das soluções Lagrangeanas, sem no entanto perder a propriedade de separabilidade na resolução do PRL, adicionamos as seguintes desigualdades válidas ao conjunto de restrições que definem a região poliédrica P_{LAGR}:

$$\sum_{a \in A_E \cap \delta^-(i)} x_a \leq 1, \forall i \in V. \tag{4.6}$$

Apesar de serem redundantes para (3.10), estas restrições reforçam a estrutura de rede do PRL. Esta estratégia é especialmente útil quando a informação dual Lagrangeana é usada para guiar uma heurística construtiva para encontrar soluções viáveis para o problema, como será o caso discutido na Seção 4.3.

Para resolver o PRL incluindo as restrições (4.6), basta uma simples alteração no passo 1 acima. Ao invés de escolher os k arcos em A_E com os menores valores de s_a, apenas selecionamos um arco $a = [i, j] \in A_E$ se nenhum outro arco em $A_E \cap \delta^-(j)$ tiver sido anteriormente escolhido.

4.2 Otimização via Método dos Subgradientes

Para resolver o Problema Dual Lagrangeano (4.4), implementamos uma versão do Método dos Subgradientes (Held et al. (1974)). Em nossa implementação, a principal modificação introduzida no método diz respeito à escolha da direção de busca. Por simplicidade, assuma que $\pi^p = (\alpha^p, \beta^p, \gamma^p, \lambda^p) \in R^l$ denota o vetor de multiplicadores Lagrangeanos (de dimensão l apropriada) no início da p-ésima iteração do Método dos Subgradientes. Assuma também que $w^p \in R^l$ é um subgradiente da função DL (4.4).

Na implementação de Held et al. (1974), os mulitiplicadores são atualizados da seguinte forma:

$$\pi_i^{p+1} = \pi_i^p + t^p d_i^p, i = 1, \ldots, l. \tag{4.7}$$

onde a direção de busca $d_p \in R^l$ é um subgradiente w^p e o tamanho do passo $t^p \in R_+$ é dado por:

$$t^p = \varepsilon^p \frac{\overline{z} - \underline{z}^p}{\sum_{i=1}^{l} (w_i^p)^2}. \tag{4.8}$$

Em (4.8), ϵ^p denota um escalar real no intervalo $(0, 2]$ e $\overline{z}$, $\underline{z}^p$ são, respectivamente, um limite superior para a função objetivo ótima e o valor do PRL formulado e resolvido na iteração p.

Em uma tentativa de conferir melhores propriedades de convergência ao Método dos Subgradientes, não usamos o subradiente w^p puro como direção de busca. Ao contrário, seguimos a sugestão proposta por Crowder (1976), que foi aplicada com sucesso por Holmberg e Yuan (1998). Neste procedimento, a direção de busca empregada na iteração p leva em consideração os subgradientes obtidos nas iterações anteriores, $1 \ldots p-1$. Com exceção da primeira, para a qual $d^1 = w^1$, nas demais iterações d^p é avaliada como:

$$d^p = \frac{1}{1+\eta} \left(w^p + \eta d^{p-1} \right), p \geq 2, \eta > 0. \tag{4.9}$$

Holmberg e Yuan (1998) observaram que $\eta = 0.7$ é um bom valor a ser usado na prática. Com relação aos demais parâmetros da Relaxação Lagrangeana, iniciamos $\epsilon^1 = 2$ e então atualizamos ϵ^{p+1} para $0.9\epsilon^p$ sempre que o melhor limite Dual Lagrangeano não melhora após 300 iterações. O método é aplicado enquanto um certificado de otimalidade não é obtido (que ocorre quando o limite superior se iguala ao limite inferior) e enquanto um número máximo de 5000 iterações não for atingido.

Observe que este método de solução do problema Dual Lagrangeano exige que um limite superior ($\overline{z}$) seja conhecido, para o cálculo do passo (equação (4.8)). Na próxima seção, discutimos como limites primais são gerados para o k-ACM.

4.3 Limites superiores e Heurística Lagrangeana

Para a geração de limites superiores para o k-ACM, empregamos o algoritmo proposto por Blum (2007b), aqui denominado por `DynamicTree`. Este algoritmo envolve dois passos:

- (passo 0): Obtenção de uma árvore $U = (V_U, E_U)$ de G com pelo menos k arestas;
- (passo 1): Extração da melhor sub-árvore com k arestas de U, através de um algoritmo exato, baseado em Programação Dinâmica.

Dentre as muitas opções existentes, no primeiro passo do `DynamicTree`, Blum (2007b) sugere a utilização de uma árvore geradora de G. Uma vez que custos associados aos vértices e arestas de G podem coexistir, é necessário que uma *boa* árvore geradora equilibre estas duas fontes de custos. Um dos procedimentos propostos por Blum (2007b) para determinar

uma árvore geradora inicial consiste no emprego do algoritmo de Kruskal, onde os custos das arestas utilizadas no algoritmo, $\hat{c}_{(i,j)}$, são dados por:

$$\hat{c}_{(i,j)} = c_{(i,j)} + d_i + d_j \mid (i,j) \in E. \tag{4.10}$$

Note que $\hat{c}_{(i,j)} := c_{(i,j)}$ para o caso do k-ACM-CA e que $\hat{c}_{(i,j)} := d_i + d_j$ para o caso do k-ACM-PV.

O (passo 1) se baseia no fato, inicialmente observado por Fischetti et al. (1994), de que o k-ACM pode ser resolvido em tempo polinomial se o grafo de entrada do problema for uma árvore. Blum (2007b), entretanto, generalizou o algoritmo de Fischetti et al. (1994), para ser aplicado sobre grafos valorados tanto nos vértices quanto nas arestas (o Apêndice A apresenta uma descrição detalhada deste algoritmo de Programação Dinâmica). Observe, portanto, que os custos modificados pela equação (4.10) são usados apenas na entrada do algoritmo de Kruskal, uma vez que o algoritmo de Programação Dinânica naturalmente lida com grafos com custos nos vértices e nas arestas.

A heurística acima foi utilizada no contexto da Relaxação Lagrangeana para a geração do primeiro limite superior, imediatamente antes da primeira iteração do Método dos Subgradientes. Além disso, utilizamos o mesmo procedimento durante as demais iterações do Método dos Subgradientes, porém considerando custos modificados conforme a informação dual Lagrangeana. A cada iteração p do Método dos Subgradientes, alteramos os custos de entrada do algoritmo de Kruskal, utilizado no (passo 0) do `DynamicTree`. Para cada aresta (i,j), o custo de entrada efetivamente usado na fase construtiva é dado por:

$$\hat{c}'_{(i,j)} = \min\{(1 - x^p_{[i,j]})\hat{c}_{(i,j)},\ (1 - x^p_{[j,i]})\hat{c}_{(i,j)}\}, \tag{4.11}$$

onde $x^p_{[i,j]}$ e $x^p_{[j,i]}$ denotam o valor das variáveis de decisão associadas, respectivamente, aos arcos $[i,j]$ e $[j,i]$, na solução do PRL formulado e resolvido na iteração p. A motivação desta idéia consiste em privilegiar as arestas (i,j) de G que tenham um dos seus arcos em A_E selecionados na solução Lagrangeana na p-ésima iteração. Uma vez que esta Árvore Geradora (de mínimo custo modificado $\hat{c}'_{(i,j)}$) $U = (V_U, T_U)$ é obtida, passamos para o (passo 1). A Figura 4.1 ilustra o esquema empregado para gerar os limites superiores para a Relaxação Lagrangeana.

4.4 Fixação de variáveis

Como vimos anteriormente, no caso da aplicação aqui discutida, o PRL é um problema de otimização separável. Esta propriedade permite que em cada um dos 4 passos possamos escolher de maneira independente quais arcos devem fazer parte da solução Lagrangeana.

Para fins de ilustração, podemos dizer que a solução do PRL é dada pela união de arcos em certas *classes*: selecionamos inicialmente o arco $[n+1, n+2]$ (classe 0, que possui apenas um arco), em seguida os k arcos com menor valor de s_a da classe A_E (classe 1), escolhemos os $n-(k+1)$ arcos de $[n+1, i] : i \in V$ com menor valor de s_a (classe 2) e finalmente o arco de

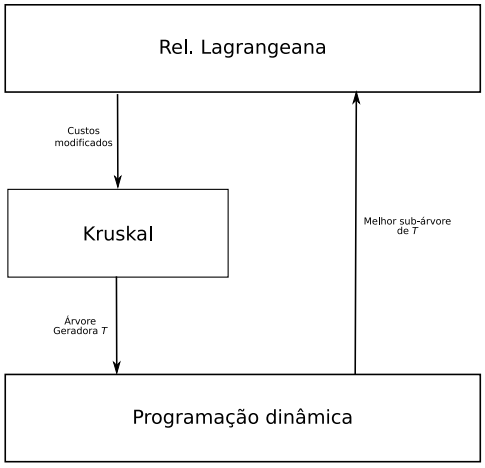

Figura 4.1: Visão geral da Heurística Lagrangeana.

menor valor de s_a dentre aqueles do conjunto $\{[n+2, r] : r \in V\}$ (classe 3). Em cada passo do algoritmo, portanto, escolhemos arcos restritos a uma única classe.

Nesse tipo de estrutura, os custos reduzidos de Programação Linear podem ser facilmente estimados. Para cada classe de arcos, comparamos os valores de s_a dos arcos que não foram selecionados para fazer parte da solução Lagrangeana com os valores de s_a dos arcos que foram selecionados na mesma classe.

Sem perda de generalidade, vamos descrever como funciona o procedimento para o cálculo dos custos reduzidos para os arcos na classe 1. Os mesmos princípios valem para os arcos nas classes 2 e 3. Assuma então que após executar o (passo 1) na p-ésima iteração do Método dos Subgradientes, um determinado arco, digamos $[i, j]$, não foi selecionado para fazer parte da solução ótima do PRL, isto é, $[i, j] \in (A_E \setminus A(\alpha, \beta, \gamma, \lambda))$. Assuma também que s^{max} denota o maior valor de s_a de um arco a, da mesma classe de $[i, j]$, tal que $x_a^p = 1$, isto é, a faz parte da solução Lagrangeana na iteração p. O custo reduzido de Programação Linear de $[i, j]$ é dado por:

$$rc_{[i,j]} := s_{[i,j]} - s^{max}. \tag{4.12}$$

Note que, se $rc_{[i,j]} + z(\alpha, \beta, \gamma, \lambda) > \overline{z}$, então $[i, j]$ não pertencerá a nenhuma arborescência ótima em D, e portanto, tanto $[i, j]$ pode ser eliminado de D quanto sua respectiva variável de decisão $x_{[i,j]}$ pode ser eliminada do modelo. Quando os arcos $[i, j]$ e $[j, i]$ forem fixados, a correspondente aresta (i, j) pode ser eliminada de G e portanto deixa de ser considerada nos procedimentos de geração de limites superiores. Este procedimento de fixação de variáveis é chamado sempre que se observa uma melhora no melhor limite inferior ou superior durante o Método dos Subgradientes.

Graças a estes testes, pode acontecer que, em uma determinada iteração do Método dos Subgradientes, as arestas de E ainda não fixadas induzam componentes desconectados em G (ver Figura 4.2). Nestes casos, o algoritmo de Kruskal aplicado sobre G irá obter uma floresta de G. Sejam $G_i = (V_i, T_i)$, $i = 1, \ldots, t$, os componentes desconectados que constituem esta

floresta. Para cada um dos possíveis componentes G_i, verificamos a cardinalidade de V_i. Se $|V_i| \leq k$, todas as arestas de T_i e vértices de V_i são removidos de G, e suas respectivas variáveis de decisão podem ser removidas do modelo.

Por outro lado, para todos os componentes G_i que possuem $|V_i| \geq k+1$, executamos o algoritmo de Programação Dinâmica considerando apenas o sub-grafo induzido pelos vértices em V_i.

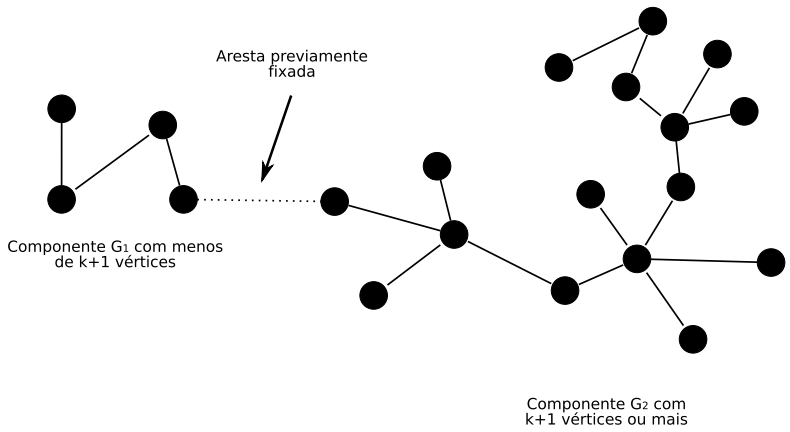

Figura 4.2: O componente G_1 tem menos de $k+1$ vértices. Neste caso, todas as variáveis associadas a ele podem ser removidas do modelo. Por outro lado, G_2 possui $k+1$ vértices ou mais, e portanto o algoritmo de Programação Dinâmica deve ser aplicado a ele.

4.5 Experimentos computacionais

Nesta seção são apresentados os resultados computacionais obtidos com a Heurística Lagrangeana que descrevemos para o k-ACM. Os mesmos conjuntos de instâncias `g`, `d` e `NWG`, descritos na Seção 3.3, são utilizados para os testes nesta Seção. Consideramos também instâncias adicionais em cada um destes conjuntos, que, devido às limitações já destacadas, não foram testadas com o algoritmo B&B que emprega a reformulação Multi-Fluxo.

Um sumário com os principais resultados computacionais é indicado na Tabela 4.1. Em cada linha da tabela, apresentamos resultados médios para cada conjunto de instâncias, identificados pela primeira coluna. As próximas três colunas apresentam o número de instâncias no conjunto, o número de certificados de otimalidade obtidos e, finalmente, o número de vezes em que o nosso limite primal foi pelo menos tão bom quanto o melhor limite conhecido na literatura ($\overline{z} \leq MVC$). As duas últimas colunas apresentam os valores médios para os *gaps* de dualidade da Relaxação Lagrangeana ($\frac{\overline{z}-z_d}{z_d}$, em valores percentuais) e a média das razões $\frac{z_d}{LP_{MF}}$.

Os resultados detalhados por sua vez são apresentados nas Tabelas 4.2, 4.3, 4.4 e 4.5.

As primeiras colunas das Tabelas 4.2 - 4.5 indicam o nome da instância e os valores de n, m e k. As próximas colunas apresentam os resultados obtidos pelo algoritmo: o melhor limite dual encontrado z_d, o melhor limite superior $\overline{z}$, o *gap* de dualidade $\frac{\overline{z}-z_d}{z_d}$, a proporção de arestas sub-ótimas fixadas e o tempo, em segundos, gasto pelo algoritmo. As próximas

3 colunas apresentam a razão z_d/LP_{MF}, o limite superior $\overline{z}_{DP}$ obtido pelo procedimento `DynamicTree` e, por último, o melhor limite superior conhecido na literatura até este estudo (MVC – *Melhor Valor Conhecido*).

	# de instâncias	$\overline{z} = z_d$	$\overline{z} \leq$MVC	Valores médios gap (%)	Valores médios $\frac{z_d}{LP_{MCFF}}$
`g`	30	4	30	1.29	0.991
`d`	74	1	41	4.89	0.981
`NWG`	24	0	-	4.70	0.966
Total	128	5	71	4.01	0.982

Tabela 4.1: Heurística Lagrangeana – visão geral dos resultados.

Na média, o limite dual Lagrangeano representa 98.2% do melhor limite teoricamente possível de ser obtido, dado por LP_{MF}. Entretanto, estes limites Lagrangeanos são obtidos em tempos computacionais uma ou duas ordens de magnitude menores do que os tempos gastos pelo CPLEX para obter LP_{MF}. Além disso, em 71 dos 128 casos testados, o limite superior obtido é pelo menos tão bom quanto o melhor limite atualmente disponível na literatura. Para 8 instâncias são fornecidos novos melhores limites superiores com a Heurística Lagrangeana aqui proposta. Além disso, os limites superiores encontrados pelo algoritmo nunca são piores em mais de 3.8% que o melhor valor conhecido na literatura. Na média, os limites superiores da literatura são apenas 0.28% melhores do que os apresentados neste trabalho. Cabe destacar que ao compararmos os limites superiores obtidos pela Heurística Lagrangeana aqui proposta com o MVC, não estamos comparando nosso algoritmo com um algoritmo em particular da literatura. Na verdade, estamos comparando a qualidade das nossas soluções viáveis com as melhores soluções encontradas pelos melhores algoritmos propostos até este estudo.

Apesar de ser muito mais rápida do que o *solver* CPLEX, a Heurística Lagrangeana requer muito mais tempo computacional do que as melhores heurísticas e meta-heurísticas conhecidas na literatura. Por exemplo, para as instâncias `g400` (com $k = 20$), as meta-heurísticas propostas em Blum e Blesa (2005) nunca requerem mais do que 20 segundos de computação (em um PC Athlon 1100 MHz). Para a mesma instância, o procedimento aqui proposto emprega cerca de 280 segundos.

Dados da instância				Heurística Lagrangeana							
	n	m	k	z_d	$\overline{z}$	gap (%)	Fix (%)	$t(s)$	$\frac{z_d}{LP_{MF}}$	$\overline{z}_{DP}$	MVC
g25-4-01	25	50	20	218.009	219	0.00	34.00	1.29	0.995	219	219
g25-4-02	25	48	20	606.109	607	0.00	50.00	1.12	0.998	607	607
g25-4-03	25	50	20	461.633	464	0.43	50.00	1.69	0.992	464	464
g25-4-04	25	48	20	617.293	620	0.32	39.58	1.79	0.999	620	620
g25-4-05	25	50	20	571.702	573	0.17	36.00	1.75	0.998	573	573
g50-4-01	50	98	20	456.884	460	0.66	30.61	7.04	0.995	464	460
g50-4-02	50	99	20	417.674	421	0.72	44.44	5.56	0.992	421	421
g50-4-03	50	99	20	563.735	565	0.18	46.46	6.79	0.998	580	565
g50-4-04	50	100	20	432.588	434	0.23	55.00	6.16	0.997	434	434
g50-4-05	50	100	20	386.014	387	0.00	47.00	3.78	0.997	406	387
g75-4-01	75	149	20	363.564	366	0.55	45.64	10.94	0.998	366	366
g75-4-02	75	150	20	282.996	295	4.24	40.00	8.42	0.975	307	295
g75-4-03	75	149	20	408.539	412	0.73	36.91	14.58	0.994	428	412
g75-4-04	75	150	20	426.888	430	0.70	47.33	14.00	0.997	444	430
g75-4-05	75	150	20	283.054	284	0.00	84.67	4.49	0.997	284	284
g100-4-01	100	200	20	359.547	363	0.83	54.50	18.01	0.991	370	363
g100-4-02	100	200	20	333.448	335	0.30	47.50	14.90	0.997	338	335
g100-4-03	100	200	20	408.295	412	0.73	43.50	17.35	0.999	412	412
g100-4-04	100	199	20	431.537	442	2.32	33.67	19.24	0.982	442	442
g100-4-05	100	199	20	382.001	388	1.31	40.20	19.88	0.998	388	388
g200-4-01	200	400	20	301.400	308	1.99	32.75	72.44	0.987	318	308
g200-4-02	200	400	20	292.897	299	2.05	42.50	63.81	0.985	299	299
g200-4-03	200	400	20	298.764	300	0.33	51.25	62.14	0.996	300	300
g200-4-04	200	399	20	290.305	304	4.48	29.82	75.50	0.984	310	304
g200-4-05	200	399	20	350.842	357	1.71	37.34	67.84	0.992	357	357
g400-4-01	400	800	20	247.931	253	2.02	67.75	261.26	0.981	253	253
g400-4-02	400	799	20	321.931	328	1.86	33.67	301.55	0.983	328	328
g400-4-03	400	799	20	294.844	302	2.37	53.44	279.99	0.985	306	302
g400-4-04	400	800	20	298.728	306	2.34	43.00	288.71	0.989	306	306
g400-4-05	400	800	20	303.907	320	5.26	21.50	315.71	0.966	320	320

Tabela 4.2: Resultados Heurística Lagrangeana, Conjunto g.

4.6 Comentários

Neste capítulo, apresentamos um procedimento baseado em Relaxação Lagrangeana para a reformulação Multi-fluxo proposta para o k-ACM. Uma heurística para a obtenção de limites superiores, que emprega informação dual, também foi proposta.

Dados da instância				Heurística Lagrangeana							
	n	m	k	z_d	$\overline{z}$	gap (%)	Fix (%)	$t(s)$	$\frac{z_d}{LP_{MF}}$	$\overline{z}_{DP}$	MVC
bb15x15_1	225	420	20	254.382	257	0.79	48.10	70.46	0.989	267	257
			40	596.655	642	7.54	1.43	90.71	0.973	645	642
			60	954.177	977	2.31	1.43	93.30	0.980	1012	977
			80	1328.778	1335	0.45	1.67	100.70	0.995	1443	1335
bb15x15_2	225	420	20	228.430	253	10.51	2.14	81.48	0.978	253	253
			40	572.795	585	2.09	13.81	86.24	0.990	649	585
			60	921.850	927	0.54	21.43	87.03	0.995	1030	927
			80	1282.391	1290	0.55	14.05	96.31	0.994	1395	1290
bb45x5_1	225	400	20	292.218	306	4.45	6.25	79.28	0.989	342	306
			40	665.818	695	4.36	1.00	87.48	0.975	754	695
			60	1090.407	1107	1.47	1.50	94.58	0.993	1195	1115
			80	1509.539	1551	2.72	0.75	103.38	0.990	1648	1568
bb45x5_2	225	400	20	268.637	302	12.28	5.25	80.26	0.984	302	302
			40	635.033	654	2.83	7.00	86.23	0.985	654	654
			60	1086.046	1122	3.22	1.00	95.390	0.978	1122	1122
			80	1595.648	1617	1.32	1.00	101.850	0.987	1690	1617
bb33x33_1	1089	2112	100	1435.891	1577	9.82	0.00	6633.150	-	1620	1562
			200	3053.369	3342	9.43	0.00	7079.540	-	3400	3303
			300	4730.450	5207	10.06	0.00	7755.700	-	5284	5112
			400	6565.646	7172	9.23	0.00	9011.410	-	7276	7070
			500	8627.560	9332	8.16	0.00	6710.190	-	9439	9204
bb33x33_2	1089	2112	100	1398.567	1547	10.58	0.00	5548.970	-	1577	1524
			200	3086.634	3379	9.46	0.00	5797.900	-	3468	3255
			300	4864.541	5296	8.86	0.00	6096.480	-	5555	5185
			400	6758.674	7339	8.58	0.00	6259.840	-	7638	7252
			500	8792.866	9510	8.15	0.00	6578.050	-	9828	9465
bb100x10_1	1000	1890	100	1472.408	1605	8.96	0.00	7369.16	-	1605	1601
			200	3224.183	3593	11.41	0.00	8881.59	-	3612	3520
bb100x10_2	1000	1890	100	1486.055	1668	12.18	0.00	9809.180	-	1668	1661
			200	3208.608	3637	13.34	0.00	9157.960	-	3695	3618

Tabela 4.3: Resultados Heurística Lagrangeana, Conjunto `d`, Parte I.

As principais contribuições do algoritmo proposto são:

- Novos limites inferiores para o k-ACM. Até o momento, poucos trabalhos haviam explorado técnicas para geração de limites inferiores para o k-ACM.
- Apresentação de novos melhores limites superiores. Fornecemos novos limites superiores para 8 de 104 instâncias disponíveis na literatura que foram testadas.

Apesar destas contribuições, em nossa avaliação existem muitas oportunidades de melhora, tanto dos limites primais e duais quanto dos tempos de computação requeridos pelo procedimento proposto.

Para obtermos soluções viáveis de melhor qualidade ao longo da Heurística Lagrangeana, pretendemos implementar um procedimento de Busca Local. Dada uma árvore T com k arestas, a idéia consiste em avaliar árvores com k arestas que difiram de T pela inclusão e remoção de um único vértice. Na tentativa de melhorar os limites duais, destacamos duas estratégias a serem investigadas:

- Avaliar computacionalmente a primeira opção de decomposição que destacamos na Seção 4.1 (que consiste em relaxar e dualizar as restrições de cardinalidade e acoplamento (3.14)-(3.18)).
- Implementar o Método do Volume Revisado (Bahiense et al. (2003)) para resolver os Problemas Duais associados às duas opções de decomposição destacadas.

Dados da instância				Heurística Lagrangeana							
	n	m	k	z_d	$\overline{z}$	gap (%)	Fix (%)	$t(s)$	$\frac{z_d}{LP_{MF}}$	$\overline{z}_{DP}$	MVC
g400-4-01	400	800	40	551.410	563	1.99	37.38	406.400	0.980	594	563
			80	1260.706	1304	3.41	1.75	465.760	0.968	1375	1304
			120	2048.206	2132	4.05	0.00	553.350	0.960	2157	2134
			160	2924.083	3065	4.79	0.00	526.290	0.955	3069	3062
			200	3954.386	4095	3.54	0.00	534.370	-	4099	4086
g400-4-05	400	800	40	657.112	670	1.83	24.75	352.290	0.984	729	673
			80	1424.153	1445	1.40	14.12	407.920	0.967	1488	1445
			120	2245.262	2293	2.09	0.88	469.120	0.980	2329	2293
			160	3109.961	3192	2.64	0.00	494.770	0.974	3242	3193
			200	4050.271	4156	2.59	0.00	529.760	0.975	4188	4156
g1000-4-01	1000	2000	100	1484.365	1527	2.83	2.85	9291.240	-	1596	1523
			200	3177.650	3308	4.09	0.00	10174.360	-	3431	3308
			300	5091.858	5328	4.63	0.00	11794.510	-	5423	5325
			400	7237.032	7590	4.86	0.00	10157.270	-	7653	7581
			500	9595.734	10079	5.03	0.00	10761.900	-	10118	10052
g1000-4-05	1000	2000	100	1572.676	1658	5.40	0.00	9487.400	-	1692	1652
			200	3463.566	3649	5.34	0.00	10246.420	-	3694	3620
			300	5500.633	5821	5.82	0.00	10263.630	-	5894	5801
			400	7753.856	8217	5.97	0.00	9962.520	-	8275	8206
			500	10217.080	10797	5.67	0.00	10432.050	-	10835	10793
steinc15	500	2500	50	205.005	208	0.98	83.44	1427.78	-	220	208
			100	474.209	481	1.27	75.60	1605.20	-	510	481
			150	781.961	802	2.56	60.16	1804.98	-	820	802
			200	1136.941	1182	3.96	33.76	2073.63	-	1196	1182
			250	1545.596	1625	5.11	0.72	2230.47	-	1632	1625
steinc5	500	625	50	760.839	772	1.45	26.40	465.31	0.991	810	772
			100	1692.139	1712	1.12	15.84	524.68	0.989	1724	1712
			150	2783.501	2874	3.23	0.00	619.55	-	2893	2865
			200	4136.068	4271	3.24	0.00	641.64	0.968	4301	4273
			250	5730.500	5945	3.73	0.00	712.38	-	5952	5945
steind15	1000	5000	100	433.589	454	4.61	65.48	20449.74	-	455	455
			200	960.288	1023	6.46	21.82	22201.96	-	1037	1018
			300	1586.317	1684	6.11	0.00	22561.36	-	1701	1674
			400	2317.123	2457	6.00	0.00	22794.72	-	2467	2446
			500	3149.740	3372	7.05	0.00	26530.49	-	3374	3365
steind5	1000	1250	100	1480.302	1503	1.49	13.44	7741.05	-	1623	1503
			200	3257.997	3452	5.95	0.00	8694.89	-	3591	3442
			300	5470.884	5836	6.67	0.00	7815.48	-	5925	5817
			400	8173.795	8690	6.31	0.00	8394.20	-	8742	8691
			500	11330.647	12074	6.56	0.00	9245.71	-	12101	12056
le450_15a	450	8168	45	58.012	59	0.00	96.41	7218.960	-	60	59
			135	224.567	226	0.45	94.82	14295.790	-	226	226
			225	461.809	471	1.95	86.28	16179.830	-	472	471
			405	1353.399	1388	2.51	54.71	13351.920	-	1388	1388

Tabela 4.4: Resultados Heurística Lagrangeana, Conjunto `d`, Parte II.

Dados da instância				Heurística Lagrangeana							
	n	m	k	z_d	$\overline{z}$	gap (%)	Fix (%)	$t(s)$	$\frac{z_d}{LP_{MF}}$	$\overline{z}_{DP}$	MVC
GRID_10_50_1	100	180	50	15441.722	16052	3.95	0.00	28.14	0.962	17195	-
GRID_10_50_2	100	180	50	13583.364	14026	3.25	0.00	24.44	0.968	15310	-
GRID_10_50_3	100	180	50	12698.690	13117	3.29	0.00	27.21	0.968	14983	-
GRID_10_50_4	100	180	50	13801.705	14283	3.49	0.00	27.19	0.969	16229	-
GRID_10_50_5	100	180	50	16689.657	17093	2.41	0.00	25.06	0.977	19494	-
GRID_15_112_1	225	420	112	37485.569	38322	2.23	0.00	150.44	0.978	42422	-
GRID_15_112_2	225	420	112	30989.623	32081	3.52	0.00	149.62	0.967	35704	-
GRID_15_112_3	225	420	112	30127.993	31321	3.96	0.00	150.46	0.962	33924	-
GRID_15_112_4	225	420	112	29614.887	30931	4.44	0.00	152.28	0.965	34653	-
GRID_15_112_5	225	420	112	28530.927	30063	5.37	0.00	149.83	0.955	34528	-
GRID_20_200_1	400	760	200	53017.607	55641	4.95	0.00	516.33	0.956	63281	-
GRID_20_200_2	400	760	200	52791.745	55073	4.32	0.00	469.21	0.962	62231	-
GRID_20_200_3	400	760	200	52010.669	54127	4.07	0.00	489.58	0.983	62070	-
GRID_20_200_4	400	760	200	51498.130	53761	4.39	0.00	500.62	0.962	61974	-
GRID_20_200_5	400	760	200	59510.341	62557	5.12	0.00	518.46	0.954	69023	-
GRID_30_450_1	900	1740	450	118144.291	125302	6.06	0.00	4541.19	-	143620	-
GRID_30_450_2	900	1740	450	116918.749	123152	5.33	0.00	4920.01	-	139015	-
GRID_30_450_3	900	1740	450	130425.407	136506	4.66	0.00	4150.01	-	153048	-
GRID_40_800_1	1600	3120	800	201778.857	214161	6.14	0.00	24554.24	-	247159	-
GRID_40_800_2	1600	3120	800	217371.600	231517	6.51	0.00	23186.57	-	261090	-
GRID_40_800_3	1600	3120	800	205028.649	220040	7.32	0.00	20636.56	-	254092	-
GRID_50_1250_1	2500	4900	1250	353852.168	372847	5.37	0.00	52594.87	-	414091	-
GRID_50_1250_2	2500	4900	1250	328600.848	348645	6.10	0.00	64231.10	-	394458	-
GRID_50_1250_3	2500	4900	1250	329157.216	350708	6.55	0.00	69692.09	-	400724	-

Tabela 4.5: Resultados Heurística Lagrangeana, Conjunto `NWG`.

Capítulo 5

Conclusões e Trabalhos Futuros

Nesta dissertação apresentamos um estudo do problema conhecido como *Árvore de Custo Mínimo com k arestas*, ou simplesmente, k-ACM. Embora o k-ACM tenha sido amplamente estudado na literatura, poucos trabalhos propuseram métodos de solução exata para o problema. Motivados por esta lacuna observada na literatura, neste trabalho apresentamos reformulações para k-ACM que utilizam um grafo orientado, obtido a partir do grafo de definição do problema. Duas reformulações foram propostas, ambas exploradas através de algoritmos exatos. A primeira delas é baseada em fluxos de múltiplas mercadorias, onde a eliminação de sub-circuitos é garantida através de restrições de balanço de fluxo. A segunda reformulação emprega as restrições de eliminação de sub-circuitos propostas por Miller et al. (1960) para garantir a conectividade das soluções do problema.

As reformulações apresentadas foram analisadas empiricamente, tanto no que diz respeito à qualidade de seus limites de Programação Linear, quanto no que tange o desempenho dos algoritmos *Branch-and-Bound* que utilizam Relaxações Lineares das mesmas. Experimentos computacionais conduzidos com instâncias difíceis da literatura confirmaram as nossas expectativas de que a reformulação baseada em múltiplas mercadorias é sistematicamente mais forte que aquela que emprega as restrições de Miller et al. (1960). Como os tempos computacionais necessários para avaliar a Relaxação Linear da reformulação Multi-fluxo são extremamente altos, propusemos um procedimento baseado em Relaxação Lagrangeana para tentar contornar esta dificuldade. Uma Heurística Lagrangeana foi também incorporada ao procedimento proposto.

O procedimento baseado em Relaxação Lagrangeana permitiu que os limites de Relaxação Linear associados à formulação de múltiplas mercadorias fossem estimados em tempos computacionais cerca de 2 ou 3 ordens inferiores. Apesar disto, os tempos de computação ainda são elevados. Os limites duais Lagrangeanos obtidos, na média, representam 98% dos valores que poderíamos, pelo menos teoricamente, obter. Visando melhorar a qualidade dos limites duais Lagrangeanos, pretendemos implementar o Algoritmo do Volume para resolver o Problema Dual Lagrangeano.

Como sub-produto desta dissertação, apresentamos 67 certificados de otimalidade e 16 novos melhores limites superiores para instâncias da literatura.

Apêndice A

O algoritmo de Programação Dinâmica para o k-ACM

Blum (2007b) propôs uma extensão do algoritmo de Fischetti et al. (1994) para tratar grafos com custos nos vértices e também nas arestas. Este algoritmo leva em consideração o fato de que o k-ACM pode ser resolvido em tempo polinomial quando o grafo de entrada do problema é uma árvore.

O algoritmo de Blum requer como entrada uma árvore roteada $T = (V_T, E_T)$ e uma máxima cardinalidade $k \leq |E_T|$. Antes de prosseguir com o algoritmo, vamos introduzir algumas notações que serão utilizadas. O vértice raiz da árvore de entrada T será chamado de v_{raiz}. Além disso, dada tal árvore roteada T, será usada a notação $T(v)$ para designar a sub-árvore roteada em $v \in V_T$. Note que $T(v)$ contém um nó $v' \in V$ se e somente se o caminho em T a partir de v_{raiz} para v' contém v. Além disso, o conjunto $E_{T(v)}$ irá designar o conjunto de arestas na sub-árvore roteada em v, cuja cardinalidade é $|E_{T(v)}|$.

Como é comum em problemas resolvidos através da técnica de Programação Dinâmica, é necessário definir sub-problemas a serem atacados. No caso do algoritmo de Blum, um sub-problema do problema original de encontrar uma k-ACM em T consiste em encontrar, para cada $v \in V$, l-ACMs, onde $0 \leq l \leq \min\{k, |E_{T(v)}|\}$ em $T(v)$. Dados v e l, denotamos o sub-problema correspondente pela tupla $(T(v), l)$. Para cada sub-problema, o algoritmo de Programação Dinâmica avalia e armazena os seguintes valores:

1. $f^-_{v,l}$: o valor da melhor l-ACM em $T(v)$ que **não contém** v.
2. $f^+_{v,l}$: o valor da melhor l-ACM em $T(v)$ que **contém** v.

O valor da função objetivo $f_{v,l}$ é igual a $\min\{f^-_{v,l}, f^+_{v,l}\}$, sendo que inicialmente todos os valores $f^-_{v,l}$ e $f^+_{v,l}$ são assumidos como ∞.

O funcionamento do algoritmo é baseado em *cores*: se um vértice está branco, então os seus sub-problemas ainda não foram todos resolvidos. Por outro lado, se um vértice está preto, então todos os seus sub-problemas já foram devidamente resolvidos. Obviamente, no início do algoritmo todos os vértices são coloridos de branco.

O algoritmo inicia sua operação resolvendo os sub-problemas para os vértices folhas, isto é, vértices que tem grau igual a 1. Para estes vértices, o sub-problema com 0 arestas é resolvido da seguinte maneira:

$$f^{-}_{v,0} = \infty \tag{A.1}$$

$$f^{+}_{v,0} = d_v \tag{A.2}$$

O Algoritmo A.1 apresenta uma visão geral da proposta de Blum. Inicialmente, são resolvidos os sub-problemas dos vértices folhas, e em seguida parte-se para os demais vértices, caracterizando assim uma estratégia de solução *bottom-up*.

Algoritmo de Blum(T, k)

1 **for** cada folha $v_f \in V_T$
2 **do** Resolva sub-problemas $T(v_f, 0)$
3 ▷ Ver equações (A.1) e (A.2)
4 Colora v_f de preto
5 **while** $\exists v \in T : v$ possui cor branca e possui apenas filhos pretos
6 **do**
7 Resolva sub-problemas $(T(v), l)$ para $l = 0 \ldots \min\{k, |E_{T(v)}|\}$
8 ▷ Veja equações (A.3) e (A.4)
9 Colora v de preto
10 **return** Os valores das melhores l-ACM em T $(0 \leq l \leq k)$

Figura A.1: O algoritmo estendido de Blum para a versão completa do k-ACM.

Por outro lado, os sub-problemas para os vértices internos são muito mais complexos de serem resolvidos do que os sub-problemas dos vértices folhas. Seja v um vértice interno que possua todos os filhos pintados de preto. Seja r o número de filhos de v, tal que possamos identificar os filhos de v por $v'_1, v'_2, \ldots, v'_r$. Para cada $l = 0, \ldots, \min\{k, |E_{T(v)}|\}$, os valores de $f^{-}_{v,l}$ são atualizados como segue:

$$f^{-}_{v,l} = \min\{f_{v'_i,l} | i = 1, \ldots r\} \tag{A.3}$$

A equação (A.3) nos diz que a melhor sub-árvore com l arestas sem o vértice v é dada pela melhor sub-árvore com l arestas dos filhos de v (tais sub-árvores por sua vez podem ou não englobar os filhos de v; note que $f_{v,l} = \min\{f^{-}_{v,l}, f^{+}_{v,l}\}$). Já os valores de $f^{+}_{v,l}$ são calculados de acordo com a equação (A.4):

$$f^{+}_{v,l} = d_v + \min\left\{\sum_{i=1}^{r}\left(\delta(\alpha_i) \times (c_{v,v'_i} + f^{+}_{v'_i,\alpha_i})\right) : \sum_{i=1}^{r}\alpha_i = l - r, \alpha_r \geq -1\right\} \tag{A.4}$$

onde $\delta(Z)$ é uma função que retorna 0 se o seu argumento é menor do que zero, e retorna 1

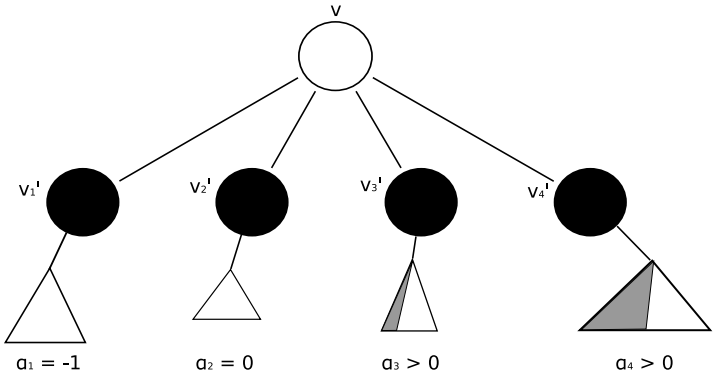

Figura A.2: Exemplo de atribuição de diferentes valores de α.

caso contrário. A atribuição $\alpha_i = -1$ para algum α_i indica que a sub-árvore correspondente ($T(v'_i)$) não contribui para o valor de $f^+_{v,l}$. Além disso, c_{v,v'_i} representa o custo da aresta entre v e seu i-ésimo filho.

A equação (A.4) determina a melhor combinação de sub-árvores menores que estão contidas nas sub-árvores $T(v'_i)$, $i = 1, \ldots, r$. Uma ilustração de diferentes valores de α é apresentada na Figura A (adaptada de Blum (2007b)). Neste exemplo, o vértice v tem quatro filhos, v'_1, v'_2, v'_3 e v'_4. Vamos assumir que buscamos pela melhor l-ACM em $T(v)$, e que os valores encontrados para os α foram os assinalados na figura, satisfazendo $\sum_{i=1}^{4} \alpha_i = l - 4$. O valor $\alpha_1 = -1$ implica que a sub-árvore $T(v'_1)$ não contribui para a l-ACM resultante. Já o valor $\alpha_2 = 0$ significa que $T(v'_2)$ contribui apenas com o seu vértice raiz v'_2. Por sua vez, as sub-árvores $T(v'_3)$ e $T(v'_4)$ contribuem com uma sub-árvore de tamanho igual ao valor de α_3 e α_4, respectivamente, sendo que tal tamanho é indicado pelo tom de cinza na figura. Adicionalmente, a l-ACM contém o vértice v e as arestas entre v e os seus filhos que possuem valor de α maior do que -1, que neste exemplo são v'_2, v'_3 e v'_4.

No caso em que $r = 1$, o mínimo da equação (A.4) é obtido com $\alpha_1 = 1$. No caso em que $r = 2$, é necessário checar todas as possíveis combinações de α_1 e α_2 tal que $\alpha_1 + \alpha_2 = l - 2$ para obter o tal mínimo, o que pode ser feito em $O(l)$ passos. Enretanto, quando $r > 2$, não é necessário checar todas as combinações de α que satisfazem $\sum_{i=1}^{r} \alpha_i = l - r$; se isto fosse necessário, então a complexidade do algoritmo seria $O(k^n)$, onde $n = |V|$. Existe uma estratégia passo a passo para resolver o problema, que leva a uma complexidade de $O(k^2 n)$ no pior caso.

Dado um nó branco v com todos os filhos pretos, inicialmente, para $l = 0, \ldots, \min\{k, |E_{T(v'_1)}| + |E_{T(v'_2)}| + 2\}$, a equação (A.4) é aplicada a exatamente dois filhos de v, no caso v'_1 e v'_2, o que significa que os outros filhos são desconsiderados momentaneamente. Obviamente, os valores obtidos $f^+_{v,l}$ não estão corretos. Então, na ordem $i = 3, \ldots, r$, estes valores são atualizados,

para $l = 0, \ldots, \min\{k, \sum_{j=1}^{i} |E_{T(v'_j)}| + i\}$, da seguinte maneira:

$$f_{v,l}^{+\text{novo}} = \min\left\{f_{v,\gamma}^{+} + \left(\delta(\beta_i) \times (c_{v,v'_i} + f_{v'_i,\beta_i}^{+})\right) : \gamma + \beta_i = l - 1, \gamma \geq 0, \beta_i \geq -1\right\} \quad \text{(A.5)}$$

Após o cálculo de todos os valores $f_{v,l}^{+\text{novo}}$ para um determinado i, os valores antigos $f_{v,l}^{+}$ são sobrescritos pelos novos valores $f_{v,l}^{+\text{novo}}$ correspondentes.

A Figura A.3 apresenta um grafo que será utilizado para exemplificar o funcionamento do algoritmo de Programação Dinâmica para uma *2*-ACM, enquanto a Tabela A.1 apresenta os valores computados. Inicialmente, são resolvidos os sub-problemas para os vértices folhas, C, D, E, para $l = 0$, que, de acordo com as equações (A.1) e (A.2), resultam em:

$$f_{C,0}^{-} = \infty, \;\; f_{D,0}^{-} = \infty, \;\; f_{E,0}^{-} = \infty$$

e:

$$f_{C,0}^{+} = d_C = 8, \;\; f_{D,0}^{+} = d_D = 2, \;\; f_{E,0}^{+} = d_E = 1$$

O primeiro vértice interno que possui apenas filhos coloridos de preto é o B, que possui $|E_{T(B)}| = 2$, e portanto é necessário resolver os sub-problemas para $l = 0, \ldots, 2$, resultando em:

- **Sub-problema para** $l = 0$
  - $f_{B,0}^{-}$: de acordo com a equação (A.3), deve-se escolher o valor do menor sub-problema de um filho de B com $l = 0$, que é igual a 1 (o que significa que a melhor sub-árvore $T(B)$ sem o vértice B e com apenas um vértice é formada isoladamente pelo vértice E).
  - $f_{B,0}^{+}$: como $r = 2$ (o número de filhos de B), tem-se que $l - r = -2$, e portanto existe apenas uma configuração de valores de α para os filhos de B, que é $\alpha_D = -1$ e $\alpha_E = -1$; portanto, nenhum dos dois filhos entra na sub-árvore para B com 0 arestas, e a solução é apenas o custo de B, que é 3.
- **Sub-problema para** $l = 1$
  - $f_{B,1}^{-}$: é fácil ver que não existem sub-árvores roteadas nos filhos de B com uma aresta, e portanto o valor para esta entrada na Tabela A.1 é igual a ∞.
  - $f_{B,1}^{+}$: tem-se que $l - r = -1$ e portanto existem duas maneiras de configurar os valores de α. A primeira delas é fazer $\alpha_D = 0$ e $\alpha_E = -1$. Esta configuração indica que será usado apenas a raiz da sub-árvore roteada em $T(D)$, o que gera um custo de:

$$3 + 10 + 2 = 15$$

Por outro lado, a outra configuração válida ($\alpha_D = -1$ e $\alpha_E = 0$) gera uma solução de custo:

$$3 + 1 + 1 = 5$$

Por este motivo, a árvore de uma aresta com a raiz B preferível é aquela que contém o vértice E, com custo 5.

- **Sub-problema para $l = 2$**
  - $f^-_{B,2}$: novamente, não existe sub-árvore formada pelos filhos de B com duas arestas, e portanto, $f^-_{B,2} = \infty$
  - $f^+_{B,2}$: por outro lado, com $l = 2$ e $l - r = 0$, existem três configurações possíveis para os valores de α: $\alpha_D = -1$ e $\alpha_E = 1$, $\alpha_D = 1$ e $\alpha_E = -1$ e, finalmente, $\alpha_D = 0$ e $\alpha_E = 0$. É fácil verificar que apenas a última combinação é capaz de gerar uma árvore roteada em B com duas arestas, e portanto é a única que não gera custo ∞. De fato, o custo dessa configução é: $3 + 10 + 2 + 1 + 1 = 17$.

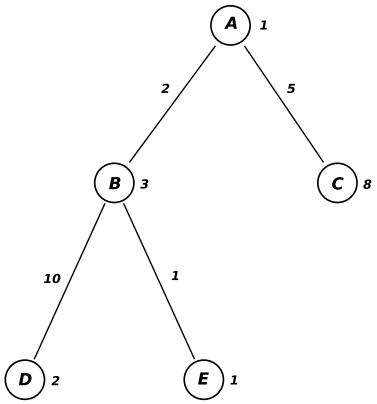

Figura A.3: Exemplo para o algoritmo de Programação Dinâmica.

Finalmente, o último vértice a ser investigado é o A, que possui uma sub-árvore roteada com 4 arestas. Portanto, deverão ser examinados os sub-problemas para $l = 0, \ldots, \min\{2, 4\} = 0, \ldots, 2$.

Para $l = 0$, $f^-_{A,0}$ é igual a $\min\{f_{B,0}, f_{C,0}\} = \min\{1, 8\} = 1$. Por outro lado, $f^+_{A,0} = 1$, pois $l - r = -2$ significa que $\alpha_B = -1$ e $\alpha_C = -1$ e portanto ambos os filhos não fazem parte da solução deste sub-problema.

Por outro lado, para $l = 1$, $f^-_{A,1} = \min\{f_{B,1}, f_{C,1}\} = \min\{5, \infty\} = 5$, e portanto a melhor sub-árvore que não contém A dentre as sub-árvores de $T(A)$ tem custo 5. Ainda, $f^+_{A,1} = 6$, o que corresponde a escolher a aresta (A, B), pois a escolha da aresta (A, C) gera um custo de 14.

Finalmente, $f^-_{A,2} = \min\{f_{B,2}, f_{C,2}\} = 17$. Esta solução equivale à árvore roteada em B que contém ainda os vértices D e E. Para finalizar, o problema $f^+_{A,2}$ apresenta três possibilidades de configuração para os valores de α, mas é fácil verificar que a melhor combinação é $\alpha_B = 1$ e $\alpha_D = -1$, que irá gerar a árvore com A, B e E de custo 8, que é a solução ótima para o 2-ACM em questão.

V	$f^-_{v,l}$			$f^+_{v,l}$			$f_{v,l}$		
	0	1	2	0	1	2	0	1	2
A	1	5	17	1	6	8	1	5	8
B	1	∞	∞	3	5	17	1	5	17
C	∞			8			8		
D	∞			2			2		
E	∞			1			1		

Tabela A.1: Tabela preenchida com os valores para a Figura A.3. As entradas vazias valem ∞ e não foram preenchidas para facilitar a visualização do exemplo.

Referências Bibliográficas

Ahuja, R. K.; Magnanti, T. L. e Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Arora, S. e Karakostas, G. (2006). A $2+\epsilon$ Approximation Algorithm for the k-MST Problem. *Mathematical Programming*, A(107):491–504.

Bahiense, L.; Barahona, F. e Porto, P. (September 2003). Solving Steiner Tree Problems in Graphs with Lagrangian Relaxation. *Journal of Combinatorial Optimization*, 7:259–282(24).

Barahona, F. e Anbil, R. (2000). "The volume algorithm: producing primal solutions with a subgradient method'. *Mathematical Programming*, 87:385–399.

Beasley, J. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Blesa, M. e Xhafa, F. (2000). "A C++ Implementation of Tabu Search for v-Cardinality Tree Problem based on Generic Programming and Component reuse". In *Net Objective Days 2000*, pp. 648–652.

Blum, C. (2007a). KCTLib: The Edge-Weighted K-Cardinality Tree Problem Test Set Archive.

Blum, C. (2007b). Revisiting Dynamic Programming for Finding Optimal subtrees in Trees. *European Journal of Operational Research*, 177:102–115.

Blum, C. e Blesa, M. J. (2005). New Metaheuristic Approaches for the Edge-weighted k-Cardinality Tree Problem. *Computers and Operations Research*, 32(6):1355–1377.

Blum, C. e Ehrgott, M. (2003). Local Search Algorithms for the k-Cardinality Tree Problem. *Discrete Applied Mathematics*, 128(2-3):511–540.

Borůvka, O. (1926a). O Jistém Problému Minimálním. *Práce Moravské Přírodovědecké Spoločnosti*, 3:37–58.

Borůvka, O. (1926b). Příspěvek k Řešení otázky ekonomické stavby elektrovodných sítí. *Elektronický Obzor*, 15:153–154.

Brimberg, J.; Urosevic, D. e Mladenovic, N. (2006). Variable Neighborhood Search for the Vertex Weighted k-Cardinality Tree Problem. *European Journal of Operational Research*, 171:74–84.

Bruglieri, M.; Ehrgott, M.; Hamacher, H. e Maffioli, F. (2006). An Annotated Bibliography of Combinatorial Optimization Problems with Fixed Cardinality constraints. *Discrete Applied Mathematics*, 154:1344–1357.

Cheung, S. e Kumar, A. (1994). Efficient Quorumcast Routing Algorithms. In *INFOCOM '94: Proceedings of the IEEE Conference on Computer Communications*, pp. 840–847, Piscataway, NJ, USA. IEEE Society Press.

Chopra, S.; Gorres, E. e Rao, M. (1992). Solving the Steiner Tree Problem on a Graph Using Branch and cut. *Orsa Journal on Computing*, 4(3):320–335.

Coimbra, J. M. M. (1995). Limites Inferiores e Limites Superiores para o Problema da q-Árvore de Custo Mínimo. Master's thesis, Universidade de Lisboa.

Crowder, H. (1976). Computational improvements for subgradient optimization. In *Symposia Mathematica, Vol. XIX*, pp. 357–372, London.

Dantzig, G.; Fulkerson, R. e Johnson, S. (1954). Solution of a Large-Scale Traveling-Salesman Problem. *Operations Research*, 2(4):393–410.

Desrochers, M. e Laporte, G. (1991). "Improvements and Extensions to the Miller-Tucker-Zemlin Subtour Elimination Constraints". *Operations Research Letters*, 10:27–36.

Ehrgott, M. e Freitag, J. (1996). K-tree / K-subgraph: A Program Package for Minimal Weighted k-Cardinality Trees and Subgraphs. *European Journal of Operational Research*, 93(1):214–225.

Ehrgott, M.; Freitag, J.; Hamacher, H. e Maffioli, F. (1997). Heuristics for the k-Cardinality Tree and Subgraph Problems. *Asia-Pacific Journal of Operations Research*, 14(1):87–114.

EXAME, P. (2006). O Matemático das Empresas. Acessado em 20/01/2008.

Fischetti, M.; Hamacher, H.; Jornsten, K. e Maffioli, F. (1994). Weighted k-Cardinality Trees: Complexity and polyhedral structure. *Networks*, 24:11–21.

Garg, N. (1996). A 3-approximation for the Minimum tree Spanning k Vertices. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, p. 302, Washington, DC, USA. IEEE Computer Society.

Garg, N. e Hochbaum, D. (1997). An o($\log k$) Approximation Algorithm for the k Minimum Spanning Tree Problem in the Plane. *Algorithmica*, 18(1):111–121.

Gendron, B. e Crainic, T. (1994). Relaxations for multicommodity capacitated network desing problems. Technical Report CRT-965, Centre de Recherche sur les Transports, Université de Monstreal, Canada.

Glover, F. e Laguna, F. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

Goemans, M. X. (1994). The Steiner Tree Polytope and Related Polyhedra. *Mathematical Programming*, A(63):157–182.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

Gouveia, L. (1995). Using the miller-tucker-zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & OR*, 22(9):959–970.

Hamacher, H. W. e Joernsten, K. (1993). Optimal Relinquishment According to the Norwegian Petrol Law: A Combinatorial Optimization Approach. *Norwegian School of Economics and Business Administration – Technical Report*, 7/93.

Hansen, P. e Mladenovic, N. (2001). Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 130(3):449–467.

Held, M. e Karp, R. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162.

Held, M.; Wolfe, P. e Crowder, H. (1974). "Validation of Subgradient Optimization". *Mathematical Programming*, 6:62–88.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA.

Holmberg, K. e Yuan, D. (1998). A Lagrangian Heuristic Based Branch-and-bound Approach for the Capacitated Network Design Problem. *Operations Research*, 48(3):461–481.

ILOG (2006). Ilog cplex. source: http://www.ilog.com/products/cplex/.

ILOG (2008). Ilog Customers. source: http://www.ilog.com/success. Acessado em 20/01/2008.

INFORMS (2008). Operations Research: The Science of Better. source: http://www.scienceofbetter.org/. Acessado em 20/01/2008.

Jornsten, K. e Lokketangen, A. (1997). Tabu Search for Weighted k-Cardinality Trees. *Asia-Pacific Journal of Operational Research*, 14(1):9–26.

Kataoka, S.; Araki, N. e Yamada, T. (2000). Upper and Lower Bounding Procedures for Minimum Rooted k-subtree Problem. *European Journal of Operational Research*, 122:561–569.

Koch, T. e Martin, A. (1998). Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232.

Kruskal, J. B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.

Lemarechal, C. (1974). "An Algorithm for Minimizing Convex Functions". In *IFIP'74 Congress*, pp. 552–556, Noth Holland.

Lucena, A. (1991). Tight Bounds for the Steiner Problem in Graphs. In *XXX SOBRAPO – XXIII Joint International Meeting*, Rio de Janeiro, Brasil.

Maculan, N. (1987). The Steiner Tree Problem in Graphs. *Annals of Discrete Mathematics*, 31:185–212.

Margot, F.; Prodon, A. e Liebling, M. (1994). The Polyhedron on 2-tree. *Mathematical Programming*, A(63):183–192.

Miller, C.; Tucker, A. e Zemlin, R. (1960). Integer Programming Formulations and Travelling Salesman Problems. *Journal of the Association for Computing Machinery*, 7:326–329.

Nemhauser, G. (2007). Scheduling on Demand Passenger Air Service. In *LAGOS'07: IV Latin American Algorithms, Graphs and Optimization Symposium*, Puerto Varas, Chile.

Nemhauser, G. e Wolsey, L. (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley.

Philpott, A. e Wormald, W. (1997). On the Optimal Extraction of ore from an open-cast mine. *University of Auckland – Technical Report*.

Prim, R. C. (1957). Shortest Connection Networks and Some Generalizations. *The Bell System Technical Journal*, 3:1389–1401.

T. H. Cormen, C. E. Leiserson, R. L. R. e. C. S. (2001). *Introduction to Algorithms, 2ed.* McGraw-Hill.

Wolsey, L. (1998). *Integer Programming*. Wiley.

Ziviani, N. (2004). *Projeto de Algoritmos Com Implementações em Pascal e C*. Pioneira Thomson Learning.