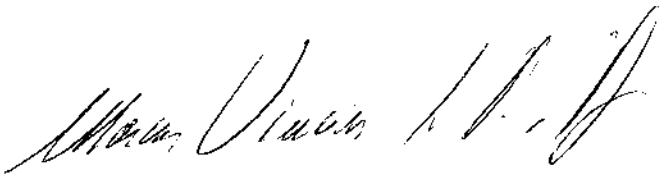
Problemas de Classificação com Restrições de Conexidade Flexibilizadas

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Eduardo Uchoa Barboza e aprovada pela Banca Examinadora.

Campinas, 17 de junho de 1997.



Marcus Vinícius S. Poggi de AragãoDepartamento de Informática - PUC-R
()

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

UNICAMP
BIBLIOTECA CENTRAL

UNIDADE BC
N.º CHAMADA: T UNICAMP
B234p
V. Ex.
TOMBO BC/ 32127
PROC. 281/97
C [ ] D [X]
PREÇO R$ 11,00
DATA 18/11/97
N.º CPD

CM-00102278-2

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Barboza, Eduardo Uchoa

B234p Problemas de classificação com restrições de conexidade flexibilizadas / Eduardo Uchoa Barboza -- Campinas, [S.P. :s.n.], 1997.

Orientador : Marcus Vinícius S. Poggi de Aragão

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Análise de aglomerado. 2. Programação inteira. 3. Otimização combinatória. 4. Pesquisa operacional. I. Poggi de Aragão, Marcus Vinícius Soledade. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Instituto de Computação
Universidade Estadual de Campinas

Problemas de Classificação com Restrições de Conexidade Flexibilizadas

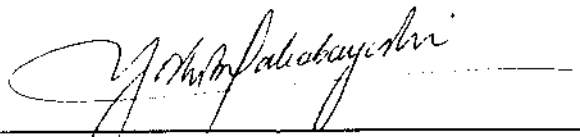
Eduardo Uchoa Barboza

Junho de 1997

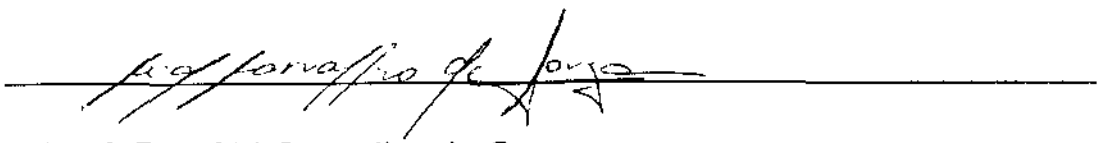
Banca Examinadora:

- Marcus Vinícius S. Poggi de Aragão
  Departamento de Informática - PUC-Rio (Orientador) ()
- Yoshiko Wakabayashi
  Instituto de Matemática e Estatística - USP
- Cid C. de Souza
  Instituto de Computação - UNICAMP
- Pedro J. de Rezende
  Instituto de Computação - UNICAMP (suplente)

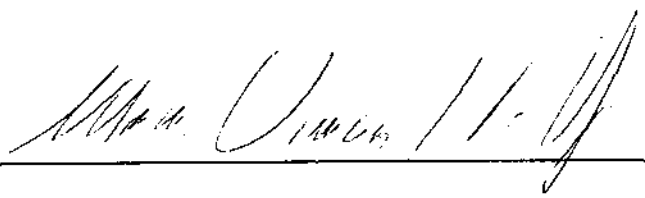
Tese de Mestrado defendida e aprovada em 31 de março de 1997 pela Banca Examinadora composta pelos Professores Doutores



Profª. Drª. Yoshiko Wakabayashi



Prof. Dr. Cid Carvalho de Souza



Prof. Dr. Marcus Vinícius Soledade Poggi de Aragão

Agradecimentos

Ao professor Marcus Poggi pela orientação. Acho que ninguém mais teria tanto entusiasmo em trabalhar num tema, que a princípio, parecia tão aberto e indefinido.

Aos professores do Instituto de Computação, em especial aos professores Cid C. de Souza e Ricardo Dahab, pelo grande apoio prestado durante o mestrado.

Aos professores Claúdio L. Lucchesi e Maria Cecília Baranauskas, pelos incentivos à minha carreira científica, dados desde a época da graduação.

Aos meus colegas do mestrado, em particular a Maria do Socorro, Aminadab, Claúdio e Elder, não apenas pelo companheirismo, mas pelas proveitosas discussões.

Aos meus amigos de Campinas, pelos bons momentos passados nesses dois anos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq e a Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP pelo apoio financeiro.

À minha família, por tudo de bom que me foi propiciado em toda a minha vida e pelo constante incentivo nesse trabalho.

À Andrea

À India

"A Alegria é a Prova dos Nove."

(*Oswald de Andrade*)

Resumo

A classificação de dados consiste em separar um conjunto de objetos, descritos por um conjunto de dados, em classes, de forma a que objetos na mesma classe sejam semelhantes entre si. A classificação freqüentemente é usada como uma ferramenta de pesquisa científica. Os objetivos de uma classificação podem diferir de acordo com as necessidades e com a origem dos dados sobre os objetos a classificar.

Em alguns contextos existe interesse em associar os objetos aos vértices de um grafo, de forma que a semelhança entre os objetos esteja relacionada a proximidade nesse grafo. Os métodos existentes, para aplicações nestes contextos, obrigam cada classe a formar um único componente conexo dentro do grafo. Chamamos essa abordagem de conexidade estrita e propomos a idéia de classificação com conexidade flexibilizada, ou seja a concepção de métodos que permitam a um usuário especificar o número de componentes de cada classe no grafo e mostramos porque essa flexibilização é desejavel.

Em seguida, estudamos a resolução de um problema computacional resultante da flexibilização da conexidade, o Problema da Atribuição γ-Conexa (PAgC). Demonstramos a NP-completude desse problema e apontamos alguns casos polinomiais. Então, concentramo-nos no estudo de diferentes formas para modelar matematicamente a conexidade flexibilizada. Os resultados obtidos podem ser naturalmente aplicados a outros problemas onde tal conexidade é necessaria.

Finalmente, propomos dois algoritmos para resolver (PAgC). Um baseado na técnica de *branch-and-bound* e outro na de *branch-and-price*. Uma comparação dos resultados obtidos por cada técnica é apresentada na seqüência. Um estudo de classes de desigualdades válidas, capazes de melhorar substancialmente ambos os algoritmos, conclui a dissertação.

Abstract

Classification of objects amounts to defining, say, K clusters on a set of N objects such that object in a same cluster are alike. Classification is often used as a tool for scientific research.

Some important contexts of objects classification use enhanced methods where the objects are associated to vertices of a graph. Proximity between two objects in this graph usually means that the dissimilarity between them is small. The methods applied in such contexts require each cluster to define a single connected component on this graph. We call this *strict connectivity* approach and we propose the idea of *flexible connectivity*. This suggests the development of methods where the user may specify the number of connected components each cluster may form on the graph. We present reasons for this flexibilization.

Next, we study the computational problem derived from the flexible connectivity, the γ-Connected Assignment Problem (PAgC). We show this problem to be NP-complete and we point out some polynomial cases. Then, we concentrate our effort on deriving formulations for considering this flexible connectivity. Clearly, the conceived models can be applied to other problems where connectivity constraints are to be considered.

Finally, we propose two algorithms to solve PAgC. One based on the *branch-and-cut* technique and other on the *branch-and-price* technique. A comparison between the results obtained with each technique is presented. We conclude this work studying classes of valid inequalities capable of improving the efficiency of both algorithms.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O problema geral de **classificação** [1] pode ser definido como: Dado um conjunto de N objetos e uma medida da dissimilaridade entre cada par desses objetos, particionar esse conjunto em K classes de modo que objetos semelhantes fiquem na mesma classe e objetos pouco semelhantes em classes diferentes.

Uma medida de **dissimilaridade** é um número positivo proporcional à "distância" entre dois objetos; no sentido que quanto maior o número, mais distintos são os objetos. A dissimilaridade entre dois objetos idênticos deve ser sempre zero. Alguns autores usam também medidas de similaridade, que são números positivos proporcionais à "proximidade" entre dois objetos. Nesse texto usaremos apenas dissimilaridades.

Uma instância do problema pode ser descrita através de uma matriz $N \times N$, simétrica e com zeros em sua diagonal, cujos elementos são as dissimilaridades entre os pares de objetos. Outra maneira de ver essa matriz é como um grafo não-direcionado completo G, com N vértices correspondendo aos objetos e com pesos nas arestas correspondendo às dissimilaridades. Deve-se especificar o número K, $1 < K < N$, de classes desejadas. Uma solução é sempre uma K-partição $P = \{C_1, C_2, ... C_K\}$, dos N objetos, onde cada classe C_i é não-vazia.

Um **método** de classificação é um critério que busca capturar e formalizar o conceito genérico de classificação dado acima, definindo as propriedades que a partição buscada deve satisfazer. Por exemplo: achar uma partição que maximize a menor dissimilaridade entre dois objetos em classes diferentes. A mesma instância pode ser classificada de acordo com diferentes métodos, que podem levar a diferentes partições. Assim, cada método, que pode ser mais ou menos adequado, de acordo com a aplicação, define um problema particular de otimização. Para que um método seja aplicável na prática, devem ser desenvolvidos algoritmos capazes de resolver esse problema computacional.

O problema de classificação é dito **irrestrito** quando qualquer uma das K-partições, em princípio (antes de se considerar as dissimilaridades), é válida. Ao contrário, ele é

[1] Freqüentemente chamado na literatura em inglês de *clustering*.

restrito quando algumas das K-partições já são descartadas à priori.

Alguns tipos usuais de restrições:

a) Cardinalidade - Deve haver um número mínimo ou máximo de objetos na mesma classe.

b) Alguns subconjuntos de objetos são obrigados a (ou proibidos de) pertencerem à mesma classe.

c) **Conexidade** - Define-se um grafo não-direcionado adicional H com N vértices, em que cada objeto é identificado com um vértice de H. Uma partição só é válida se os objetos de cada classe formarem um componente conexo em H. A restrição de conexidade [2] costuma levar a problemas computacionalmente muito difíceis.

Nesta dissertação estudamos problemas de classificação com conexidade, mas não apenas da forma usual definida em (c), que chamaremos de **estrita** (onde cada classe deve obrigatoriamente induzir um único componente conexo). Queremos tratar a conexidade de uma forma **flexibilizada**. Isso quer dizer que trabalharemos com métodos que permitam que um usuário, através de parâmetros, tenha controle sobre o número de componentes conexos induzidos por cada classe.

O nossos objetivos são:

- Definir novos métodos de classificação com restrições de conexidade flexibilizadas.
- Estudar a resolução computacional dos problemas induzidos por esses métodos através de técnicas de otimização combinatória, em especial de programação linear inteira. Não descartamos o uso de heurísticas para obter boas soluções aproximadas. mas nos concentramos no estudo de algoritmos exatos.

1.1 Roteiro da Dissertação

Nesta dissertação, não assumimos que o leitor tenha qualquer conhecimento prévio sobre classificação. Assim, procuramos apresentar da forma mais completa possível, todos os conceitos dessa área que julgamos necessários ao entendimento e justificativa deste trabalho. Por outro lado, assumimos familiaridade com os conceitos básicos de complexidade computacional, teoria de grafos, combinatória poliédrica e programação linear inteira, em particular de algoritmos de *branch-and-bound*. Nesses casos, nos limitamos a referenciar outros trabalhos sempre que preciso.

[2] Na literatura (Murtagh (1985) [28], por exemplo), isso às vezes aparece como restrição de contiguidade. Isso não é o mais correto, pois "contiguidade", sinônimo de "adjacência", define uma relação entre dois objetos. Já "conexidade", define uma relação entre n objetos, o que é o caso.

O capítulo 2 contém uma introdução à área de classificação, tanto a irrestrita quanto a com restrição de conexidade. Primeiro, tentamos mostrar que métodos com restrições de conexidade flexibilizadas são altamente recomendados em um grande número de aplicações práticas da classificação. Em seguida, fazemos um breve estudo crítico de alguns dos métodos de classificação mais utilizados hoje em dia. Esse estudo, serve tanto para apresentar trabalhos anteriores nessa área, quanto para deduzir as características que os nossos novos métodos devem satisfazer a fim de se adequar àquelas aplicações. A conclusão do capítulo é que seria interessante definir os nossos métodos como uma extensão do já tradicional método irrestrito do K-medóide.

Iniciamos o capítulo 3 descrevendo como será feita essa extensão do K-medóide. Em seguida, mostramos várias possíveis variantes da idéia da conexidade flexibilizada. Cada uma dessas variantes define um novo método, que por sua vez, gera um novo problema de otimização.

O capítulo 4, o mais importante da dissertação, apresenta um estudo sobre a resolução computacional desses novos problemas por técnicas de programação linear inteira.

Finalmente, no capítulo 5, fazemos breves comentários sobre os resultados alcançados, e indicamos possíveis trabalhos futuros a partir destes.

Capítulo 2

Classificação: Aplicações e Métodos

2.1 Aplicações da Classificação

A aplicação clássica da classificação é como uma ferramenta de pesquisa. Através dela tenta-se encontrar as "classes naturais" de um conjunto de objetos a partir de alguns de seus atributos.

Exemplo 2.1 *Um pesquisador está estudando características econômicas e sociais dos diversos países do mundo. Para tal, ele montou uma tabela com um certo número m de atributos para cada país.*

país	renda percapta	expectativa de vida	analfabetismo	...
Brasil	2900	67	15%	
Hungria	3600	71	2%	
Ruanda	300	45	40%	

Tabela 2.1: Exemplo de tabela de atributos

A partir dessa tabela ele pode estimar a dissimilaridade entre cada par de países induzida por esses atributos. Por exemplo, calculando a distância euclidiana (em m dimensões) entre esses atributos. É conveniente normalizar os valores de cada coluna, para que todas tenham a mesma média e desvio-padrão antes de fazer esse cálculo.

Agora, o pesquisador pode usar um método de classificação para ajudá-lo a encontrar grupos de países semelhantes. Nesse caso, ele terá que testar diversos valores de K, a fim de encontrar os agrupamentos mais significativos.

Existem muitos livros (Andberg (1973) [1], Späth (1980) [39], Gordon (1981) [14], Kaufman e Rousseeuw (1990) [22], Everitt (1993) [10], por exemplo) apenas sobre esse

uso da classificação como ferramenta de pesquisa; tratando da escolha dos atributos dos objetos, de formas de estimar a dissimilaridade entre eles, dos métodos a serem usados, da escolha de K, e finalmente da validação e interpretação dos resultados.

Quando, como no exemplo acima, os objetos são descritos por uma tabela de m atributos e a dissimilaridade é obtida por uma métrica sobre esses atributos, é comum interpretar cada objeto como um ponto num espaço de m dimensões. Nesse caso, a classificação pode ser vista como o problema de identificar **estruturas** desse espaço, que são regiões com alta densidade de objetos separadas por regiões de baixa densidade. Na figura 2.1 temos uma ilustração de um problema em duas dimensões com a dissimilaridade calculada pela distância euclidiana entre os pontos. Por sinal, nesses casos não existe melhor "classificador" do que o olho humano. Infelizmente, para dimensões maiores ou outros modos de calcular as dissimilaridades, temos que recorrer a técnicas mais sofisticadas e menos eficazes.

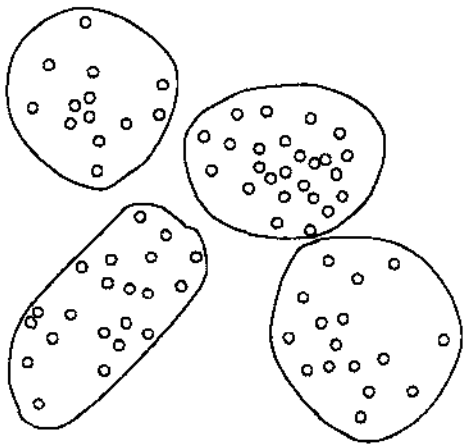


Figura 2.1: Uma classificação em duas dimensões ($N = 76$, $K = 4$)

O uso de técnicas sistemáticas para classificação de objetos, como ferramenta de pesquisa, tem uma história interessante (Andberg [1]).

Apesar dessa idéia ter sido esboçada por alguns autores, principalmente biólogos e naturalistas, desde o século XVIII; a classificação realmente floresceu, mais ou menos na mesma época e de forma independente em diversas áreas de pesquisa, nas décadas de 50 e 60. O intercâmbio entre estas áreas era limitado, conseqüentemente, mesmo a nomenclatura era particular a cada disciplina. Na biologia a classificação ficou conhecida como taxonomia numérica, em ciências sociais como tipologia, em reconhecimento de padrões e IA como um tipo de aprendizado sem supervisão, em linguística como *clumping*, em geografia como regionalização, em antropologia como serialização. Isso resultou em fatos curiosos. As mesmas idéias eram redescobertas seguidamente em diversos contextos. Por outro lado, importantes resultados computacionais ficaram anos esquecidos em revistas de biologia.

A partir da década de 70, houve um esforço em catalogar e unificar a teoria acerca dessas técnicas. Esse trabalho geralmente foi feito do ponto vista da estatística. Nessa época também descobriu-se que muitos outros tipos de problemas, nas mais diversas áreas, podem ser modelados como problemas de classificação. Essas novas aplicações vão desde projetos de circuitos VLSI a problemas teóricos em partição de grafos.

Hoje, a classificação é uma disciplina bem estabelecida, com uma publicação específica (*Journal of Classification*).

Independente do contexto em que é usada a classificação, cabe ressaltar aqui uma diferença fundamental entre dois tipos de aplicações. Chamaremos de **tipo 1** a classificação que é usada para se encontrar estruturas que possivelmente ou presumivelmente já existem entre os objetos, como geralmente acontece quando ela é usada como ferramenta de pesquisa. O exemplo 2.1 é uma aplicação de tipo 1. Nas aplicações de **tipo 2**, a classificação é usada para impor uma certa estrutura aos objetos.

Exemplo 2.2 *Um grupo de 140 alunos da Unicamp irá ao Maranhão para participar do congresso da SBPC; em 3 ônibus de 50 lugares, cedidos pela reitoria. O organizador da viagem, querendo minimizar os atritos decorrentes de 5 dias de convivência forçada dentro dos ônibus, resolveu fazer o seguinte:*

- Distribuiu um questionário entre os 140 alunos; com perguntas como fumante/não-fumante, tipo de música preferida, etc.

- Com base nas respostas, montou uma matriz com as dissimilaridades (de hábitos e "gostos") entre cada par de alunos.

A partir daí, pode-se usar um método de classificação para separar os alunos em 3 classes, de modo que alunos com hábitos parecidos fiquem no mesmo ônibus e alunos com hábitos muito distintos em ônibus diferentes. Entretanto, diferentemente do exemplo 1, aqui existe uma restrição: cada classe pode ter no máximo 50 objetos.

Esse exemplo é uma aplicação de tipo 2. O problema principal que o organizador da viagem quer resolver, não é conhecer os tipos de hábitos dos alunos (estruturas naturais); mas saber como colocá-los nos 3 ônibus (uma estrutura imposta) da melhor maneira possível. Nada impede que a solução ótima desse segundo problema, seja uma péssima solução para o primeiro.

Os autores da área muitas vezes não explicitam a divisão entre esses dois tipos de classificação. Não estar atento a isso pode levar a escolha de métodos inadequados à uma aplicação e até induzir uma interpretação equivocada dos resultados.

Pode parecer à primeira vista que a classificação com restrições só é usada em aplicações do tipo 2. O que não é verdade. Numa aplicação de tipo 1, uma restrição pode representar uma suposição sobre qual deve ser a verdadeira estrutura dos objetos, ou, uma informação adicional que queremos passar ao método de classificação. Por exemplo, consideremos a restrição que cada classe deve ter pelo menos 4 objetos. Pode ser que

essa restrição represente a suposição que as classes significativas não são tão pequenas. Ou ainda, pode representar a informação adicional que uma falsa pequena classe pode aparecer devido a ruídos nos dados e, portanto, deve ser desconsiderada.

Uma boa indicação do tipo de uma certa aplicação: o valor de K geralmente é conhecido em aplicações do tipo 2. Ao contrário, nas de tipo 1, inicialmente sempre há dúvidas sobre o melhor valor para K.

Exemplo 2.3 *(Bourjolly, Laporte e Rousseau [5]) No Canadá vigora o sistema de voto distrital. Isso significa que o país é particionado em K regiões chamadas distritos, sendo que cada distrito escolhe um deputado por eleição majoritária. A determinação dos distritos é refeita periodicamente por uma comissão neutra da qual participam especialistas em pesquisa operacional.*

Um distrito é formado reunindo-se um certo número de unidades políticas menores fixas, que para facilitar chamaremos de municípios.

Os municípios têm certas características históricas, econômicas e culturais. A determinação dos distritos é tratada como um problema de classificação, porque deseja-se que os municípios que compõem um distrito tenham uma certa homogeneidade de acordo com essas características.

Os distritos devem ter aproximadamente o mesmo número de eleitores e ser geograficamente contínuos.

Temos então uma restrição de conexidade que pode ser interpretada como: definido o grafo H planar com um vértice para cada município e uma aresta entre cada par de municípios que fazem divisa, cada distrito deve formar um componente conexo em H. Esta é uma aplicação do tipo 2 com restrição de conexidade estrita, pois pela natureza do problema os distritos obrigatoriamente devem ser contínuos.

Exemplo 2.4 *Uma companhia de petróleo está pesquisando a composição do subsolo profundo de uma certa região da Amazônia. Essa região foi dividida num certo número de áreas menores e em cada uma delas foi feito um estudo sísmico. Deseja-se classificar as áreas de acordo com os dados desse estudo. O problema é que como esses dados são inerentemente muito imprecisos e sujeitos a erros, dificilmente um método normal de classificação será capaz de identificar as estruturas corretas, mesmo que elas existam. Por outro lado, sabe-se por experiência anterior, que áreas com composição semelhante têm uma grande tendência de se agruparem em blocos conexos. Nesse caso, gostaríamos de ter um método que soubesse usar essa informação adicional, de forma a melhorar a qualidade da classificação obtida.*

Vários autores (os citados em Murtagh [28], Maravale e Simeone [27], Hansen et al. [15]) usam métodos com restrição de conexidade estrita para tratar esse tipo de problema.

Dessa forma, a restrição de conexidade serviria como um modo implícito de corrigir dissimilaridades imprecisas, com o objetivo de obter uma classificação mais correta e fiel à realidade.

É importante notar que não apenas casos como o do exemplo 2.4, em que a dificuldade de levantar os dados é extrema, levam a dissimilaridades pouco precisas. Isso acontece na maioria dos casos reais, devido a toda a sorte de problemas práticos que ocorrem na estimação das dissimilaridades. Esses casos poderiam eventualmente se beneficiar do uso de métodos que incorporem informações adicionais através de restrições de conexidade.

A questão que colocamos, é que em quase todas as aplicações, mesmo que seja razoável a suposição de que as áreas semelhantes costumam se agrupar em blocos conexos; é pouco razoável supor que elas obrigatoriamente devam se agrupar em um único bloco. Assim, insistir no uso da conexidade estrita provavelmente irá levar a uma classificação que não refletirá as estruturas naturais, mas uma estrutura artificialmente imposta, o que é inaceitável numa aplicação de tipo 1, como a do exemplo 2.4.

Parece claro que seria muito interessante termos métodos que permitam a flexibilização da restrição de conexidade. Ou seja, métodos que permitam que o usuário especifique, através de parâmetros, o grau de conexidade desejado. Seria desejável que este pudesse variar desde a conexidade estrita, nos casos em que se acredita haver fortíssima tendência dos objetos semelhantes de se agruparem em um único componente conexo; até uma conexidade fraca, permitindo um número relativamente grande de componentes por classe, quando se acredita haver apenas uma leve tendência dos objetos semelhantes a se agruparem em componentes conexos.

Exemplo 2.5 *Um pesquisador levantou dados sócio-econômicos sobre cada um dos 574 municípios do estado de São Paulo. Usando um método de classificação, ele achou uma boa divisão em 6 classes. Ele deseja apresentar seus resultados de forma sucinta num mapa, de forma que municípios da mesma classe apareçam com a mesma cor. Só que o mapa colorido dessa maneira ficou muito confuso e de difícil interpretação, pois as cores estavam fragmentadas em pequenos pedaços e espalhadas por todo o mapa. Na verdade, o pesquisador gostaria de ter uma partição mais "grossa", não tão precisa (alguns municípios não estariam na classe mais correta), mas que permitisse localizar facilmente as classes no mapa, de forma a caracterizar claramente as grandes regiões sócio-econômicas do estado.*

Em princípio, esse problema poderia ser tratado usando-se um método com restrição de conexidade estrita, pois assim cada cor estaria separada em uma única região contínua no mapa e a visualização seria imediata. Mas provavelmente, a classificação assim obtida, estaria muito longe de representar a realidade. Para isso, bastaria haver duas regiões do estado (ambas formadas por um número significativo de municípios) geograficamente distantes, mas com características semelhantes.

Aqui também seria possível usar as restrições de conexidade flexibilizadas. Isso implicaria em que as regiões pintadas com cada cor não seriam necessariamente conexas. Mas as restrições forçariam que o número de componentes conexos de cada cor seja pequeno, o que ainda permitiria uma boa visualização dos resultados. A vantagem é que flexibilizando a conexidade, temos muito mais liberdade para encontrar uma partição mais fiel à realidade. O usuário teria de testar diversos graus de conexidade, a fim de obter o compromisso desejado entre a precisão da classificação e a facilidade de visualização.

Pretendemos criar métodos de classificação com restrição de conexidade, que incorporem mecanismos para a flexibilização dessa conexidade, conforme explicado acima. Temos duas categorias de aplicações em mente:

- *Aplicações de tipo 1 em que se pretende usar informações adicionais de conexidade com objetivo de obter uma classificação mais correta do que seria possível apenas usando dissimilaridades pouco precisas.*
- *Aplicações geográficas, também de tipo 1, em que se deseja obter uma classificação mais grossa que possa ser facilmente visualizada num mapa, mas que ainda esteja fiel à realidade.*

Nada impede que uma aplicação use a conexidade para melhorar tanto a qualidade quanto a visualização da classificação.

Como o leitor já reparou, as aplicações da classificação com restrições de conexidade, estrita ou flexibilizada, costumam usar grafos H planares, pelo fato de H usualmente estar associado à localização espacial dos objetos a serem classificados numa superfície. Freqüentemente esses objetos são áreas na superfície terrestre, mas podem também ser *chips* num circuito impresso, etc.

Um subcaso muito comum, ocorre quando H além de planar, é um grafo de grade. Ele pode acontecer em aplicações semelhantes à descrita no exemplo 2.4, em que a localização dos objetos na superfície foi feita de maneira sistemática e regular. Por outro lado, aplicações semelhantes a do exemplo 2.5, em que os objetos foram posicionados de forma mais caótica, por forças naturais ou históricas, levam a grafos planares mais irregulares.

Um outro subcaso especial de grafo H relativamente comum, é o grafo de linha. [1] Esse costuma ocorrer quando H está associado à localização no tempo dos objetos, algo freqüente em aplicações em arqueologia, geologia, história e antropologia. Por sinal, esta é a razão da classificação já ter sido chamada de "serialização" nessas disciplinas.

Exemplo 2.6 *Fazendo-se uma escavação vertical num terreno sedimentar, sabe-se que os sedimentos mais profundos são estritamente mais antigos do que os mais rasos. Por*

[1] O nome *grafo de linha* é usual no contexto de classificação, por exemplo em Murtagh (1985) [28]. Esse grafo corresponde a um caminho, ou seja, uma árvore não-ramificada, como é habitualmente referenciado em teoria de grafos.

outro lado, a taxa em que a deposição ocorreu pode ter sido extremamente variada ao longo das eras, mas também sabe-se por experiência anterior que costuma ocorrer em "pacotes", ou períodos curtos com grande deposição, seguidos de períodos longos com pouca deposição. Há o problema de classificar as camadas da escavação de acordo com os dados fósseis encontrados. Como camadas adjacentes têm alta chance de pertencerem ao mesmo pacote e portanto serem semelhantes, aplicar um método com restrições de conexidade flexibilizadas sobre o grafo de linha definido pela seqüência de camadas, pode ajudar a obter uma melhor classificação.

O valores usuais de K na classificação de tipo 1, irrestrita ou com conexidade, são pequenos, quase sempre menores do que 10.

Pretendemos desenvolver métodos e algoritmos que se apliquem a um grafo H qualquer e a um valor de K qualquer. Porém, em algumas partes dessa dissertação, comentaremos também sobre esses três casos particulares de H (grafo planar, grade ou linha). Além disso, muitos dos algoritmos desenvolvidos podem não ser eficientes para grandes valores de K.

2.2 As Diferentes Definições de Método

Conforme dito no capítulo 1, um método de classificação é um **critério** que nos permite julgar a qualidade de uma dada partição. Dessa forma, um método especifica um problema de otimização, que pode ser resolvido, exata ou aproximadamente, por vários possíveis algoritmos.

Entretanto, não podemos deixar de dizer que essa nossa definição é um tanto recente e que ainda não é adotada por muitos dos autores atuais. Ela data do final da década de 60, a partir dos trabalhos de Singleton e Kautz (1965) [38], Vinod (1969) [43] e Rao (1971) [33], que não por coincidência foram os primeiros a usar técnicas de otimização para resolver problemas de classificação.

Tradicionalmente, um método é um **algoritmo** que a partir de uma instância de classificação, retorna uma K-partição. Naturalmente esse algoritmo é construído de modo a gerar soluções que satisfaçam o que se considera intuitivamente como uma boa classificação. Um método com restrição, seria um algoritmo que de alguma forma garanta que essa partição irá satisfazer uma certa propriedade.

Em alguns casos é possível estabelecer a equivalência entre as duas definições, mostrando-se que o método-algoritmo sempre otimiza um certo critério. Mas na maioria dos casos, que incluem vários dos métodos-algoritmos adotados com sucesso na prática, não é possível identificar o critério adotado. Na literatura é comum encontrar argumentos e justificativas a favor desses métodos em formato matemático, mas não há caracterizações precisas das

soluções procuradas. Esses métodos-algoritmos são mais facilmente interpretados como heurísticas para um ou mais critérios.

Nessa dissertação, a menos que indicado, usaremos apenas a definição de método como critério, mesmo porque os nossos métodos devem ser especificados dessa forma para poderem ser tratados por técnicas de otimização combinatória. Mas não temos dúvidas de que essa definição é conceitualmente superior, pois nos leva a explicitar claramente que tipo de estruturas procuramos. É verdade que escolher um bom critério para uma certa aplicação nem sempre é uma tarefa fácil, mas a longo prazo, é melhor aprender a fazer essa escolha, do que confiar no critério subjetivo embutido num método-algoritmo.

Mas reconhecemos que os autores que trabalham com a idéia de métodos-algoritmos têm um forte argumento: de que adianta definir métodos como problemas de otimização, se estes não podem ser computacionalmente resolvidos em instâncias reais ?

Essa objeção vem se tornando menos importante com o passar do tempo, devido ao enorme progresso que vem acontecendo nas últimas décadas, tanto nos computadores, quanto nos algoritmos de otimização. Hansen e Jaumard (1995) [16] listam os métodos-critério mais utilizados e fazem comentários sobre o estado-da-arte dos algoritmos para resolver os problemas associados. Pode-se ver que, apesar de todo o progresso, alguns desses problemas ainda permanecem desafiadores.

De qualquer forma, achamos que ainda é melhor usar um algoritmo aproximado para resolver um problema bem definido; do que usar um método-algoritmo, que no fundo também é uma heurística, só que para um problema mal definido.

2.3 Características dos Métodos

Há um grande número de métodos de classificação, que podem ser mais ou menos adequados a uma dada aplicação. Algumas características desses métodos devem ser levadas em conta na escolha de um deles. Falaremos de três delas, com o propósito de apresentar as características que pretendemos que os nossos métodos satisfaçam.

Viés - Muitos dos métodos possuem um viés estatístico, no sentido de que eles levam a algumas famílias de K-partições, numa freqüência maior do que o normal (Kaufman e Rousseeuw [22]). Todos os métodos têm algum viés. Mas, o que chamaremos de métodos com viés são aqueles que apresentam esse comportamento de forma acentuada.

Para entendermos com mais profundidade a questão do viés, devemos nos perguntar: qual é o formato esperado das classes naturais ou estruturas presentes nos dados [2]?

[2]Essa discussão em termos geométricos só se aplica diretamente se usarmos a interpretação da classificação como busca de regiões densas num espaço, mas em um sentido mais abstrato é válida em geral.

Essa é uma questão complicada, para a qual não poderia existir uma única resposta, mas depois de décadas de experiências concorda-se que, na maioria dos casos, as estruturas significativas que deveriam ser identificadas como uma classe são aproximadamente "esféricas" ou "elipsoidais" (Jain e Dubes [18]). Muitas vezes, na fase de construção das dissimilaridades, aplica-se aos dados uma operação de normalização, justamente para tentar transformar estruturas elipsoidais em esféricas, pois estas são mais facilmente detectáveis.

Supondo que numa instância todos os objetos estão distribuídos num certo número dessas estruturas mais ou menos esféricas e estas estão bem separadas entre si, como por exemplo na figura 2.1, temos um problema relativamente fácil e qualquer método razoável deveria ser capaz de encontrar a partição correta. Só que isso raramente acontece. Normalmente as estruturas estão parcialmente sobrepostas e existem alguns objetos bastante isolados dos demais (*outliers*). Temos um exemplo disso na figura 2.2, onde aparentemente há uma sobreposição parcial entre as duas classes da direita. Um bom método deveria ser capaz de reconhecer essas condições e fazer uma classificação semelhante (mesmo porque não é absolutamente clara a melhor maneira de classificar muitos dos objetos) à indicada. Naturalmente, é possível não haver estruturas significativas nos dados ou seja, os objetos podem estar espalhados de uma forma mais ou menos aleatória.

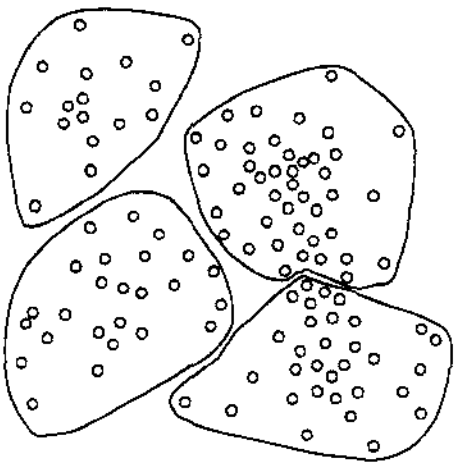


Figura 2.2: Uma instância mais complicada ($N = 115$, $K = 4$)

Quanto à distribuição do número de objetos em cada classe, supondo que há estruturas, pouco se pode afirmar a respeito. Tanto pode ser equilibrada, quanto existirem algumas razoavelmente maiores que as demais, embora não seja comum haver classes significativas excessivamente pequenas. [3]

Feitas essas considerações, podemos interpretar muitos dos métodos com viés, como aqueles que buscam estruturas especiais, atípicas. Assim, como na maioria dos casos essas

[3] É recomendável fazer uma análise à parte dos *outliers*. Eles freqüentemente indicam erros grosseiros na coleta dos dados ou fenômenos excepcionais.

estruturas não estão presentes, o método não vai conseguir achar as verdadeiras classes, mas vai impor ou "forçar" a sua própria estrutura sobre os dados. Isso vai se refletir num viés estatístico, pois essas estruturas especiais vão aparecer com uma freqüência muito maior do que seria razoável.

Por outro lado, se estamos conscientes de quais estruturas o método costuma encontrar e elas são compatíveis com a estrutura que supomos estar presente nos dados; então não somente o método é adequado, mas é até recomendável. Um bom indício de que um método tem viés (e um viés por si só): uma forte "preferência" por um tipo de distribuição do número de objetos nas classes. Por exemplo, fazendo com que esse número seja aproximadamente igual para todas as classes.

Entretanto, um método só pode ser bom para aplicações de tipo 1 em geral, caso ele tenha pouco viés. Isso quer dizer que o método, idealmente, deve procurar por classes esféricas, mas deve poder reconhecer sobreposições parciais de classes e ainda ter alguma flexibilidade para respeitar o tamanho das classes reais presentes nos dados.

Robustez ou Estabilidade - Alguns métodos são intrinsecamente instáveis, no sentido de que uma pequena alteração dos dados de entrada costuma levar a grandes alterações na classificação. A robustez é sempre desejável, mas é fundamental para um método ser adequado a aplicações de tipo 1. Caso contrário, os ruídos, que inevitavelmente estarão presentes na estimativa das dissimilaridades, podem fazer o método perder a estrutura verdadeira presente nos dados, tornando-o pouco confiável.

Na verdade, para qualquer método, sempre será possível construir instâncias para quais o método se mostra instável. O que desejamos é que um método seja estável quando aplicado a instâncias que contenham estruturas significativas.

Uma outra definição mais ou menos equivalente desse conceito: suponha que um método é aplicado a uma instância que contenha estrutura significativa. Se o método for estável, provavelmente todas as boas soluções aproximadas desse problema devem ser parecidas com a solução ótima. Essa equivalência decorre do fato de que pequenas mudanças nas dissimilaridades podem fazer uma dessas soluções aproximadas passar a ser a solução ótima.

Tratabilidade Computacional - Existem alguns poucos métodos para os quais se conhecem algoritmos exatos polinomiais. Infelizmente, estes não são considerados muito bons para a maioria das aplicações.

Os melhores métodos disponíveis levam a problemas que foram demonstrados ser NP-difíceis. Isto não quer dizer que todos esses problemas sejam igualmente intratáveis. Existem problemas NP-difíceis que costumam ter instâncias que podem ser bem resolvidas na prática, e outros que são realmente intratáveis, a não ser em instâncias pequenas.

Uma regra interessante, comprovada empiricamente por diversos autores: as instâncias

que contêm estruturas significativas são bem mais facilmente resolvidas do que instâncias aleatórias de mesmo tamanho, que não contêm estrutura. Esse bom comportamento é particularmente verdadeiro em métodos estáveis, em conseqüência da própria definição de estabilidade colocada acima.

Queremos propor métodos com pouco viés, robustos e computacionalmente tratáveis para instâncias de tamanho razoável.

2.4 Alguns Métodos de Classificação Irrestrita e suas Extensões para Conexidade

Os métodos de classificação com conexidade que aparecem na literatura costumam ser definidos como uma **extensão** de algum método irrestrito já bem estabelecido.

No caso de métodos-critério, isso é feito de modo direto: simplesmente temos o mesmo critério com uma restrição adicional. Naturalmente, outros algoritmos, exatos ou aproximados, devem ser apresentados para a resolução do novo e em geral mais difícil problema computacional. Já quando se trabalha com métodos-algoritmos, a relação entre o método irrestrito e sua extensão é mais subjetiva. O que se faz, é tentar alterar o mínimo possível o algoritmo original, mas de forma a garantir que as soluções geradas satisfaçam às novas restrições, na esperança de preservar as supostas boas propriedades do método irrestrito.

Ao se estudar uma extensão é importante conhecer um pouco das características do método irrestrito original, pois naturalmente ela "herda" parte dessas características. Por exemplo, se um método é instável, provavelmente suas extensões também o serão.

Comentários, avaliações de resultados práticos, comparações e críticas aos métodos irrestritos podem facilmente ser encontrados na literatura, dado que estes são o *mainstream* da disciplina e já são estudados há muito tempo. Há bem menos comentários sobre os métodos com conexidade, pois esse é um assunto mais recente e pouco explorado. Murtagh (1985) [28] é a referência básica sobre os métodos conhecidos até aquele ano. Para os criados posteriormente, temos apenas as indicações dos próprios autores dos métodos.

2.4.1 *Single-Linkage*

O *single-linkage* é talvez o método mais simples de classificação. Foi proposto (Florek et al. (1951), Sneath (1957), citados em Kaufman e Rousseeuw [22]) como o seguinte algoritmo:

1. Inicialmente associe cada um dos N objetos a uma classe.
2. Una as duas classes menos separadas. A separação entre a classe A e a classe B é definida como a menor dissimilaridade entre um objeto de A e outro de B.

3. Repita o passo 2 até que o número de classes seja igual a K.

Existem vários outros métodos-algoritmos que usam a mesma estratégia: partir de N classes e ir a cada passo unindo duas classes até que se tenha as K classes desejadas. Eles são conhecidos como **métodos aglomerativos** e só diferem entre si pela maneira de escolher quais classes devem ser unidas a cada iteração. (Passo 2).

O *single-linkage* foi introduzido como um método-algoritmo, mas ele tem uma definição equivalente como um critério: maximizar a separação da partição, que é definida como sendo a menor separação entre duas classes dessa partição.

$$\max_{P} \left[\min_{C_A \in P, C_B \in P, A \neq B} \left[\min_{i \in C_A, j \in C_B} d_{ij} \right] \right]$$

Apenas em 1969 (Gower e Ross, citado em Andberg [1]), percebeu-se que o *single-linkage* é exatamente equivalente ao algoritmo de Kruskal (1956) para achar uma árvore geradora de custo mínimo! Isso é compreensível quando se nota que o artigo de Sneath foi publicado no *Journal of Microbiology*. Cada uma das K classes geradas podem ser vistas como sendo um componente (uma árvore) da K-floresta de custo mínimo que cobre G (o grafo completo com custos nas arestas dados pelas dissimilaridades). As arestas dessas árvores são justamente as dissimilaridades mínimas encontradas no passo 2. Se especificarmos $K = 1$, teremos uma árvore de custo mínimo.

Apesar de ter outras propriedades matemáticas e computacionais interessantes e poder ser implementado em $O(N^2)$ (acha-se uma árvore geradora de custo mínimo T de G pelo algoritmo de Prim e remove-se as $K - 1$ maiores arestas de T), o *single-linkage* não é considerado um bom método pela maioria dos autores (Kaufman e Rousseeuw [22], por exemplo). Uma mínima sobreposição ou mesmo aproximação entre duas ou mais estruturas esféricas significativas pode ser suficiente para fazer o método uní-las em uma única classe alongada, um grande viés, como ilustrado na figura 2.3 (a separação da partição é indicada). Como conseqüência, freqüentemente temos uma classificação com quase todos os objetos em algumas poucas classes. O método também é bastante instável, por exemplo: adicionar um único objeto a uma instância pode fazer o *single-linkage* mudar completamente uma classificação.

Alguns trabalhos que estendem o *single-linkage* para restrição de conexidade:

- Extensão "míope" (Gordon (1974), Monestiez (1977), Fisher (1980), citados em Murtagh [28]). Consiste em alterar o critério de aglomeração do algoritmo dado acima, para unir as duas classes menos separadas, desde que essas classes sejam adjacentes em H. Assim, garante-se por construção, que as classes finais sejam conexas. O "míope" é um comentário de Murtagh, indicando o fato que o algoritmo toma suas decisões com base em poucas informações locais, o que freqüentemente leva a decisões ruins. Esse método-algoritmo tem como única vantagem ser computacionalmente muito simples.

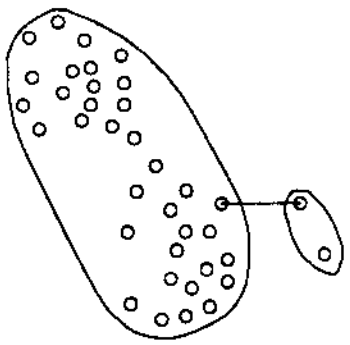


Figura 2.3: Uma classificação pouco representativa do *single-linkage* ($N = 37$, $K = 2$)

- *Maximum Split Clustering Under Connectivity Constraints* (Hansen et al. [15]). Aqui temos a extensão direta do critério do *single-linkage*: Procura-se a K-partição em classes conexas que maximiza a separação, um problema NP-difícil [15].

Parece que este trabalho é o primeiro a propor um algoritmo exato, chamado de CCMSC, especialmente construído para um método com restrição de conexidade. O CCMSC usa algumas das fortes propriedades combinatórias do *single-linkage* que ainda permanecem válidas com a nova restrição.

Proposição 2.1 *(Hubert (1973) [17]) O critério da separação máxima é invariante com respeito a transformações monótonas (que preservam a ordem) das dissimilaridades. Isso quer dizer que o single-linkage, mesmo com restrições adicionais, tem a mesma solução ótima, antes e depois de uma tal transformação.*

Por essa propriedade, o CCMSC não precisa trabalhar diretamente com os valores das dissimilaridades, mas apenas com inteiros na faixa de 1 até no máximo $N(N-1)/2$, uma simplificação muito útil.

Proposição 2.2 *(Delatre e Hansen (1980) [8]) Seja T uma árvore geradora de custo mínimo de G. A separação de uma partição arbitrária de G é sempre igual ao peso de alguma aresta (dissimilaridade) em T.*

Esse forte resultado implica que o valor da solução ótima estará dentro de um conjunto S, facilmente computável, com no máximo $N-1$ elementos.

De fato, o CCMSC procede pegando cada elemento s de S, em ordem decrescente, e procurando por uma partição conexa com separação exatamente s. Existem uma série de outras propriedades, muito complexas para serem expostas aqui, que muitas vezes permitem ao algoritmo verificar rapidamente que tal partição não existe ou deduzir que alguns conjuntos de objetos devem necessariamente estar unidos na mesma classe numa eventual partição desse tipo. Só que para decidir definitivamente essa questão, muitas

vezes é preciso resolver um *set covering* nada trivial. O CCMSC gasta a maior parte do tempo nesses subproblemas.

O CCMSC pode resolver em tempo razoável instâncias aleatórias com cerca de 150 objetos e instâncias estruturadas com até 300 objetos, desde que o número desejado de classes não seja grande (maior do que 10). Esse desempenho modesto, principalmente se considerarmos a presença de tantas propriedades combinatórias fortes, ilustra a dificuldade introduzida pela conexidade. Ao se introduzir a conexidade em praticamente qualquer outro método, não esperamos contar com esse tipo de propriedades.

Introduzir as restrições de conexidade flexibilizadas no *single-linkage* seria uma idéia interessante. Só que tal método não atenderia aos nossos objetivos, sendo pouco adequado para aplicações de tipo 1, devido às características negativas já acusadas no *single-linkage* irrestrito: forte viés e instabilidade.

2.4.2 *Complete-Linkage*

O *complete-linkage* (McQuitty (1960), Sokal e Sneath (1963), citado em Kaufman e Rousseeuw [22]) foi definido como o algoritmo aglomerativo em que o passo 2 é:

2. Una duas classes de forma que a nova classe obtida tenha o mínimo diâmetro. O diâmetro de uma classe é a máxima dissimilaridade entre dois objetos dessa classe.

A solução gerada por esse algoritmo tem a propriedade de minimizar o diâmetro da partição, que é definido como o máximo diâmetro entre duas de suas classes e portanto o *complete-linkage* também é um critério.

$$\min_{P} \left[\max_{C_A \in P} \left[\max_{i \in C_A, j \in C_A} d_{ij} \right] \right]$$

O *complete-linkage* tem um viés "oposto" ao do *single-linkage*. O método costuma separar as grandes estruturas esféricas em mais de uma classe. Isso se reflete numa tendência a gerar classificações com aproximadamente o mesmo número de objetos por classe. Na figura abaixo temos um exemplo desse comportamento. Indicamos o diâmetro de cada classe. O método também é muito instável. Pode ser implementado em $O(N^2)$.

Extensões algorítmicas míopes do *complete-linkage* foram experimentadas por Fisher (1980) e Ferligoj (1982), citados em Murtagh [28].

2.4.3 Mínima Variância

O método da mínima variância, que também é conhecido como método dos mínimos erros quadrados ou ainda mínima inércia, foi introduzido por Singleton e Kautz (1965) [38]. Esse método usa diretamente como entrada uma tabela em que cada objeto é descrito por m atributos. A variância de uma partição P é dada por:

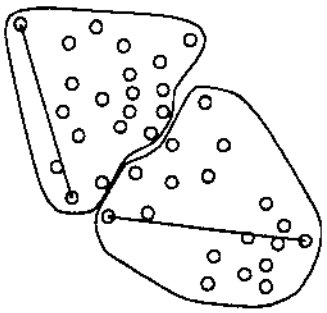


Figura 2.4: Uma classificação pouco representativa do *complete-linkage* ($N = 38$, $K = 2$)

$$Var(P) = \sum_{k=1}^{K} \sum_{i \in C_k} (d_{x(i)\,c(C_k)})^2$$

,

onde $d_{x(i)\,c(C_k)}$ é a distância euclidiana entre os pontos em m dimensões correspondentes aos atributos do objeto i e ao centróide da classe C_k, definido como:

$$c(C_k) = \sum_{i \in C_k} x(i)/|C_k|$$

.

O método da mínima variância está relacionado a procedimentos estatísticos clássicos e é um dos métodos-critério mais utilizados na prática.

As suas características de viés e estabilidade são boas, pelo menos em comparação com o *single-linkage* e o *complete-linkage*. Elas podem até ser provadas ótimas sob certas premissas, a saber: que os objetos das verdadeiras classes estão dispersos de acordo com uma distribuição normal e que essas classes estão bem separadas entre si. Na verdade isso não quer dizer muito, pois praticamente qualquer método razoável irá se sair bem nessas condições.

Nos últimos anos, têm surgido muitas críticas ao comportamento desse método em instâncias reais (Kaufman e Rousseeuw [22]), principalmente pelo fato de que a medida quadrática presente no critério "amplifica" demais o efeito dos ruídos presentes nos dados, causando instabilidade. [4] Essa medida quadrática também aumenta a tendência a distribuir uniformemente o número de objetos por classes.

O problema computacional gerado é NP-difícil. Os algoritmos exatos existentes costumam resolver instâncias com até 120 objetos, desde que as classes estejam razoavelmente separadas. Mas algumas instâncias complicadas bem menores ainda não podem ser resolvidas em tempo razoável (Hansen e Jaumard [16]).

[4] Isso deve ser visto no contexto da crítica ao tradicional uso da variância nas técnicas estatísticas em geral, feita recentemente pela chamada **estatística robusta**. Por exemplo, Kariya e Sinha (1989) [21].

Normalmente, usa-se na prática heurísticas de construção e busca local, com resultados aceitáveis (Jain e Dubes [18]). Essas heurísticas constroem uma partição inicial e em seguida tentam trocar um ou mais objetos de classe de modo a reduzir o custo da partição corrente, até que se fique bloqueado num mínimo local. Isso pode ser feito repetidas vezes, com diferentes partições iniciais.

O problema da mínima variância com restrição de conexidade foi atacado por Scott (1971) e Lebart (1978) (citados em [27]) com técnicas semelhantes. Aqui, constrói-se uma partição conexa e só são permitidas trocas de objetos que preservem a conexidade. Infelizmente, o desempenho desses algoritmos é pouco satisfatório. Um dos problemas é que, como a conexidade restringe fortemente o espaço de busca, rapidamente o algoritmo pode ficar bloqueado.

Para construir uma heurística de busca local que evita esse problema, chamada de MIDAS, Maravale e Simeone (1990) [27], usam a seguinte propriedade:

Proposição 2.3 *Existe uma árvore geradora T de H, tal que, para qualquer critério, o problema com conexidade sobre H tem a mesma solução do problema com conexidade sobre T.*

Encontrar essa árvore T é equivalente a resolver o problema original sobre H, mas essa proposição pode ajudar a projetar outro tipo de heurística de busca local. O MIDAS constrói uma "boa" árvore inicial T e resolve heuristicamente o problema, bem mais fácil, sobre T (o que fornece uma solução factível sobre H). Em seguida, T é modificada localmente, obtendo-se uma nova árvore T'. Resolve-se novamente o problema sobre T' e assim por diante. As modificações nas árvores são feitas de maneira a reduzir o custo das partições produzidas. Assim, pelo menos o problema do rápido bloqueio num mínimo local é evitado.

É difícil avaliar a qualidade do MIDAS dado que os autores não conhecem as soluções ótimas das instâncias de teste. Eles apenas comparam o MIDAS favoravelmente com as outras heurísticas citadas acima.

Optamos por não trabalhar com a conexidade flexibilizada sobre a mínima variância pelos seguinte motivos:

- O método só se aplica ao caso particular em que os objetos são descritos por uma tabela de atributos e as dissimilaridades são as distâncias euclidianas nessa tabela.
- As características de viés e estabilidade ainda não são boas o bastante para aplicações de tipo 1 em geral.
- O problema computacional gerado já é pouco tratável no caso irrestrito. As novas restrições provavelmente o tornariam tão complicado, que seria inviável pensar em algoritmos exatos para instâncias reais.

2.4.4 K-Medóide

O método do K-medóide (Vinod (1969) [43], Rao (1971) [33]) se baseia na escolha de K objetos dentre os N possíveis como **representantes** ou **medóides** das classes. Uma vez escolhidos esses representantes, as classes ficam definidas associando cada um dos objetos restantes ao representante com menor dissimilaridade. O método procura minimizar a soma das dissimilaridades de cada objeto ao seu representante.

A classificação a partir de representantes, apesar de semelhante, tem duas potenciais vantagens em relação à baseada em centróides. Primeiro, pode ser aplicada a qualquer matriz de dissimilaridades. Além disso, o fato de os representantes serem objetos reais, e não uma média, pode ser útil em várias aplicações. Por exemplo, quando se deseja selecionar os K objetos mais "representativos" de uma grande coleção de tamanho N, para um estudo mais aprofundado.

O K-medóide é equivalente a um problema clássico em otimização conhecido como *p-median*, que é uma versão do problema genérico de localização de facilidades sem capacidade (*Uncapacitated Facility Location*). Por exemplo: existem N cidades e deve-se decidir em quais delas instalar K fábricas de modo a minimizar o custo total do transporte entre cada cidade e a fábrica mais próxima. O termo "sem capacidade" indica que não consideramos a existência de possíveis limites ao número de cidades que podem ser atendidas por uma fábrica.

Temos a seguinte formulação em programação linear inteira:

Dados: d_{ij}: Dissimilaridade entre os objetos i e j
N: Número de objetos
K: Número de classes desejadas

Variáveis: $s_{ij} = 1$ se o objeto i está associado ao representante j, 0 c.c.
$r_j = 1$ se o objeto j é um representante, 0 c.c.

$$\text{Min} \quad \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}s_{ij} \tag{1}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{N} s_{ij} = 1 \qquad i = 1,\ldots,N \tag{2}$$

$$s_{ij} \leq r_j \qquad i,j = 1,\ldots,N \tag{3}$$

$$\sum_{j=1}^{N} r_j = K \tag{4}$$

$$s_{ij}, r_j \in \{0,1\} \qquad i,j = 1,\ldots,N \tag{5}$$

O K-medóide tem um certo viés, pois pode associar mais de um representante para uma única classe natural, se esta tiver um grande número de objetos. Acredita-se (Kauf-

man e Rousseeuw [22]) que esse viés seja menor do que o da mínima variância, pelo fato de se usar uma medida linear de dispersão e não quadrática. A estabilidade do método é bastante satisfatória.

O método do K-medóide define um problema NP-difícil, mas que costuma ser tratável na prática. O espaço de soluções é limitado a $\binom{N}{K}$ possibilidades, o que ainda pode ser um número muito grande, mas é bem menor que o número de possíveis K-partições de N objetos. O problema poderia ser resolvido por enumeração completa em tempo polinomial para um K fixo. Mas é claro, temos o recurso de usar todas as técnicas do bem estudado *p-median*. Com elas, instâncias com alguns milhares de objetos já foram resolvidas de forma exata. Boas soluções aproximadas podem ser rapidamente obtidas para instâncias ainda maiores.

As melhores técnicas estão baseadas na formulação (1-5) dada acima, que é uma **formulação forte**, no sentido de que o valor obtido ao se resolver a sua relaxação linear quase sempre está muito próximo, ou até se iguala, ao valor da solução inteira procurada. Só que não é possível resolver diretamente esses programas lineares (**PLs**), por exemplo pelo método simplex, quando as instâncias são grandes. O número quadrático de variáveis e restrições usadas, faz com que o tempo gasto seja excessivo. Procura-se então resolver variações desses PLs, através de algoritmos capazes de explorar as propriedades particulares do problema.

Uma das abordagens mais bem sucedidas (Christofides e Beasley (1982) [6], Beasley (1985) [3]) utiliza uma relaxação lagrangeana, dualizando as restrições (2).

$$Z_D = \text{Min} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}(d_{ij} - \lambda_i)s_{ij} + \sum_{i=1}^{N}\lambda_i \qquad (6)$$

$$\text{Sujeito a} \quad s_{ij} \leq r_j \qquad i,j = 1,\ldots,N \qquad (7)$$

$$\sum_{j=1}^{N} r_j = K \qquad (8)$$

Para um vetor λ fixo, o problema acima pode ser facilmente resolvido. Suponhamos que $r_j = 1$. Nesse caso também devem estar em 1 todos os s_{ij} tais que $d_{ij} - \lambda_i < 0$. Seja α_j a soma desses valores negativos, ou seja, a contribuição à função objetivo de se fazer $r_j = 1$. Deve-se colocar em 1 os r_j correspondentes aos K menores valores de α_j.

Usa-se um procedimento de subgradiente para ajustar iterativamente os valores de λ, a fim de maximizar Z_D. O limite dual [5] obtido é exatamente igual ao valor da relaxação linear de (1-5).

Os r_j escolhidos a cada iteração são usados para obter boas soluções factíveis, simplesmente associando-se cada objeto ao representante (objeto j com r_j=1) mais próximo.

[5] **Limites duais** significam limites inferiores em problemas de minimização e limites superiores em maximização. **Limites primais** tem um significado complementar.

Eventualmente pode ser necessária uma fase de *branch-and-bound* para fechar o *gap* entre os limites duais e primais conhecidos.

Mas a técnica mais eficiente disponível para o *p-median* (e portanto para o K-medóide) é derivada de uma técnica que resolve o seguinte problema:

$$\text{Min} \quad \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}s_{ij} + \sum_{j=1}^{N} f_j r_j \tag{9}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{N} s_{ij} = 1 \qquad i = 1,\ldots,N \tag{10}$$

$$s_{ij} \leq r_j \qquad i,j = 1,\ldots,N \tag{11}$$

$$s_{ij}, r_j \in \{0,1\} \qquad i,j = 1,\ldots,N \tag{12}$$

Aqui, ao invés de termos um número pré-determinado de representantes (e portanto de classes), há um custo fixo positivo de se fazer de j um representante. Esse problema computacional às vezes é chamado de *simple facility location*. O dual da relaxação linear de (9-12) pode ser escrito como:

$$\text{Max} \quad \sum_{i=1}^{N} v_i \tag{13}$$

$$\text{Sujeito a} \quad \sum_{i=1}^{N} w_{ij} \leq f_j \qquad j = 1,\ldots,N \tag{14}$$

$$v_i - w_{ij} \leq d_{ij} \qquad i,j = 1,\ldots,N \tag{15}$$

$$w_{ij} \geq 0 \qquad i,j = 1,\ldots,N \tag{16}$$

Para qualquer escolha factível das variáveis duais v, fixar cada variável w_{ij} no mínimo valor possível, irá manter a factibilidade sem mudar o valor da função objetivo. Logo, podemos assumir que $w_{ij} = \max\{0, v_i - d_{ij}\}$. Fazendo essa substituição, temos o seguinte dual condensado apenas nas variáveis v.

$$\text{Max} \quad \sum_{i=1}^{N} v_i \tag{17}$$

$$\text{Sujeito a} \quad \sum_{i=1}^{N} \max\{0, v_i - d_{ij}\} \leq f_j \quad j = 1,\ldots,N \tag{18}$$

Erlenkotter (1978) [9] propôs um procedimento heurístico chamado de DUALLOC para obter rapidamente boas soluções para esse dual. Basicamente, esse procedimento aumenta uma variável v_i por vez até que ela seja bloqueada por alguma restrição de (18). Korkel (1989) [23] propôs um melhoramento desse procedimento, de forma a permitir que mais de um v_i seja aumentado por vez. Dada uma solução dual factível, pode-se (para

esse problema particular) obter boas soluções primais inteiras usando o teorema das folgas complementares e arredondamentos. Pode ser necessária uma fase de *branch-and-bound*.

Suponhamos que resolvemos um *simple facility location* com todos os valores de f_j iguais a um certo f e obtemos uma solução com K representantes. Podemos ter certeza que a solução obtida é a melhor solução possível com esse número de representantes e portanto também é a solução do K-medóide para K classes. Dessa forma, pode-se resolver instâncias do K-medóide ajustando-se o valor de f (por exemplo, por busca binária) até que se obtenha o número de classes desejado. Como as técnicas descritas acima para o *simple facility location* são muito eficientes, vale a pena fazer isso. A única dificuldade é que é possível não haver nenhum valor de f que leve a exatamente K classes, para alguns valores de K. Mas isso só costuma ocorrer quando K é relativamente grande.

Ponderando-se as características de viés, estabilidade e tratabilidade computacional mostradas acima, podemos dizer que o K-medóide é um bom método para aplicações de tipo 1 em geral. Ele é especialmente recomendado em Kaufman e Rousseeuw [22]. [6]

É importante ainda notar que o K-medóide pode facilmente ser convertido num método muito parecido com a mínima variância. Basta usar como dissimilaridade, a distância euclidiana entre os atributos dos objetos elevada ao quadrado. A única diferença, é que ao invés de uma medida de dispersão quadrática em relação a um centróide, teremos uma medida quadrática em relação a um representante. É razoável supor que, pelo menos em instâncias com estrutura, os centróides que seriam obtidos pela mínima variância e os representantes obtidos pelo novo método, estejam próximos. Portanto, as classificações dos dois métodos devem ser semelhantes.

Por essa observação, nos parece natural usar a maior facilidade computacional do K-medóide, para obter uma boa heurística para a mínima variância, idéia que ainda não foi devidamente explorada na literatura. Essa heurística teria uma boa propriedade: a qualidade da aproximação tenderia a melhorar com o aumento de N, pois haveria maior chance de haver objetos muito próximos a cada um dos centróides.

Não conhecemos extensões do K-medóide para a conexidade.

Decidimos trabalhar sobre métodos que estendem o K-medóide, incorporando a conexidade flexibilizada, pelos seguintes motivos:

- *Esperamos que as boas características de viés e estabilidade do método irrestrito, adequadas para aplicações do tipo 1 em geral, sejam mais ou menos preservadas nessas extensões.*
- *Como o problema computacional do método irrestrito é bem tratável na prática,*

[6] Aparentemente, esses autores não estavam a par das técnicas de otimização mais modernas para resolver o problema, pois após o comentário de que instâncias de até 50 objetos podiam ser resolvidas exatamente por *branch-and-bound*, é proposta uma heurística de busca local para instâncias de até 100 objetos. Isso não seria de estranhar, dada a interdisciplinaridade envolvida no estudo da classificação. Os autores que dominam o aspecto estatístico do problema, geralmente não são especialistas também nos aspectos computacionais e vice-versa.

esperamos que, mesmo com a significativa complicação adicional introduzida pela conexidade, ainda sejamos capazes de desenvolver algoritmos para instâncias de tamanho razoável.

- *Esses métodos podem ser facilmente convertidos em boas heurísticas para o clássico critério da mínima variância.*

Capítulo 3

Estendendo o K-Medóide para a Conexidade Flexibilizada

3.1 O Uso de Duas etapas

A maneira usual de se estender um método para incorporar a conexidade, é simplesmente manter o critério irrestrito, mas agora sujeito a novas restrições. Vejamos como isso ficaria no nosso caso.

Seja $P = \{C_1, C_2, ..., C_K\}$ uma partição qualquer dos objetos. O representante da classe k, $r(k)$ é definido como objeto contido em C_k que minimiza $S_k = \sum_{i \in C_k} d_{i\,r(k)}$. O nosso objetivo é encontrar a partição factível (que respeite a conexidade flexibilizada) que minimize $\sum_k S_k$.

Anteriormente, nas formulações do K-medóide irrestrito, usamos implicitamente uma propriedade altamente simplificadora: numa partição ótima, para qualquer objeto $i \in C_k$, $d_{i\,r(k)}$ deve ser menor ou igual a $d_{i\,r(k')}$ para qualquer outra classe k'. O que implica que um objeto sempre deve estar na mesma classe em que estiver o representante mais próximo. Assim, as partições que precisavam ser consideradas eram unicamente determinadas pela escolha dos K representantes (a menos de objetos equidistantes de dois representantes).

Com as novas restrições isso deixa de ser verdade. Agora temos que saber quais objetos estão na mesma classe e, a partir daí, achar os representantes e avaliar o custo de uma solução. Uma possível formulação para esse problema, a partir da formulação do K-medóide irrestrito (Cap 2:(1-5)), é dada a seguir. Novas variáveis binárias x são introduzidas, sendo que x_{ik} é igual a 1 se e somente se o objeto i é atribuído à classe k. Assim, dois objetos i e j estão na mesma classe se e somente se $x_{ik} = x_{jk} = 1$, para algum k.

$$\text{Min} \quad \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}s_{ij} \tag{1}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{N} s_{ij} = 1 \qquad i = 1,\ldots,N \tag{2}$$

$$s_{ij} \leq r_j \qquad i,j = 1,\ldots,N \tag{3}$$

$$\sum_{j=1}^{N} r_j = K \tag{4}$$

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,N \tag{5}$$

$$s_{ij} + x_{ik'} + x_{jk''} \leq 2 \qquad i,j = 1,\ldots,N; \quad k',k'' = 1,\ldots,K, k' \neq k'' \tag{6}$$

$$\sum_{i=1}^{N} x_{ik} \geq 1 \qquad k = 1,\ldots,K \tag{7}$$

$$CF(x) \tag{8}$$

$CF(x)$ representa as restrições de conexidade flexibilizadas que devem ser satisfeitas pelo vetor x de atribuições. A restrições (5-6) garantem respectivamente que cada objeto seja atribuído a exatamente uma classe e que um objeto atribuído à classe k se associe a um representante também atribuído à classe k, mesmo quando houver outro representante mais próximo. Temos que garantir por (7) que haverá pelo menos um objeto em cada classe e conseqüentemente, exatamente um representante será atribuído a cada classe.

Na verdade, no início do nosso trabalho decidimos não trabalhar com esse tipo de extensão direta. As razões para essa decisão:

1. Supusemos na época, que os problemas computacionais gerados por essa extensão direta seriam muito difíceis para serem tratados em instâncias de tamanho razoável e que as técnicas que fazem o K-medóide irrestrito ser bem resolvido não teriam como ser adaptadas para esse caso.

   Seria inviável usar formulações semelhantes à mostrada acima. Em primeiro lugar, restrições como (6) enfraquecem muito a relaxação linear dessa formulação. Haveria forte tendência de obter soluções x fracionárias. Temos também um número muito grande de variáveis e restrições e aparentemente poucas propriedades estruturais para explorar. Há ainda uma **simetria** embutida nas variáveis x, pois uma mesma solução x pode ser representada de $K!$ maneiras diferentes, apenas permutando-se os índices k. A simetria de uma formulação pode comprometer definitivamente o desempenho de um algoritmo de *branch-and-bound*. Todas essas complicações já aparecem antes mesmo de considerarmos as restrições de conexidade. Não sabíamos

como produzir uma formulação melhor, mais forte e razoavelmente compacta para esse problema.

2. Na mesma época, tínhamos em mente apenas aplicações que usam conexidade flexibilizada para facilitar a apresentação dos resultados num mapa. Nesses casos, supõe-se que uma classificação irrestrita seja a melhor possível. Usa-se a conexidade para se obter um compromisso entre a precisão da classificação e a facilidade de visualização. Nesse contexto, envolver as restrições de conexidade na determinação dos representantes não seria conceitualmente correto, pois estes não seriam de fato os objetos mais típicos de uma classe.

Assim, decidimos trabalhar num método que introduz a conexidade flexibilizada como uma segunda etapa, após ter sido feita uma classificação inicial pelo K-medóide irrestrito. Nessa classificação irrestrita, encontramos os K representantes e os fixamos. Cada um deles passa a identificar uma classe. Em seguida resolvemos o seguinte problema: achar uma atribuição de cada objeto a uma das classes que satisfaça a conexidade flexibilizada e que minimize a soma das dissimilaridades de cada objeto ao representante fixo que identifica a sua classe.

A figura 3.1 ilustra as duas etapas do processo. As setas entrando num objeto indicam a qual representante ele está associado. Em (A) temos a classificação irrestrita da primeira etapa. Cada objeto está associado ao representante mais próximo. Em (B) temos a classificação da segunda etapa. Os representantes continuam os mesmos, pois foram fixados. Mas 3 objetos mudaram de classe, a fim de satisfazer as restrições de conexidade com mínimo custo. Agora já não é verdade que cada objeto esteja associado ao representante mais próximo. Nesse exemplo, um dos objetos a mudar de classe foi justamente o representante da classe da esquerda. Isso não impede que esse representante continue identificando a mesma classe e tenha outros objetos associados a ele. Na prática, isso é pouco comum. Os objetos com maior probabilidade de mudar de classe são aqueles que estão localizados perto da "fronteira" entre duas ou mais classes na classificação original.

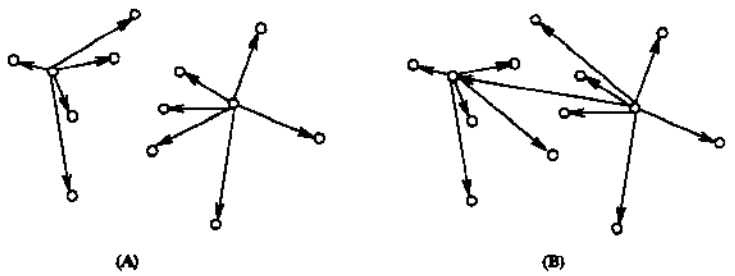


Figura 3.1: As classificações de cada etapa ($N = 13$,$K = 2$)

Agora, o leitor pode entender com mais clareza o mecanismo envolvido no uso dos métodos com conexidade. Por exemplo, na aplicação de correção de dissimilaridades rui-

dosas. Nesses casos, os ruídos podem fazer as estruturas corretas que supõe-se existir nos dados, mais dispersas, espalhadas, a ponto de se sobreporem e se confundirem parcialmente. Nesses casos, como mostrado na figura 2.2, os objetos localizados próximos à fronteira entre duas ou mais classes, têm alta chance de serem mal classificados por um método irrestrito. As informações adicionais dadas pela conexidade podem melhorar a qualidade da classificação desses objetos. Mas deve-se saber dosar o grau de conexidade. O uso de uma restrição excessivamente exigente, como na maioria casos é a conexidade estrita, pode forçar muitos objetos que estavam bem classificados a mudar de classe

A primeira etapa pode ser resolvida por qualquer uma das excelentes técnicas disponíveis para o K-medóide irrestrito. A segunda etapa pode ser formulada da maneira descrita abaixo. A partir de agora, d_{ik} será usado como abreviação de $d_{i\,r(k)}$, onde $r(k)$ é o representante que identifica a classe k.

$$\text{Min} \quad \sum_{i=1}^{N}\sum_{k=1}^{K} d_{ik}x_{ik} \tag{9}$$

$$\text{Sujeito a} \quad \sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,N \tag{10}$$

$$CF(x) \tag{11}$$

Essa formulação não tem nenhum dos problemas apontados na formulação anterior: tamanho excessivo, fraqueza e simetria. Notar que permutar os índices k de uma solução x muda completamente o valor da função objetivo, pois os r(k) estão fixos. Resta tratar as restrições de conexidade, um problema nada trivial, mas não há dúvidas que dividir o problema em duas etapas facilita em muito a resolução computacional.

Por outro lado, nas aplicações que usam a conexidade flexibilizada para corrigir dissimilaridades imprecisas, essa divisão em duas etapas não é completamente justificada. Nesses casos, é possível que melhores representantes pudessem ser escolhidos levando-se em conta a conexidade desde o início, em uma única etapa, como acontece na extensão direta.

Achamos muito provável que a diferença entre as classificações obtidas pela extensão direta e pela divisão em etapas, não seja grande.

Essa observação está baseada no fato de que os objetos "centrais" de uma classe (aqueles próximos aos representantes encontrados pelo K-medóide irrestrito), costumam permanecer na mesma classe após a introdução de novas restrições, mesmo que a nova classificação fosse feita em uma única etapa pela extensão direta. Por outro lado, os objetos com alta tendência a mudar de classe (novamente em relação à classificação irrestrita), são os objetos periféricos próximos às fronteiras com outras classes. Mas nessas fronteiras, o processo de mudança é estatisticamente simétrico, tanto há objetos entrando quanto saindo de uma classe, embora os objetos que entram sempre estejam mais distantes do representante irrestrito do que os que saem. Mas esses centros, que normalmente são os

locais com maior concentração de objetos funcionam como um "lastro" , de forma que os novos representantes que seriam encontrados pela extensão direta devem continuar próximos aos representantes do K-medóide irrestrito. [1]

Essa aproximação deveria funcionar particularmente bem se as instâncias tiverem estruturas significativas; um tamanho razoável; e o grau de conexidade especificado não for alto, pois quanto mais estrita a restrição, maiores são as mudanças de classe.

Seria interessante verificar experimentalmente até que ponto tudo isso é verdadeiro. A dificuldade de fazer essa comparação é resolver exatamente o problema da extensão direta em instâncias de tamanho razoável.

De qualquer maneira, deve ficar claro que: em aplicações de correção de dissimilaridade, os nossos métodos em duas etapas, mesmo se ambas as etapas forem resolvidas exatamente, devem ser vistos como heurísticas para o método, possivelmente mais adequado, da extensão direta.

3.2 Conexidade Flexibilizada por Contagem de Componentes

Falta agora especificar quais são as propriedades que devem ser satisfeitas por um vetor x de atribuições, conforme a formulação (9-11), para completarmos a definição da segunda etapa dos nossos métodos com conexidade flexibilizada.

Conforme dito, queremos que um usuário do método possa controlar através de parâmetros o grau de conexidade desejado. Esse controle será feito através da contagem do número de componentes induzidos pelo vetor de atribuições x. Um **componente** é um subgrafo conexo maximal de H formado por vértices atribuídos à uma mesma classe.

Há várias versões do controle por contagem de componentes:

a) O usuário usaria como parâmetro um vetor $\gamma = (\gamma_1, \ldots, \gamma_K)$ para especificar o número máximo γ_k de componentes permitido por classe k. Diremos que uma atribuição x é **γ-conexa** quando ela respeitar essas restrições.

b) O usuário especificaria apenas um número γ, que indicaria o número máximo total de componentes, ou seja a soma do número de componentes em cada classe. Dessa forma, algumas classes podem ter mais componentes, desde que outras tenham menos.

[1] A analogia com o conceito físico de centro de massa é perfeita no critério quadrático da mínima variância (por isso também chamado de mínima inércia). Felizmente, a linearidade do nosso critério funciona a nosso favor, minimizando o efeito que as alterações ocorridas na periferia de uma classe podem ter sobre o "medóide de massa".

c) Não existiriam limites pré-estabelecidos ao número de componentes, mas haveria um custo fixo f_k por um componente da classe k. Nesse caso teríamos uma formulação um pouco diferente do modelo dado em (9-11). Necessariamente teremos que ter variáveis adicionais, que contem o número de componentes por classe, para que possamos "cobrar" na função objetivo o termo f_k.

Talvez as versões (b) e principalmente a (c), sejam as mais interessantes para o uso prático em aplicações de tipo 1. Isso porque nesses casos . o método tem maior liberdade para encontrar por si mesmo uma distribuição dos componentes por classe que respeite as estruturas presentes nos dados. Por outro lado, na versão (a), a menos que o usuário escolha cuidadosamente os valores de γ_k (talvez por tentativa e erro), há uma chance maior de se forçar uma estrutura inexistente nos dados.

Entretanto, decidimos nos concentrar no estudo do problema gerado pela versão (a), que chamamos de **Problema da Atribuição γ-Conexa (PAgC)**, pelos seguintes motivos:

- Esse é a versão mais "completa". De fato, todos os algoritmos que desenvolvemos para essa versão podem facilmente ser adaptados para as versões (b) e (c).

- Essa versão é a que generaliza de modo imediato a idéia da conexidade estrita, simplesmente fazendo-se todos os γ_k iguais a 1. Dessa forma teremos um método diretamente aplicável nos casos em que a conexidade estrita é realmente desejada. Além disso, podemos ter uma base de comparação dos resultados computacionais por nós obtidos para esse caso estrito, com outros resultados encontrados na literatura.

O PAgC é um problema muito interessante por si só. que não apenas serve como segunda etapa no nosso método de classificação, mas surge naturalmente em outros contextos.

Exemplo 3.1 *Um fazendeiro decidiu plantar soja, milho e feijão para a próxima safra. A sua grande propriedade já foi objeto de um estudo agronômico e existe um relatório que descreve áreas menores dentro da fazenda, de acordo com a composição do solo, topografia e umidade. A partir desses dados pode-se estimar o custo (ou o lucro esperado) de se plantar uma certa cultura em cada uma dessas áreas. A primeira idéia é plantar em cada área, a cultura com mínimo custo para aquela área, o que naturalmente leva ao mínimo custo total possível. Mas fazendo-se isso, percebeu-se que as áreas plantadas com a mesma cultura ficariam separadas em muitas partes isoladas entre si. Isso não é conveniente por várias razões logísticas. Por exemplo: o maquinário agrícola, que é específico a cada cultura, ficaria ocioso enquanto estivesse sendo transportado de uma parte a outra. Nesse sentido, seria muito melhor atribuir culturas a áreas de forma que cada uma das três culturas estivessem unidas numa única parte contínua. Por outro*

lado, usar uma restrição tão estrita faria o custo total crescer demais. Pensando-se melhor percebeu-se que essa restrição não era absolutamente necessária. Não haveria problema em ter as culturas separadas, desde que em algumas poucas partes. Com essa flexibilização pode-se encontrar uma atribuição significativamente mais econômica, sem complicar demais a logística.

Essa aplicação leva exatamente ao PAgC, só que os d_{ik} não são dissimilaridades, mas os custos (ou lucros esperados) de se plantar a cultura k na área i. No caso de se trabalhar com lucros, teremos um problema de maximização, o que não muda a essência do problema. As versões (b) e (c) da contagem de componentes também poderiam ser aplicadas aqui.

Na aplicação 3.1, obter uma solução com 0 componentes em uma classe é algo normal. Isso indica que não vale a pena plantar alguma das culturas.

Já quando o PAgC é usado como segunda etapa dos nossos métodos, é interessante que não existam classes com 0 componentes, ou seja, que obtenhamos uma classificação em K classes e não menos. Não nos preocupamos em colocar uma restrição adicional desse tipo por simplicidade, pois é praticamente certo que ela será naturalmente satisfeita em instâncias reais desse tipo. Mas não haveria qualquer dificuldade em fazê-lo.

Capítulo 4

Problema da Atribuição γ-Conexa

4.1 Complexidade do Problema

Como mencionado no capítulo 2, as aplicações reais de classificação freqüentemente levam a instâncias mais restritas, por exemplo, quando H é planar, uma grade ou uma linha, ou ainda, quando K é pequeno. Procuramos estabelecer a complexidade também desses casos particulares.

4.1.1 Casos NP-completos

A fim de provar a complexidade do caso geral, para deixar claro qual é exatamente o problema que está sendo tratado e também para fixar a notação usada nesse capítulo, vamos expressar o PAgC formalmente, como um problema de decisão. Note que uma atribuição de cada objeto a uma das classes pode ser vista como uma partição $P = \{C_1, C_2, ..., C_K\}$ dos vértices de H. Uma partição é factível se os subgrafos induzidos por cada C_k tem no máximo γ_k componentes. Seja $c(P) = \sum_{k=1}^{K} \sum_{i \in C_k} d_{ik}$ o custo de uma tal partição. Essa partição é ordenada, ou seja, a ordem dos conjuntos que formam P é significativa: ordens diferentes podem implicar em custos diferentes.

Problema da Atribuição γ-Conexa (PAgC)

Instância: Grafo $H = (V, E)$; inteiro positivo K, matriz $D = (d_{ik})$ de dimensão $|V| \times K$ de racionais não-negativos, vetor $\gamma = (\gamma_1, \ldots, \gamma_K)$ de inteiros positivos, racional positivo c.

Questão: Existe uma partição factível P tal que $c(P) \leq c$?

Proposição 4.1 *PAgC é NP-completo.*

Prova: O problema claramente pertence à classe NP. A fim de provar sua NP-completude, mostraremos uma redução a partir da seguinte versão do problema de Steiner em grafos:

Problema de Steiner com Peso nos Vértices (PSPV)

Instância: Grafo $H' = (V', E')$, subconjunto $R \subseteq V'$, racionais não-negativos q_i para cada $i \in V' \setminus R$; racional q.
Questão: Existe uma árvore T, contida em H', que inclui todos os vértices de R com custo $q(T) = \sum_{i \in T \setminus R} q_i$ no máximo q ?

A versão do problema de Steiner em grafos que geralmente aparece na literatura e que é sabidamente NP-completo (problema ND12 em Garey e Johnson [12]), tem pesos não-negativos nas arestas. Para ver que o problema de Steiner permanece NP-completo mesmo quando os pesos não-negativos estão nos vértices, basta subdividir cada aresta (u,v) com um novo vértice w, que recebe o peso de (u,v). Os vértices originais do grafo recebem peso 0. Visto que PSPV é NP-completo, podemos prosseguir.

Faça $H = H'$; $K = 2$; $d_{i1} = 0, \forall i \in R$; $d_{i1} = q_i, \forall i \in V' \setminus R$; $d_{i2} = q + 1, \forall i \in R$ e $d_{i2} = 0, \forall i \in V' \setminus R$; $\gamma_1 = 1$, $\gamma_2 = |V'|$; $c = q$.

Suponha que PSPV tem resposta "sim". Defina uma partição P da seguinte maneira: C_1 é o conjunto dos vértices em T. C_2 é formado pelos vértices restantes. Como C_1 é conexo, P é uma partição factível. $c(P) = q(T) \leq q = c$. Logo, PAgC também tem resposta "sim". Agora suponha que PAgC tem resposta "sim". Defina T como qualquer árvore geradora do componente conexo C_1. Como $c(P) \leq c = q < q + 1$, C_1 contém todos os vértices de R, e T é uma árvore de Steiner factível. $q(T) = c(P) \leq c = q$, logo PSPV também tem resposta "sim". □

Essa prova mostra que o PAgC é NP-completo mesmo se $K = 2$. O mesmo raciocínio pode ser usado para mostrar que o PAgC é NP-completo para qualquer valor fixo de $K \geq 2$.

Além disso, como a prova preserva o grafo, o PAgC permanece NP-completo mesmo quando H é restrito a ser um grafo planar, bipartido ou de grade, pois o PSPV também permanece NP-completo nesses casos. Essa última afirmação segue do fato de que o problema de Steiner com peso nas arestas é NP-completo nesses casos [12]. A transformação é a mesma dada acima para o caso geral. Como a subdivisão das arestas preserva a planaridade e a bipartição, ela é suficiente para esses dois casos. No caso do grafo de grade, temos que adicionar alguns vértices e arestas ao grafo, a fim de restaurar a estrutura de grade. Esses vértices adicionais recebem pesos altos para não ocorerrem na solução ótima.

4.1.2 Casos Polinomiais

Queremos mostrar que o PAgC pode ser resolvido em tempo polinomial quando H é uma árvore. Para mostrar isso, primeiro tratamos o caso da linha, uma árvore especial, e a versão com um limite γ ao número total de componentes. A descrição de um algoritmo de programação dinâmica para esse caso segue:

ALGORITMO LINHA

1. **Definições e Inicialização**

   Escolha um vértice f de grau 1. Chamaremos f de *folha* do grafo de linha. O outro vértice r de grau 1 será chamado de *raiz*. Para cada $i \in V, i \neq r$, $p(i)$ (o *predecessor* de i) é definido como o vértice adjacente a i que está no caminho de r a i. O *sucessor* de u, é definido como $s(u) = i$ se e somente se $p(i) = u$.

   Para cada $i \in V$, cada inteiro k entre 1 e K e cada inteiro c entre 1 e γ; seja $H(i,k,c)$ o valor da solução ótima do problema sobre o subgrafo enraizado em i formado pelo caminho de i até f, atribuindo i à classe k e usando no máximo c componentes. Claramente, $H(f,k,c) = d_{fk}$ para qualquer k e qualquer c.

2. **Iterações**

   As soluções H restantes são calculadas pela seguinte recursão:

$$H(i,k,c) = d_{ik} + \min[\{H(s(i),k,c)\} \cup \{H(s(i),k',c-1) : \forall k' \neq k \in \{1,\ldots,K\}, \text{ se } c \geq 1\}]$$

3. **Solução Ótima**

   O valor da solução ótima do problema é dado por $\min_k H(r,k,\gamma)$.

A idéia intuitiva do algoritmo é usar o número de componentes disponíveis c, um recurso escasso, da melhor maneira possível. Sempre que se atribui o vértice i para uma classe diferente da classe de $s(i)$, "gasta-se" um componente. Mas sabendo-se todos os valores de $H(s(i),k,c)$ pode-se tomar essa decisão de maneira ótima.

O algoritmo tem complexidade $O(|V|.K^2.\gamma)$, o que é polinomial sobre o tamanho da instância, pois esse tamanho cresce linearmente com K e γ é limitado por $|V|$. Para adaptar esse algoritmo para o PAgC, temos que controlar o número de componentes disponíveis por classe, calculando valores da forma $H(i,k,c_1,\ldots,c_K)$. O algoritmo resultante tem complexidade $O(|V|.K^2.\prod_k \gamma_k)$, o que é polinomial para um K fixo.

Agora generalizamos essa abordagem para o caso em que H é uma árvore.

ALGORITMO ÁRVORE

1. **Definições e Inicialização**

   Escolha qualquer vértice para ser a raiz r. Todos os vértices de grau 1 (com a possível excessão de r) serão chamados de folhas. A definição de predecessor e sucessor continua a mesma, exceto que agora temos um conjunto $S(i)$ dos sucessores de um vértice i.

   Para cada $i \in V$, cada $k \in K$ e cada inteiro c $(1 \leq c \leq \gamma)$, seja $H(i,k,c)$ o valor da solução ótima do problema sobre a subárvore enraizada em i, atribuindo i à classe k e usando no máximo c componentes. $H(f,k,c) = d_{fk}$ para cada folha f.

Para cada vértice u não-raiz, definimos $P(u,k,c)$ como o valor da melhor solução do subproblema enraizado em u, mas dado que a classe de $p(u)$ já foi fixada em k e usando c componentes *a mais*, nessa subárvore u (c varia entre 0 e $\gamma - 1$ aqui). Por essa definição, u pode ser atribuído à classe k sem "gastos" adicionais de componentes.

2. **Iterações**

   $P(u,k,c)$ é obtido pela expressão:

$$P(u,k,c) = \min[\{H(u,k,c+1)\} \cup \{H(u,k',c) : \forall k' \neq k \in \{1,\ldots,K\}, \text{ se } c \geq 1\}]$$

   As soluções H de um vértice não-folha i são calculadas como:

$$H(i,k,c) = d_{ik} + \min_{c_u : \sum_{u \in S(i)} c_u = c-1} \sum_{u \in S(i)} P(u,k,c_u)$$

3. **Solução Ótima**

   Como antes, o valor da solução do problema é dado por $\min_k H(r,k,\gamma)$.

Aqui, para cada vértice i com $|S(i)| \geq 2$ temos duas "decisões": a atribuição de i e como distribuir o número de componentes disponíveis entre as subárvores enraizadas em i. Assim, após a escolha da classe de i, temos um problema clássico de alocação de recursos, que também pode ser resolvido otimamente por programação dinâmica.

Uma vez que os $P(u,k,c_u)$ já estejam calculados, resolver o subproblema de alocação para calcular $H(i,k,c)$ custa $O(c^2.|S(i)|)$. Como temos um número polinomial desses subproblemas, o algoritmo resultante ainda é polinomial para árvores. Na versão desse algoritmo para o PAgC, o subproblema de calcular $H(i,k,c_1,\ldots,c_K)$ é uma alocação de multirecursos, que custa $O(\prod_k c_k^2.|S(i)|)$. Assim, o PAgC é polinomial, para um K fixo, em árvores. Uma descrição detalhada do problema de alocação de um ou mais recursos, pode ser encontrada nos capítulos 1 e 2 de Bellman e Dreyfus [4].

É interessante notar que o PAgC em árvores é muito bem resolvido na prática pelas formulações que serão mostradas na próxima seção.

4.2 Formulações em Programação Inteira

O nosso objetivo é encontrar uma formulação adequada para ser usada em um algoritmo de *branch-and-bound*. As relaxações lineares dessas formulações fornecerão os limites duais necessários a esse tipo de algoritmo. Quanto mais fortes forem as formulações, melhores serão esses limites e menos nós da árvore de *branch-and-bound* terão de ser explorados. Por outro lado, geralmente as formulações mais fortes levam a PLs com um número maior

de variáveis e restrições, e portanto de resolução mais demorada. Em princípio, sempre devemos chegar a um compromisso entre a força e a facilidade computacional de uma formulação. Mas, depois de muitos anos de experiências práticas, sabe-se que, na maioria dos casos, só vale a pena trabalhar com formulações fortes, mesmo que elas levem a PLs mais difíceis (Nemhauser e Wolsey [30], por exemplo).

Freqüentemente, é possível encontrar formulações fortes que utilizam um número muito grande, até exponencial, de restrições. Isso não é necessariamente um problema, desde que estas possam ser **separadas** de modo eficiente, de preferência por algoritmos polinomiais exatos. Esse tipo de formulação pode ser incorporada numa variação do algoritmo de *branch-and-bound* conhecida por *branch-and-cut*.

Da mesma forma, às vezes é possível encontrar formulações fortes que utilizam um número muito grande de variáveis, que levam a algoritmos de *branch-and-price*. É necessário que essas variáveis possam ser **apreçadas** (priced) de forma eficiente para que o algoritmo funcione bem.

Nessa seção usamos as variáveis binárias de atribuição x, onde $x_{ik} = 1$ se e somente se o vértice i estiver atribuído à classe k. As formulações correspondentes têm a forma:

$$\text{Min} \quad \sum_{i \in V} \sum_{k=1}^{K} d_{ik} x_{ik} \qquad (1)$$

$$\text{Sujeito a} \quad \sum_{k=1}^{K} x_{ik} = 1 \qquad \forall i \in V \quad (2)$$

$$x \text{ é } \gamma\text{-Conexo}$$

As restrições (2) garantem que x é uma atribuição válida, mas temos que encontrar boas maneiras de forçar a γ-conexidade.

4.2.1 Formulações para a Versão do Limite Total γ

Assim como fizemos com os algoritmos de programação dinâmica da seção 4.1.2, começaremos falando dessa versão do problema. Novamente, queremos apresentar nossas idéias primeiro nesse caso mais simples, e posteriormente estendê-las para o PAgC.

Seja $C_k^n \subseteq C_k$, o n-ésimo componente conexo da classe k induzido por uma atribuição x. C_k^n pode ser representado alternativamente por uma das árvores geradoras do subgrafo de H induzido por esse componente, juntamente com uma identificação da classe k. O problema pode, então, ser reduzido a particionar os vértices de H em menos de γ árvores, mas de modo que todos os vértices de uma dessas árvores estejam atribuídos à mesma classe. Essa observação leva imediatamente à seguinte formulação:

Para cada aresta $e = (i, j) \in E$, e para cada classe k. $k = 1, \ldots, K$, definimos novas variáveis binárias z_e^k. Uma variável $z_e^k = 1$ se e somente se a aresta e pertence a uma árvore da classe k.

$$\left.\begin{array}{l} z_e^k \leq x_{ik} \\ z_e^k \leq x_{jk} \end{array}\right\} \quad \forall e = (i,j) \in E;\ k = 1,\ldots,K \quad (3)$$

$$\sum_{k=1}^{K} \sum_{e \in E} z_e^k \geq |V| - \gamma \quad (4)$$

$$\sum_{k=1}^{K} \sum_{e \in E(S)} z_e^k \leq |S| - 1 \quad \forall S \subset V,\ 1 < |S| \leq |V| \quad (5)$$

As restrições (3) ligam as variáveis x e z e obrigam cada árvore a ser formada por vértices da mesma classe. As restrições (5) impedem a existência de ciclos e assim garantem que z induz uma floresta. A restrição de cardinalidade (4) faz com que essa floresta seja composta por no máximo γ árvores. Temos então que (1-5), juntamente com as restrições de integralidade das variáveis x, é uma formulação completa para essa versão do problema. As restrições de eliminação de ciclos (5), semelhantes àquelas freqüentemente usadas no problema do caixeiro viajante, podem ser eficientemente separadas em tempo polinomial [31].

Essa formulação é fraca, mas pode ser bastante melhorada substituindo-se (5) por:

$$\sum_{e \in E(S)} z_e^k \leq \sum_{i \in S \setminus \{d\}} x_{ik} \quad \forall S \subset V,\ 1 < |S| \leq |V|;\ \forall d \in S;\ k = 1,\ldots,K \quad (6)$$

Essa é uma variação das chamadas "restrições de eliminação de ciclos generalizadas", às vezes usadas para formular algumas versões do problema de Steiner em grafos. O problema de separação correspondente pode ser resolvido em tempo polinomial (Lucena e Beasley (1992) [24]).

A figura 4.1 é um exemplo de uma solução fracionária que respeita (1-5), mas é cortada por (6). [1] Nessa instância $K = 2$ e $\gamma = 1$, ou seja, numa solução inteira todos os vértices devem estar em uma das duas classes. Vemos nessa figura que $\sum z_e^k = 7$ e portanto (4) é respeitada. (5) também é respeitada. Mas existem duas desigualdades de (6) violadas. Uma delas tem como S os três vértices mais à esquerda, d o vértice mais à esquerda e $k = 1$. A outra desigualdade violada está no triângulo superior. Adicionando-se (6), nessa instância particular com $\gamma = 1$, teremos uma solução inteira para qualquer função objetivo.

Mas mesmo com a formulação (1-4)(6) podemos ter soluções fracionárias. A figura 4.2 mostra uma tal solução.

[1] Usamos as seguintes convenções para apresentar graficamente as soluções fracionárias de algumas instâncias com $K = 2$: Um vértice i cheio representa $x_{i1} = 1$ e um vazio $x_{i2} = 1$. Vértices meio cheios (cinza) representam $x_{i1} = 0.5$ e $x_{i2} = 0.5$. Um valor na forma de fração, próximo a uma aresta e, representa z_e^1. Já um valor na forma decimal representa z_e^2. As variáveis de aresta não indicadas têm valor 0.

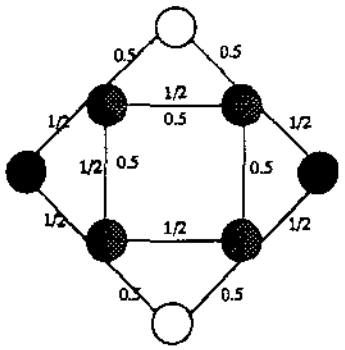


Figura 4.1: Solução fracionária de (1-5) ($K = 2$, $\gamma = 1$).

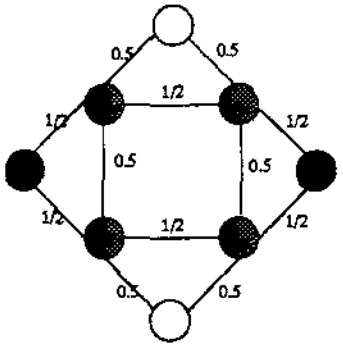


Figura 4.2: Solução fracionária de (1-4)(6) ($K = 2$, $\gamma = 2$).

4.2.2 Formulação F1

Dadas as boas características de (1-4)(6), gostaríamos de estender essa formulação para o PAgC.

Suponha que adicionamos K novos supervértices 0_k, um para cada classe. Cada supervértice é ligado por novos arcos direcionados $0_k i$, indo de 0_k para cada $i \in V$. Suponha que no máximo γ_k desses arcos possam estar ativos por supervértice 0_k. Os vértices sobre os quais incidem um arco ativo são chamados de **raízes**. Os superarcos ativos também serão chamados de arcos raízes.

Queremos mandar K diferentes tipos de fluxos dos supervértices para os vértices em V. Apenas o supervértice 0_k pode fornecer (a quantidade necessária de) unidades do fluxo k. Os fluxos só podem entrar no grafo original por um arco ativo. Dentro desse grafo, o fluxo k só pode passar por arestas incidentes aos vértices atribuídos à classe k. Para qualquer solução inteira x, sempre existe um fluxo que respeite essas restrições e que pode ser interpretado como uma partição de H em arborescências (árvores direcionadas). Cada arborescência está enraizada num vértice raiz e contém os vértices que recebem fluxo através dessa raiz. Como todos os vértices de uma arborescência devem obrigatoriamente estar atribuídos à mesma classe, x será γ-conexo.

Seja $y_{0_k i}$ uma variável binária para cada novo arco, tal que $y_{0_k i} = 1$ se e somente se

esse arco está ativo. Seja A o conjunto de arcos obtidos pela troca de cada aresta (i,j) em E por um par de arcos opostos (i,j) e (j,i). Sejam f_{ij}^k variáveis positivas para cada arco $(i,j) \in A$. f_{ij}^k representa a quantidade de fluxo k passando de i para j. Seja B_k um limite superior ao número de vértices que podem vir a ser atribuídos a classe k. B_k pode ser feito igual $|V|$, se um limite melhor não é conhecido.

$$\sum_{(i,j)\in\delta^+(i)} f_{ij}^k - \sum_{(j,i)\in\delta^-(i)} f_{ji}^k - f_{0_k i} + x_{ik} = 0 \quad \forall i \in V;\ k = 1,\ldots,K \qquad (7)$$

$$\sum_{i\in V} f_{0_k i} - \sum_{i\in V} x_{ik} = 0 \qquad k = 1,\ldots,K \qquad (8)$$

$$\left.\begin{array}{l} f_{ij}^k + f_{ji}^k \le B_k.x_{ik} \\ f_{ij}^k + f_{ji}^k \le B_k.x_{jk} \end{array}\right\} \quad \forall (i,j) \in E;\ k = 1,\ldots,K \qquad (9)$$

$$f_{0_k i} \le B_k.y_{0_k i} \qquad \forall i \in V;\ k = 1,\ldots,K \qquad (10)$$

$$\sum_{i\in V} y_{0_k i} \le \gamma_k \qquad k = 1,\ldots,K \qquad (11)$$

A formulação (1-2),(7-11), apesar de correta, é muito fraca. O limite dual fornecido por sua relaxação linear está próximo do limite trivial obtido atribuindo cada vértice i à classe k com mínimo d_{ik}. A principal razão para esse fraqueza é que o acoplamento das variáveis x e y, com os fluxos f, feito pelas restrições (9) e (10), é muito frouxo. Devido ao grande coeficiente B_k, um pequeno valor fracionário de x_{ik} é suficiente para permitir a passagem de muitas unidades de fluxo k através do vértice i .

Uma pequena melhoria pode ser conseguida reduzindo-se esse coeficiente para $\lceil B_k/2 \rceil$ em (9). Isso é possível, pois mesmo que numa solução ótima todos os possíveis B_k vértices de uma classe estejam em um único componente, fixando a raiz no "centro" do componente, obtemos uma solução equivalente sem precisar passar muitas unidades de fluxo f_{ij}^k por algum arco. Fazendo-se uma análise do grafo H de uma dada instância particular, pode-se aplicar outros argumentos desse tipo para reduzir-se ainda mais esses coeficientes.

Como a relaxação linear dessa formulação pode ser resolvida muito rapidamente, devido ao número relativamente pequeno de restrições e variáveis e a estrutura de rede que está presente, é tentadora a idéia de obter uma formulação mais forte seguindo esse modelo. Há diversos tipos de raciocínios para fortalecer a formulação, aplicáveis quando, num esquema de *branch-and-bound*, algumas das atribuições x já estão fixadas. Por exemplo, suponha que $\gamma_k = 1$ para alguma classe k. Se existe algum $x_{ik} = 1$, pode-se fixar $y_{0_k i} = 1$ e eliminar do modelo todas as demais variáveis $y_{0_k j}$ para $j \in V, j \neq i$. [2]

Experiências preliminares indicaram que essa abordagem até funciona bem para pequenas instâncias, mas falha em instâncias maiores. O problema é que o número de nós da árvore de *branch-and-bound* cresce muito rapidamente.

[2] O uso desse tipo de idéias, para fortalecer e/ou reduzir o tamanho de uma formulação, antes de se resolver a relaxação inicial, costuma ser chamada de **pré-processamento**. Já quando ela ocorre dentro do *branch-and-bound*, é conhecida como **fixação por implicação lógica**

A fim de obter uma formulação mais forte, especializamos os K fluxos em $K.|V|$ tipos de fluxo. Agora f^{dk} é um fluxo que vem do supervértice 0_k, mas só pode ser consumido no vértice destino d. Um vértice d atribuído à classe k consome 1 unidade de fluxo f^{dk}. Agora, no máximo uma unidade de um certo fluxo passa por algum arco do grafo. Isso permite um acoplamento muito mais justo das variáveis x e y com os fluxos f, o que nos dá a formulação (1-2),(12-17) .

$$\sum_{(i,j)\in\delta^+(i)} f_{ij}^{dk} - \sum_{(j,i)\in\delta^-(i)} f_{ji}^{dk} - f_{0_ki}^{dk} + x_{dk} = 0 \quad \forall i \in V;\ d = i;\ k = 1,\ldots,K \tag{12}$$

$$\sum_{(i,j)\in\delta^+(i)} f_{ij}^{dk} - \sum_{(j,i)\in\delta^-(i)} f_{ji}^{dk} - f_{0_ki}^{dk} = 0 \quad \forall i \in V; \forall d \neq i \in V;\ k = 1,\ldots,K \tag{13}$$

$$\sum_{i\in V} f_{0_ki}^{dk} - x_{dk} = 0 \quad \forall d \in V;\ k = 1,\ldots,K \tag{14}$$

$$\left.\begin{array}{l} f_{ij}^{dk} + f_{ji}^{dk} \leq x_{ik} \\ f_{ij}^{dk} + f_{ji}^{dk} \leq x_{jk} \end{array}\right\} \quad \forall (i,j) \in E; \forall d \in V; k = 1,\ldots,K \tag{15}$$

$$f_{0_ki}^{dk} \leq y_{0_ki} \quad \forall i \in V;\ k = 1,\ldots,K \tag{16}$$

$$\sum_{i\in V} y_{0_ki} \leq \gamma_k \quad k = 1,\ldots,K \tag{17}$$

Podemos melhorar ainda mais essa formulação, se notarmos que para cada solução inteira x sempre existe um fluxo em que há exatamente um arco da classe k com fluxo não-nulo entrando em cada vértice i atribuído à classe k, o que corresponde a uma arborescência. Introduzindo novas variáveis binárias y_{ij}^k, $\forall (i,j) \in A,\ k = 1,\ldots,K$ podemos levar isso em consideração e reforçar a formulação.

$$\left.\begin{array}{l} y_{ij}^k + y_{ji}^k \leq x_{ik} \\ y_{ij}^k + y_{ji}^k \leq x_{jk} \end{array}\right\} \quad \forall (i,j) \in E;\ k = 1,\ldots,K \tag{18}$$

$$f_{ij}^{dk} \leq y_{ij}^k \quad \forall (i,j) \in A; \forall d \in V;\ k = 1,\ldots,K \tag{19}$$

$$\sum_{(j,i)\in\delta^-(i)} y_{ji}^k + y_{0_ki} = x_{ik} \quad \forall i \in V;\ k = 1,\ldots,K \tag{20}$$

(1-2),(12-14),(16-20) é uma formulação forte o suficiente para fornecer bons limites duais. O problema é que agora temos um número excessivo ($O(|E|.|V|.K)$) de variáveis e restrições. Não é possível resolver nem a relaxação linear inicial dessa formulação, para qualquer instância de tamanho razoável. Felizmente, o teorema do fluxo máximo - corte mínimo nos fornece imediatamente uma formulação equivalente, substituindo os fluxos por restrições de cortes orientados sobre as variáveis y. Mostramos aqui essa formulação completa, para facilidade de referência, a chamaremos de F1.

$$
(F1)\begin{cases}
\text{Min} & \displaystyle\sum_{i\in V}\sum_{k=1}^{K} d_{ik}x_{ik} & & (1)\\
\text{Sujeito a} & \displaystyle\sum_{k=1}^{K} x_{ik} = 1 & \forall i \in V & (2)\\
& \left.\begin{array}{l} y^k_{ij} + y^k_{ji} \le x_{ik} \\ y^k_{ij} + y^k_{ji} \le x_{jk} \end{array}\right\} & \forall e=(i,j)\in E;\ k=1,\ldots,K & (3)\\
& \displaystyle\sum_{i\in V} y_{0_k i} \le \gamma_k & k=1,\ldots,K & (4)\\
& \displaystyle\sum_{(j,i)\in\delta^-(i)} y^k_{ji} + y_{0_k i} = x_{ik} & \forall i\in V:\ k=1,\ldots,K & (5)\\
& \displaystyle\sum_{(j,i)\in\delta^-(S)} y^k_{ji} + \sum_{i\in S} y_{0_k i} \ge x_{dk} & \forall S\subset V,\ 1<|S|\le|V|\ ; & (6)\\
& & \forall d\in S:\ k=1,\ldots,K &
\end{cases}
$$

A figura 4.3 mostra uma solução fracionária de F1. Uma seta com uma das extremidades soltas, entrando no vértice i, representa um superarco raiz $y_{0_k i}$.

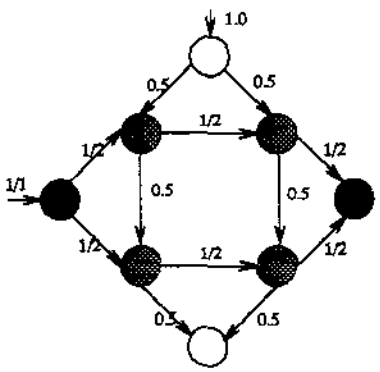


Figura 4.3: Solução fracionária de F1 ($K=2$. $\gamma_1=1$, $\gamma_2=1$).

As restrições (F1.6) têm o papel de substituir os fluxos f^{dk}. Seja $H^+ = (V \cup V(0_k), A \cup A(0_k, i))$ o grafo aumentado orientado, onde $V(0_k)$ é o conjunto de supervértices e $A(0_k, i)$ é o conjunto dos arcos que emanam desses supervértices. As restrições (F1.6) podem ser separadas achando-se o corte mínimo em H^+ que separa cada supervértice 0_k de cada vértice $d \in V$. As capacidades de cada arco são dadas pelos valores correntes das variáveis y correspondentes.

Assim, mesmo com um número exponencial de restrições, F1 pode ser embutida num algoritmo de *branch-and-cut*. A força de F1 será comprovada empiricamente, mas temos um resultado que nos permite comparar essa formulação com aquela apresentada para a versão do limite global γ de componentes, (1-4)(6). F1 pode ser adaptada para essa versão substituindo-se as restrições (F1.4) por:

$$\sum_{k=1}^{K} \sum_{i \in V} y_{0_k i} \leq \gamma \qquad \text{(F1.4')}$$

Proposição 4.2 *F1 (adaptada para a versão do limite global γ) é pelo menos tão forte quanto (1-4)(6).*

Prova: Mostraremos que, dada uma solução (x, y), possivelmente fracionária, de F1, podemos construir uma solução (x, z) de (1-4)(6).

Essa construção é muito simples: fazemos $z_e^k = y_{ij}^k + y_{ji}^k$. $\forall e = (i,j) \in E;\ k = 1, \ldots, K$. Resta agora verificar que (x, z) é factível.

Como x respeita (F1.2), também respeitará (2), pois essas restrições são idênticas. Da mesma forma, como y respeita (F1.3), z respeitará (3). Somando-se as $K.|V|$ restrições (F1.5), e aplicando-se (F1.2) obtemos:

$$\sum_{(i,j) \in E} \sum_{k=1}^{K} (y_{ij}^k + y_{ji}^k) + \sum_{i \in V} \sum_{k \in K} y_{0_k i} = |V|.$$

Essa expressão, juntamente com (F1.4') implica em

$$\sum_{(i,j) \in E} \sum_{k=1}^{K} (y_{ij}^k + y_{ji}^k) \geq |V| - \gamma,$$

logo z respeita (4).

Seja S um subconjunto de V, tal que $1 < |S| \leq |V|$ e k tal que $1 \leq k \leq K$. Somando-se as $|S|$ restrições (F1.5) correspondentes aos vértices em S obtemos:

$$\sum_{(i,j) \in E(S)} (y_{ij}^k + y_{ji}^k) + \sum_{(j,i) \in \delta^-(S)} y_{ji}^k + \sum_{i \in S} y_{0_k i} = \sum_{i \in S} x_{ik}.$$

Seja d um vértice qualquer em S. Subtraindo-se dessa expressão a restrição de (F1.6) correspondente ao conjunto S, ao vértice d e à classe k obtemos:

$$\sum_{(i,j) \in E(S)} (y_{ij}^k + y_{ji}^k) \leq \sum_{i \in S \setminus \{d\}} x_{ik},$$

logo z respeita (6).

□

Essa proposição mostra que realmente conseguimos estender (1-4)(6) para o PAgC, embora ao custo de um aumento no tamanho e na complexidade da formulação, pois enquanto a primeira formulação representava componentes por árvores diretamente no grafo original H, em F1 eles são representados por arborescências em H enraizadas nos supervértices 0_k.

4.2.3 Formulação F2

Os problemas encontrados no *branch-and-cut* sobre F1, que serão mostrados na seção 4.3, nos motivaram a também trabalhar com um outro tipo de formulação para o PAgC.

Seja L o conjunto dos vetores de incidência de todos os possíveis componentes conexos de H, juntamente com uma "etiqueta" identificando a classe do componente. Isso significa que para cada $l \in L$, $l_i = 1$ se e somente se o vértice i está no componente l e $l_{|V|+k} = 1$ se e somente se l for um componente da classe k. Assim, l é um vetor de tamanho $|V|+K$. O custo de um componente l da classe k é dado por $c_l = \sum_{i \in l} d_{ik}$. Podemos formular o PAgC como o seguinte *set-partitioning* com restrições adicionais de cardinalidade. Chamaremos essa formulação de F2.

$$(F2)\begin{cases} \text{Min} & \sum_{l \in L} c_l \lambda_l & & (1) \\ \text{Sujeito a} & \sum_{l \in L} l_i \lambda_l = 1 & \forall i \in V & (2) \\ & \sum_{l \in L} l_{|V|+k} \lambda_l \leq \gamma_k & k = 1, \ldots, K & (3) \\ & \lambda_l \in \{0,1\} & \forall l \in L & (4) \end{cases}$$

O enorme número de variáveis dessa formulação exige o uso de técnicas de geração de colunas. Com essas técnicas, podemos trabalhar com um pequeno número dessas colunas por vez, gerando novas colunas, por um algoritmo de apreçamento, sempre que necessário. Para obter as soluções inteiras buscadas, temos que embutir F2 num esquema de *branch-and-price*. Falaremos em detalhes desse algoritmo na seção 4.4.

Por enquanto queremos mostrar o seguinte resultado:

Proposição 4.3 *F2 é pelo menos tão forte quanto F1*

Prova: Apresentamos um procedimento para construir uma solução factível de F1, a partir de uma solução factível de F2. Estas soluções têm o mesmo valor da função objetivo.

PROC (ENTRADA: λ; SAÍDA: (x, y))
  $x \longleftarrow 0$;
  $y \longleftarrow 0$;
  Para cada $l \in L$ com $\lambda_l > 0$ {
    $k \longleftarrow$ a classe de l;
    Ache uma arborescência T geradora de l enraizada a partir de algum i em l;
    $y_{0_k i} \longleftarrow y_{0_k i} + \lambda_l$;
    para cada arco (i,j) de T { $y^k_{ij} \longleftarrow y^k_{ij} + \lambda_l$; }
    para cada vértice i de T { $x_{ik} \longleftarrow x_{ik} + \lambda_l$; }
  }

Vamos verificar que esse procedimento produz uma solução válida para F1. Por (F2.2) garantimos que x respeita (F1.2). Como uma variável y_{ij}^k só é aumentada de λ_l unidades, quando x_{ik} e x_{jk} também são aumentadas de λ_l unidades, (F1.3) será satisfeita. (F2.3) implica (F1.4). Como x_{ik} só é aumentada junto com $y_{0_k i}$ ou com algum y_{ji}^k, (F1.5) é respeitada. (F1.6) segue do fato que x_{ik} só é aumentada juntamente com um caminho (contido em T) ligando 0_k a i. Pela definição de c_l, o custo da solução permanece inalterado.

□

A técnica construtiva usada na demonstração tem a vantagem de prover um procedimento para converter as soluções de F2 para a uma solução de F1. Nas seções seguintes, veremos várias situações em que isso é desejável. Por exemplo, suponha que temos uma heurística para gerar soluções inteiras a partir de soluções fracionárias (x, y). Esse procedimento pode ser incorporado ao *branch-and-price* sobre F2, simplesmente convertendo-se as soluções λ correntes.

Mas vamos entender melhor a relação entre F1 e F2. Supondo que todos os γ_k são iguais a 1, F2 pode ser vista como a reformulação obtida aplicando-se a decomposição Dantzig-Wolfe sobre F1, mantendo-se (F1.2) no problema mestre e dividindo-se as demais restrições em K subproblemas independentes, cada um correspondendo a uma classe. Além disso, forçamos a integralidade dos subproblemas.

O poliedro em (x_k, y_k), possivelmente fracionário, correspondente ao subproblema da classe k é o seguinte:

$$
(P_{sub}^k)\begin{cases}
\left.\begin{array}{l} y_{ij}^k + y_{ji}^k \le x_{ik} \\ y_{ij}^k + y_{ji}^k \le x_{jk} \end{array}\right\} & \forall e = (i,j) \in E & (1)\\
\sum\limits_{i \in V} y_{0_k i} \le 1 & & (2)\\
\sum\limits_{(j,i) \in \delta^-(i)} y_{ji}^k + y_{0_k i} = x_{ik} & \forall i \in V & (3)\\
\sum\limits_{(j,i) \in \delta^-(S)} y_{ji}^k + \sum\limits_{i \in S} y_{0_k i} \ge x_{dk} & \forall S \subset V, 1 < |S| \le |V| \,; \forall d \in S & (4)\\
0 \le x_{ik} \le 1 & \forall i \in V & (5)
\end{cases}
$$

Cada solução x_k inteira, tal que existe (x_k, y_k) pertencente ao poliedro acima, corresponde à uma coluna da classe K, pois essas soluções inteiras definem um componente conexo em H. A restrição (F2.2) corresponde à restrição (F1.2) do problema mestre. (F2.3) são as restrições de convexidade sobre cada um dos subproblemas.

Esse tipo de reformulação, em que se faz uma decomposição e se força a integralidade em alguns dos subproblemas gerados, é conhecida como **convexificação parcial**. Um resultado de Geoffrion (1974) [13] garante que se o poliedro de algum desses subproblemas não for inteiro (que é o nosso caso), a reformulação obtida será estritamente mais forte, ou seja, o limite dual obtido será obrigatoriamente melhor em alguma instância.

Caso exista algum $\gamma_k > 1$, F2 não é exatamente a formulação obtida pela convexificação parcial de F1 nos moldes acima, pois nesses casos, uma coluna da classe k seria formada pelos vértices correspondentes a subgrafos de H formados por até γ_k componentes. Em F2, mesmo quando $\gamma_k > 1$, as colunas da classe k representam subgrafos formados por um único componente conexo. Isso significa que nesses casos, fizemos uma nova decomposição do subproblema k em γ_k subproblemas independentes. Mesmo assim, ao se resolver esses subproblemas à integralidade estamos fazendo uma convexificação parcial de F1, e portanto o resultado de Geoffrion ainda se aplica (optamos por trabalhar sempre com apenas um componente por coluna, a fim de facilitar a resolução do subproblema de apreçamento).

Na figura 4.4 observamos uma solução fracionária de F2, equivalente à solução fracionária de F1 mostrada na figura 4.3. As linhas fechadas pontilhadas indicam os dois componentes formados por vértices da classe 1. As linhas fechadas cheias mostram os dois componentes da classe 2. Todos esses componentes têm $\lambda_l = 0.5$. Isso quer dizer, que nesse caso, a convexificação parcial não conseguiu melhorar o limite dual obtido.

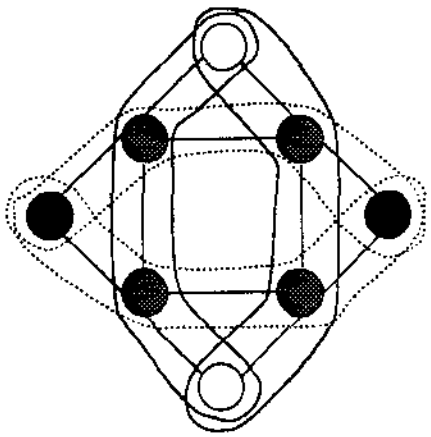


Figura 4.4: Solução fracionária de F2 ($K = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

Já a figura 4.5, mostra uma solução fracionária de F1 que não pode ser convertida numa solução equivalente de F2. Nesse caso, a convexificação parcial fará com que se obtenha um limite dual melhor.

Está claro que a formulação F2 pode ser estritamente mais forte que F1. Entretanto, para a nossa grande surpresa, em todas as dezenas de instâncias de teste descritas na seção 4.3.3, o limite dual obtido ao se resolver a relaxação linear inicial de F2 foi exatamente igual ao limite obtido por F1 ! Ou seja, pelo menos com o tipo e com o tamanho de instâncias que utilizamos, F2 não costuma ser mais forte do que F1.

Podemos especular um pouco sobre as razões desse comportamento:

a) O poliedro P^k_{sub} já é uma boa aproximação ao poliedro inteiro dos subproblemas,

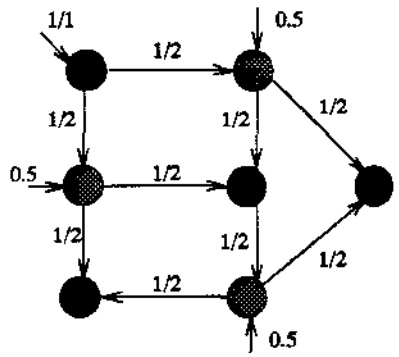


Figura 4.5: Solução fracionária de F1 ($K = 2$. $\gamma_1 = 1$, $\gamma_2 = 2$).

pelo menos em instâncias típicas relativamente pequenas. Assim, na prática pode não haver um ganho significativo em convexificar (integralizar) esse poliedro.

b) A formulação F1 já é uma formulação razoavelmente forte para o PAgC, que não deixa muito espaço para melhorias, talvez justamente por conter as restrições de P^k_{sub}. A fraqueza de F1, que às vezes leva a soluções fracionárias, está relacionada a uma indevida "mistura" (combinação convexa) entre soluções parciais inteiras de cada classe (os (x_k, y_k) com x_k inteiro). A figura 4.3 mostra uma dessas típicas soluções fracionárias. Assim, a introdução de novas restrições envolvendo apenas variáveis de uma mesma classe k (o equivalente a convexificar P^k_{sub}), não costuma melhorar a formulação. Para haver melhoria, seriam necessárias restrições envolvendo variáveis de classes diferentes.

c) As soluções fracionárias (x_k, y_k) de P^k_{sub} tendem a deixar "buracos", ou seja vértices i com x_{ik} fracionário, mais ou menos cercados por vértices j com $x_{jk} = 1$. Mas (F1.2) exige que esses buracos sejam "preenchidos", sendo atribuídos fracionariamente outras classes. A figura 4.5 é um exemplo disso. Aqui, como $\gamma_2 = 2$, pudemos usar as raízes para conectar os vértices fracionários da classe 2. Mas normalmente, as raízes são um recurso escasso que não vale a pena ser desperdiçado dessa maneira. Nessa instância, se fizermos $\gamma_2 = 1$, F1 será equivalente à F2, para quaisquer valores da função objetivo.

4.3 O Algoritmo de *Branch-and-Cut* sobre F1

4.3.1 Comentários Gerais

Apesar de os subproblemas de separação poderem ser resolvidos de forma eficiente, há dois gargalos que dificultam a rápida resolução de uma relaxação linear de F1 e que portanto comprometem a eficiência do *Branch-and-Cut* sobre F1.

1. **O tamanho dos PLs** - O número relativamente grande de variáveis e restrições de F1 leva a PLs um tanto quanto caros.

2. **O número de iterações** - Separar (F1.6) pode demandar a solução de muitos desses PLs. O problema é que para uma dada solução x, que é o que realmente nos interessa, correspondem inúmeros possíveis valores das variáveis auxiliares y. Seja $F1^p$ o PL formado pelas restrições (F1.1-F1.5) além de um subconjunto das desigualdade de (F1.6) já separadas na iteração p. Seja (x, y) uma solução ótima desse PL. Suponha que adicionamos novas desigualdades violadas à formulação e resolvemos o novo PL $F1^{p+1}$. Freqüentemente, principalmente em instâncias grandes, obtemos uma solução da forma (x, y'). Ou seja, essa iteração não causou melhoria da função objetivo. Às vezes, várias iterações são necessárias para sairmos dessa solução x.

   Esse tipo de comportamento, quando a adição de desigualdades causa pouca melhoria na função objetivo, muito comum em algoritmos de *branch-and-cut*, é conhecido como *tailing-off*. [3] Quando o *tailing-off* ocorre ao se adicionar novas classes de desigualdades válidas, não incluídas na formulação básica, alguns autores recomendam suspender a separação dessas classes de desigualdades e ir direto para o *branching* (por exemplo, Padberg e Rinaldi (1991) [31]). Só que isso não é possível no nosso caso, pois o *tailing-off* ocorre na adição de desigualdades da própria formulação, necessárias para a obtenção de soluções inteiras factíveis. Além disso, esse comportamento pode ocorrer muito cedo, logo ao se resolver a relaxação inicial de F1.

Boa parte do trabalho feito para melhorar a eficiência do *Branch-and-Cut* sobre F1, teve como objetivo reduzir o efeito desses dois gargalos.

Por exemplo, pode-se notar que as restrições (F1.3) poderiam ser retiradas de F1 sem enfraquecer a formulação. Usando a mesma argumentação usada na proposição 4.2, podemos ver que uma restrição de (F1.3) pode ser obtida a partir de (F1.5) e de uma restrição de (F1.6). As restrições (F1.3) na verdade são um caso particular de (6) (página 37), quando $|S| = 2$.

Mas optamos por manter (F1.3) dentro da formulação, pois verificamos que essas restrições reduzem significativamente o número de iterações necessárias para a resolução da relaxação linear de F1. Não seria conveniente manter as restrições equivalentes de (F1.6), pois o número de coeficientes não-nulos nos PLs aumentaria muito. De qualquer maneira, pode-se manter apenas algumas dessas restrições dentro dos PLs, pois é muito fácil separar (F1.3).

[3] O nome "*tailing-off*" vem do fato que esse efeito é comum quando já se está próximo da solução do subproblema de um nó.

4.3.2 Implementação do *Branch-and-Cut*

Dado que temos a formulação F1 e sabemos como separar as restrições F1.6, é imediato que já temos um algoritmo de *branch-and-cut* para o PAgC. Isso é verdade. Só que esse algoritmo básico nem sempre vai ter um bom desempenho prático.

Inicialmente, construímos um protótipo que implementava um *branch-and-cut* básico, a partir do zero, usando apenas o pacote CPLEX 3.0 [7] para a resolução de PLs. Esse protótipo foi muito útil em nossas experiências, mas logo ficou claro que para a resolução de instâncias maiores teríamos que melhorar esse algoritmo básico, incorporando uma série de funcionalidades, que seriam de demorada e difícil codificação.

Decidimos então usar o pacote ABACUS (Thienel (1995) [41]), recentemente liberado para uso acadêmico. Esse pacote implementa muitas das funções comuns a algoritmos de *branch-and-cut* e *branch-and-price* em geral. Usando esse pacote, podemos fazer experiências com diversas variantes do algoritmo, apenas mudando um arquivo de configuração. Por exemplo: podemos escolher entre algumas estratégias de branching, mudar o mecanismo de gerenciamento dos programas lineares, ativar a fixação de variáveis por custos reduzidos, etc. Dessa forma pode-se tentar encontrar uma configuração mais adequada ao problema em questão.

Mas mesmo com o ABACUS, ainda tivemos que escrever muito código, pois a melhoria obtida apenas ajustando-se o arquivo de configuração é limitada. As técnicas mais efetivas para se melhorar um *branch-and-cut* são justamente as que exploram a estrutura particular de um problema. Naturalmente essas técnicas devem ser implementadas pelo usuário do pacote. Jünger, Reinelt e Thienel (1995) [20]; Ferreira e Wakabayashi (1996) [11]; Lucena e Beasley (1996) [25] contêm referências atualizadas sobre as técnicas mais utilizadas para se melhorar o desempenho prático de um *branch-and-cut*.

Vamos descrever alguns pontos do algoritmo que implementamos:

Pré-Processamento

Para cada vértice i, podemos eliminar da formulação uma (e apenas uma) variável de arco raiz $y_{0_k i}$. Essa raiz deve corresponder a uma classe $kMax(i)$ tal que $d_{i\,kMax(i)}$ é máximo, ou seja, uma das classes mais desvantajosas para se atribuir i.

Seja (x, y) uma solução ótima inteira qualquer. Sempre será possível encontrar uma solução inteira (x', y') de mesmo custo, tal que nenhum vértice i seja raiz da classe $kMax(i)$. Isso não quer dizer que numa solução ótima não possa haver vértices i atribuídos à classe $kMax(i)$. Isso pode acontecer, desde que i pertença a um componente conexo maior da classe $kMax(i)$. Pelo menos um dos vértices j desse componente terá $kMax(j) \neq kMax(i)$ e portanto poderá funcionar como raiz desse componente.

Esse pré-processamento reduz um pouco, tanto o tamanho dos PLs, quanto o número de iterações.

Política de Separação

Esse é um ponto crítico na eficiência do algoritmo. A questão é: quantas e quais desigualdades violadas de (F1.6) devem ser geradas a cada iteração ? Incluir apenas uma desigualdade por vez é muito pouco, pois o número de iterações torna-se enorme. Por outro lado, pode haver um número exponencial de desigualdades (F1.6) violadas por uma solução (x, y), pois podem existir muitos conjuntos S tal que $\delta^-(S)$ é um corte mínimo separando dois vértices s e t.

Colocar mais desigualdades aumenta o tamanho dos PLs, mas pode reduzir o número de iterações. Depois de algumas experiências, optamos pelo seguinte compromisso:

Para cada i e cada k com $x_{ik} > 0$, achamos um corte mínimo $\delta^-(S)$ entre 0_k e i passando por arcos com capacidade y_k. Caso o valor desse corte seja menor que x_{ik}, adicionamos uma desigualdade correspondente à S, i e k. Observamos que é preferível adicionar o corte mínimo "mais próximo" de i, ou seja o correspondente a um conjunto S (contendo i) minimal.

Gerenciamento das Restrições nos PLs

Para evitar o crescimento excessivo no tamanho dos PLs. é possível retirar as desigualdades de (F1.6) que não estão correntemente ativas, ou seja, satisfeitas na igualdade pela solução corrente. Ao se retirar uma desigualdade do PL há duas opções: simplesmente descartá-la ou colocá-la num *pool*. O *pool* é uma estrutura de dados que armazena desigualdades. Ele deve permitir verificar se alguma dessas desigualdades voltou a ser violada, a um baixo custo (em comparação com o custo de se usar o algoritmo de separação normal).

No nosso caso, as experiências mostraram que a remoção das desigualdades inativas. melhora um pouco o algoritmo. Na verdade, esperávamos que a melhoria fosse maior. Mas verificamos que, pelo menos com a política de separação descrita acima, uma grande parte (80% em alguns casos) das desigualdades (F1.6) removidas, voltam a ser violadas em alguma etapa do algoritmo.

Mesmo assim, dado que a separação de (F1.6) é eficiente, verificamos que não há vantagem significativa em se usar um *pool*.

Fixação por Implicação Lógica

Sempre que no decorrer da execução do algoritmo, existir uma variável x_{ik} fixada em 1, (por *branching*, por custo reduzido ou por qualquer outro motivo), pode-se fixar uma (e apenas uma) raiz da classe k, fazendo-se $y_{0_k i} = 1$. Caso $\gamma_k = 1$, pode-se fixar todas as demais raízes da classe k em 0.

Essa fixação pode reduzir significativamente o *tailing-off* associado à separação de (F1.6) em alguns dos nós da árvore de *branch-and-bound*, e portanto o número de iterações desses nós, principalmente no caso em que $\gamma_k = 1$.

Regra de *Branching*

Optamos por usar o tradicional *branching* sobre variáveis binárias. No caso, escolhemos um x_{ik} fracionário e geramos dois subproblemas, um com $x_{ik} = 0$ e outro com $x_{ik} = 1$.

Note que devido à restrição (F1.2), fixar $x_{ik} = 1$ implica em fixar os demais x'_{ik} em 0. Isso pode levar a um desbalanceamento dos subproblemas, se K for grande. Nesse caso, seria melhor dividir as classes em dois conjuntos A e B de forma a que $\sum_{k \in A} x_{ik}$ seja fracionário. Em um subproblema os x_{ik} em A seriam fixados em 0. No outro subproblema os em B seriam fixados em 0. Talvez por trabalharmos com um número relativamente pequeno de classes, é pouco comum encontrar vértices i divididos em mais de duas classes numa solução fracionária. Dessa forma, optamos por fazer o *branching* simples sobre uma única variável x_{ik}.

Quanto à escolha da variável de *branching*, começamos usando o x_{ik} com valor mais próximo de 0.5.

Houve uma melhoria, dando-se preferência a variáveis da classe k que ainda não tivessem nenhum vértice fixado em 1 no nó corrente do *branch-and-bound*. Uma razão para isso, é que para cada classe com um vértice i fixado em 1, pode-se aplicar o pré-processamento descrito acima. A escolha da variável era feita ponderando-se esses dois critérios.

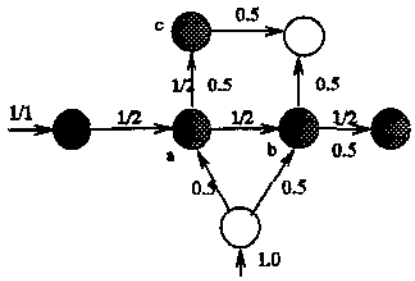


Figura 4.6: Solução fracionária antes do *branching* ($K = 2$, $\gamma = 1$).

Uma nova melhoria foi obtida usando-se também um terceiro critério: dá-se preferência a variáveis correspondentes a vértices i "atravessados por fluxos" de mais de uma classe, ou seja, com variáveis do tipo y_{ij}^k maiores do que 0, para diferentes k, na solução corrente. A figura 4.6 mostra uma solução fracionária de F1. Os vértice rotulados de a e b são atravessados por fluxos de mais de uma classe. O vértice rotulado de c não. Nesse caso, fazer o *branching* sobre a ou b leva a dois subproblemas com solução inteira, para qualquer função objetivo. O mesmo não é verdade ao se fazer o *branching* sobre c, como mostra a figura 4.7.

Após a seção 4.5, poderemos entender melhor porque esse critério costumar funcionar.

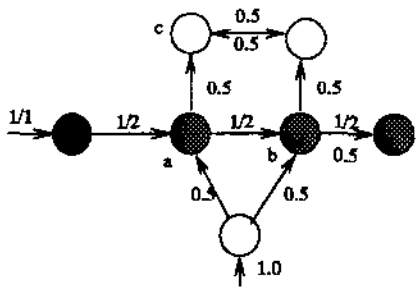


Figura 4.7: Solução fracionária após *branching* sobre c ($K = 2$, $\gamma = 1$).

Heurísticas Primais

Um algoritmo de *branch-and-bound* necessita de limites primais, a fim de cortar os ramos da árvore de *branching* que garantidamente não contêm a solução ótima. Esses limites primais normalmente são dados pelas soluções inteiras encontradas pelo próprio algoritmo, em alguns de seus subproblemas.

Entretanto, pode ocorrer que o algoritmo não escolha a melhor ordem de exploração dos subproblemas, de forma que bòas soluções inteiras demorem a ser encontradas. Nesses casos, pode-se perder muito tempo explorando-se ramos ruins.

Uma das maneiras de evitar esse comportamento, é tentar fornecer ao algoritmo bons limites primais calculados por outros meios.

As tentativas de construir boas heurísticas primais que usem diretamente os dados que definem uma instância, o grafo H, os d_{ik} e γ, não foram muito bem sucedidas.

Um modelo de heurística gulosa:

1. Para cada classe k escolha γ_k vértices i com baixo d_{ik} para serem as raízes dos componentes. É conveniente evitar a concentração de raízes da mesma classe na mesma região do grafo. Essa raízes serão atribuídas à classe k.

2. Seja D o conjunto de pares (i, k) formados por cada vértice i ainda não atribuído, mas adjacente à um vértice já atribuído à classe k. Encontre o par (i, k) em D com menor d_{ik} e atribua i à classe k.

3. Repita o passo 2 até que todos os vértices já estejam atribuídos.

Sem dúvida é possível melhorar essa heurística, mas os resultados preliminares obtidos não nos encorajaram a trabalhar nesse sentido. Uma heurística gulosa bem melhor foi obtida a partir das soluções fracionárias x de F1 obtidas em cada subproblema da árvore de *branching*. Essa heurística segue o modelo anterior, mas usa os dados mais "globais" fornecidos pelas soluções fracionárias de F1.

1. Seja C_k um componente conexo maximal de H formado por vértices com $x_{ik} = 1$. Escolha os γ_k componentes C_i com maior número de vértices. Esses componentes

funcionarão como "raízes" da classe k. Todos os vértices dos componentes escolhidos já podem ser atribuídos à classe k.

2. Seja D o conjunto de pares (i,k) formados por cada vértice i ainda não atribuído, mas adjacente à um vértice já atribuído à classe k. Encontre o par (i,k) em D com maior x_{ik} e atribua i à classe k. Em caso de empate, escolhemos o par (i,k) que maximize a soma dos $y_{ij}^k + y_{ji}^k$, para os vértices j ainda não atribuídos adjacentes à i.

3. Repita o passo 2 até que todos os vértices já estejam atribuídos.

Na maioria das instâncias, essa heurística de arredondamento das soluções fracionárias tem uma modesta influência no tempo de execução. Mas ela tem um importante papel, fazendo com que o algoritmo seja bem mais robusto, evitando a degradação do desempenho nos piores casos.

4.3.3 Resultados Computacionais

A Geração das Instâncias de Teste

O principal ponto, que diferencia os tipos de instâncias que geramos, é a relação entre os valores de d_{ik} e o grafo H.

- **Instâncias com Estrutura** - Tentamos reproduzir a estrutura que espera-se encontrar ao se usar o PAgC como segunda etapa do nosso método de classificação. Aqui, sempre usamos como grafo H uma grade quadrada $M \times M$. Cada um dos $M^2 = N$ objetos é associado a um vértice com "coordenadas" (X,Y); $1 \leq X \leq M$, $1 \leq Y \leq M$, dentro da grade. Cada objeto é associado, também, a um conjunto de 5 atributos. Três deles são valores aleatórios inteiros, obtidos de uma distribuição uniforme no intervalo de 1 a M. Os outros dois, recebem os valores de X e de Y. As dissimilaridades d_{ij} entre cada par de objetos são calculadas como a distância euclidiana (em 5 dimensões) entre os atributos de i e de j.

  Nesse momento, temos uma instância de classificação em que existe uma certa correlação entre a posição dos objetos em H e as suas dissimilaridades. Resolvemos esse problema de classificação pelo K-medóide, obtendo os K representantes. Agora temos os d_{ik} para o PAgC.

- **Instâncias Randômicas** - Aqui, usamos um grafo H qualquer e para cada d_{ik}, atribuímos um valor aleatório inteiro, retirado de uma distribuição uniforme no intervalo de 1 a 30. Não há portanto, a menor correlação entre os d_{ik} e o grafo H.

Usamos aqui dois tipos de grafo: grades quadradas e grafos planares aleatórios gerados pela função *random_planar_graph* oferecida pela biblioteca LEDA (Näher e Uhrig [29]).

Mesmo em outras aplicações do PAgC, que não como segunda etapa do nosso método (como a descrita na página 30), é bastante razoável supor a existência de alguma correlação entre os os d_{ik} e H. Mas optamos por trabalhar principalmente com as instâncias randômicas pelas seguintes razões:

a) As instâncias randômicas são bem mais difíceis e antecipam as dificuldades que vão ocorrer em instâncias grandes com estrutura. Por exemplo, um certo tipo de solução fracionária que já costuma aparecer em instâncias randômicas de 25 vértices, pode só se manifestar em instâncias com estrutura de mais de 70 vértices. É muito mais fácil para nós, analisar manualmente (a fim de melhorar o algoritmo) esse tipo de solução num grafo pequeno.

b) Os algoritmos desenvolvidos sobre essas instâncias difíceis, provavelmente serão mais robustos e aplicáveis a um PAgC qualquer. Por outro lado, um algoritmo desenvolvido sobre instâncias com um tipo particular de estrutura tenderia por "seleção natural" a se especializar nesse tipo de estrutura.

Resultados Computacionais

Os testes foram realizados numa estação de trabalho SPARC 1000 com o sistema SunOS 5.5.

Cada uma das tabelas abaixo mostra os resultados para um certo grupo de instâncias, caracterizadas pelo tipo de instância, randômica ou com estrutura, e pelo grafo H usado. Cada linha dessas tabelas mostra os resultados obtidos por um subgrupo de instâncias de mesmo tamanho, para diferentes valores de γ_k (sempre todos os γ_k são iguais). Para cada subgrupo e cada valor de γ_k, foram rodadas 5 instâncias diferentes. **Tm** é o tempo médio de cpu, em segundos, dessas rodadas. **Tmd** é o tempo da instância particular que obteve o tempo mediano. Essa instância mediana pode ser considerada a mais representativa dentre as testadas. Assim fornecemos alguns dados adicionais sobre essa instância particular. **Nós** indica o número de nós de *branch-and-bound* da instância mediana. Grandes valores dessa coluna indicam que limite dual fornecido por F1 não foi muito bom. **PLs** indica o número de PLs que tiveram que ser resolvidos. Grandes valores dessa coluna, em relação a Nós, indicam a ocorrência de *tailing-off*.

O tempo de cpu de uma rodada foi limitado a 1800 segundos. Dessa forma, em alguns casos, não foi possível calcular a média de tempo Tm. Mas pode ser possível calcular a mediana Tmd, caso a maioria das rodadas não estoure o tempo.

Tabela 4.1: Instâncias randômicas, num grafo H de grade quadrado

$\|V\|$	K	$\gamma_k = 1$				$\gamma_k = 2$				$\gamma_k = 3$			
		Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs
25	3	9	6	3	16	2	1	1	6	1	1	1	2
36	3	71	56	23	57	5	4	1	7	3	3	1	9
49	3	-	216	19	109	176	39	9	30	28	24	7	14
64	3	-	-	-	-	285	150	17	59	55	67	7	23

Tabela 4.2: Instâncias randômicas, num grafo H obtido por *random_planar_graph*

$\|V\|$	K	$\gamma_k = 1$				$\gamma_k = 2$				$\gamma_k = 3$			
		Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs
25	3	6	5	1	12	2	1	3	13	1	1	1	2
36	3	63	66	21	83	7	6	3	15	3	2	1	6
49	3	282	333	39	269	123	29	3	18	59	12	3	13
64	3	-	-	-	-	288	253	11	116	99	82	13	33

Tabela 4.3: Instâncias com estrutura, num grafo H de grade quadrado

$\|V\|$	K	$\gamma_k = 1$				$\gamma_k = 2$				$\gamma_k = 3$			
		Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs
100	5	253	148	3	30	171	70	1	39	105	29	1	7

Alguns comentários sobre esses resultados:

- Claramente, os casos mais difíceis de ser resolvidos são quando os γ_k são iguais a 1. Se o PAgC vier de uma aplicação de classificação, essa seria a conexidade estrita. Parece que a idéia de se flexibilizar a conexidade leva a problemas computacionalmente mais fáceis. Isso é intuitivo, pois quanto mais estrita for a conexidade, mais teremos que forçar a classificação para longe da solução do problema irrestrito (que no PAgC é a solução, provavelmente não-factível, obtida ao se atribuir cada vértice à classe mais barata).

- Os problemas com estrutura são bem mais fáceis de serem resolvidos do que os aleatórios. Isso já era esperado.

- Os dois tipos de grafos usados, o de grade e os obtidos por *random_planar_graph*, levam a problemas de dificuldade mais ou menos equivalente. Através de algumas experiências não descritas aqui, descobrimos que as seguintes condições em H facilitam o PAgC:

  1. Grafos H muito esparsos. No caso extremo, temos a árvore. Nesses casos, F1 é extremamente forte e resolve o problema rapidamente.
  2. Grafos H muito densos. Aqui, há muitos caminhos conectando os vértices e a solução obtida pouco se afasta da solução irrestrita.
  3. Grafos H não-planares. Parece que as conexões entre vértices "distantes" que podem existir nesses grafos, facilitam o problema.
  4. Grafos H pouco regulares, ou seja, com alguns poucos vértices com elevado grau e a maioria dos vértices com grau baixo (grau menor que 3).
  5. Grafos H com separadores. Um vértice v é separador se $H - v$ é desconexo. Os separadores, de certa forma, decompõem naturalmente o problema em subproblemas menores.

  Os grafos de grade quadrados e os grafos obtidos por *random_planar_graph* (pela maneira que são construídos) não apresentam nenhuma dessas condições facilitadoras.

4.3.4 Uma Heurística Baseada em F1

A principal causa do *tailing-off* relacionado à separação de (F1.6), está na grande multiplicidade de valores de y que podem corresponder a uma dada solução x. Observamos entretanto, que essa multiplicidade e portanto o *tailing-off*, podia ser bastante reduzida.

caso algumas das variáveis de raiz $y_{0_{ki}}$ estivessem fixadas em 1. Aliás, isso é feito durante a fase de *branching*, na fixação por implicação lógica descrita anteriormente.

Mas, suponhamos que resolvemos F1 desde o início, com todas raízes $y_{0_{ki}}$ fixadas. Para cada classe k, escolhemos γ_k raízes da classe k para serem fixadas em 1. As demais raízes são fixadas em 0. A solução desse novo problema, mais fácil de ser resolvido, é uma solução heurística para o PAgC.

Não há muita chance de se obter soluções muito boas dessa forma, se as raízes forem escolhidas ao acaso. Mas, em instâncias com estrutura, é possível identificar (manualmente ou com ajuda de algum procedimento) regiões de H com predominância de alguma classe k, ou seja, baixos valores relativos de d_{ik}. Aliás, quando o PAgC é aplicado à classificação, conforme descrito no capítulo 3, é de se supor que tais regiões existam. Essa regiões são candidatas naturais para receberem uma raiz da classe k.

Essa heurística funciona bem nesses casos, porque a escolha das raízes não é crítica. Seja C_k^n o n-ésimo componente conexo da classe k na solução ótima do PAgC. Qualquer escolha de raízes em que cada C_k^n receba uma raiz da classe k levará a esse ótimo. Mesmo se errarmos algumas dessas escolhas, ainda temos a chance de obter uma boa solução aproximada. Supondo que os valores de γ_k são fixos, a qualidade da aproximação tende a crescer com o número de vértices em H, pois os C_k^n ficam maiores. Assim, essa heurística é particularmente interessante quando todos os γ_k são 1 e $|V|$ é grande, que como vimos, são justamente os casos mais difíceis.

Fizemos experiências computacionais com uma maneira muito simples de escolher as raízes: para cada classe k, o vértice i com mínimo d_{ik} será a raiz da classe k (Se a instância vier do problema de classificação, esse i será o representante). Os resultados obtidos com as mesmas instâncias descritas na tabela 4.3 são mostrados a seguir. Esses resultados são comparados com a resolução exata dessas instâncias feita anteriormente. Todos os tempos **T** são em segundos de cpu.

Tabela 4.4: Instâncias com estrutura,$|V|$=100, K=5, γ_k=1, grafo H de grade quadrada

Instância	Heurística				Exato			
	T	Sol.	Nós	PLs	T	Sol.	Nós	PLs
1	20	3920	3	6	148	3920	3	30
2	21	4025	1	3	66	4025	1	20
3	48	4156	1	11	174	4156	1	38
4	196	3943	17	30	777	3943	21	125
5	39	4088	1	11	102	4088	1	29

Em todos esses problemas grandes com estrutura, conseguimos obter uma solução ótima num tempo bastante reduzido. Com essa heurística, pudemos até achar uma solução

para uma instância com estrutura de 400 vértices e 5 classes, em 50 minutos. Há indícios de que essa solução é ótima, mas não conseguimos provar isso num tempo razoável.

É interessante notar que os limites duais obtidos com a fixação das raízes não são significativamente mais fortes do que aqueles obtidos normalmente por F1. O ganho de desempenho realmente se deve à redução do *tailing-off*.

4.4 O Algoritmo de *Branch-and-Price* sobre F2

4.4.1 O Subproblema de Apreçamento

Devido ao número exponencial de variáveis em F2, temos que usar uma abordagem de geração de colunas, para resolver sua relaxação linear. O PL restrito a um pequeno conjunto dessas variáveis, que estão sendo correntemente consideradas é chamado de **mestre**. Sejam π_i e μ_k os valores correntes das variáveis duais associadas às restrições (F2.2) e (F2.3) respectivamente. O subproblema de apreçamento para cada classe k pode ser expresso como:

$$\begin{aligned} \text{Min} \quad & \sum_{i \in V} (d_{ik} - \pi_i) x_i - \mu_k & & (21) \\ \text{Sujeito a} \quad & x \text{ é conexo} & & (22) \\ & x_i \in \{0,1\} & \forall i \in V, & (23) \end{aligned}$$

onde $x_i = 1$ se e somente se o vértice i pertence ao componente conexo. Quando a solução ótima de todos esses K subproblemas for igual à 0, sabemos que a solução primal do mestre é ótima. Por outro lado, se algum desses problemas tiver uma solução negativa, podemos adicionar a correspondente coluna ao mestre e continuar a otimização.

O subproblema de apreçamento equivale ao seguinte problema combinatório:
Subgrafo Conexo de Mínimo Peso (SCMP): *Dado um grafo $G = (V, E)$ e pesos p_i para cada um dos vértices $i \in V$, encontre um subgrafo conexo S que minimize $\sum_{i \in S} p_i$.*

Se todos os pesos são positivos ou negativos, o SCMP é trivial. Caso contrário, queremos selecionar um certo número de vértices com peso negativo e usar vértices positivos apenas quando necessário para conectá-los. Vértices negativos adjacentes podem ser contraídos a um único vértice com peso igual a soma dos seus pesos individuais, pois qualquer solução ótima conterá todos ou nenhum desses vértices. Após todas essas possíveis contrações teremos um grafo com alguns vértices negativos, separados por um número, geralmente maior, de vértices positivos. Essa estrutura pode ajudar a resolver o SCMP na prática.

Mas o SCMP é NP-completo, pois podemos fazer a seguinte redução a partir do PSPV (descrito na página 32). Os vértices de $V \setminus R$ mantêm seus pesos originais não negativos. Os vértices em R recebem um peso negativo, grande o suficiente para garantir que todos

eles estejam numa solução ótima. De fato, o SCMP é um problema muito parecido com o PSPV.

4.4.2 Comentários Gerais

Como temos apenas $|V| + K$ restrições em F2 e trabalhamos com um pequeno número de colunas, os PLs mestre resultantes são muito leves e rapidamente resolvidos. De fato, essa foi a maior motivação para trabalharmos com essa formulação. Mas há dois gargalos na resolução da relaxação de F2, que devem ser minimizados:

1. **Número de Iterações** - Depois de muitas experiências feitas com esquemas de geração de colunas em programação linear (inteira ou não), sabe-se que o número de iterações, envolvendo a resolução de subproblemas de apreçamento e um PL, pode ser muito grande até que se obtenha a otimalidade (por exemplo, em Vanderbeck [42]). [4] O problema é agravado pelo fato de que, como as soluções do problema mestre costumam ser degeneradas, é possível passar várias iterações adicionando-se colunas de custo reduzido negativo, mas sem haver melhoria na função objetivo. Esse tipo de *tailing-off* é particularmente comum quando já se está muito próximo, ou até na própria solução ótima. Todos esses problemas podem acontecer no nosso *Branch-and-Price* sobre F2.

2. **O número de apreçamentos exatos** - Não é necessário resolver o apreçamento de forma exata a maior parte do tempo, pois pode-se prosseguir com o algoritmo adicionando-se ao mestre quaisquer colunas com custo reduzido negativo. De fato, no nosso caso, temos heurísticas muito boas para essa função. Mas, mais cedo ou mais tarde, quando as heurísticas não mais encontrarem tais colunas, teremos que resolver os subproblemas de apreçamento de forma exata. Apesar de o SCMP ser NP-completo, é relativamente barato resolver exatamente *alguns poucos* desses subproblemas. O problema é que, justamente quando já se está muito próximo da solução ótima e as heurísticas podem não mais funcionar, costuma ocorrer um *tailing-off*. Devemos criar mecanismos para evitar que o número de apreçamentos exatos cresça demais nesses pontos de *tailing-off*.

4.4.3 Implementação do *branch-and-price* sobre F2

Também começamos as nossas experiências com F2 em um protótipo construído sobre o CPLEX. Posteriormente, foi feita uma reimplementação no ABACUS.

[4] Aliás, a decomposição Dantzig-Wolfe foi introduzida como um artifício para resolver PLs grandes demais para caberem na memória, pois é empiricamente comprovado que, devido a esse alto número de iterações, normalmente é mais rápido resolver o PL original.

Já vimos que para fazer de um *branch-and-cut* um algoritmo eficiente para um problema em particular, podem ser necessárias uma série de aprimoramentos e adaptações do algoritmo básico. Parece que num *branch-and-price* isso é ainda mais verdadeiro. No nosso caso pelo menos, o algoritmo básico é pouco eficiente, mas alguns desses aprimoramentos foram responsáveis por grandes ganhos de desempenho.

Talvez isso se deva ao fato desse tipo de algoritmo ainda não ter sido tão intensamente estudado quanto o *branch-and-cut.* A literatura é menor e menos conclusiva sobre o que costuma funcionar e o que não funciona. O implementador de um *branch-and-price* particular, ainda tem que descobrir muitas coisas por conta própria, por tentativa e erro. Duas das referências mais atualizadas sobre algoritmos de *branch-and-price* em geral são Vanderbeck (1994) [42] e Barnhart et al. (1996) [2].

Vamos descrever alguns pontos do algoritmo que implementamos:

Inicialização

Temos que inicializar o mestre com colunas suficientes para que este seja factível, a fim de obtermos os valores das variáveis duais necessários para começar o apreçamento. Há duas possíveis opções que aparecem na literatura:

- Montar uma base inicial com variáveis artificiais. Normalmente usa-se uma matriz identidade, no nosso caso, $|V|$ colunas com apenas um coeficiente 1, uma para cada restrição em (F2.2). Não é preciso adicionar variáveis artificiais para as restrições em (F2.3), pois as variáveis de folga já servem para esse fim. As variáveis artificiais recebem um custo alto, e são descartadas quando uma solução factível for encontrada.

- Montar as colunas correspondentes à uma boa solução heurística para o problema. A idéia é tentar reduzir o número de iterações, partindo-se de uma solução mais próxima à solução ótima. Alguns autores (e nós também) já observaram que na maioria dos casos isso não costuma funcionar, pois o número de iterações pode até aumentar. O problema é que essa boa solução inteira costuma ser altamente degenerada. Dessa forma, o algoritmo tende a ficar preso nessa solução por bastante tempo. Parece que quando partimos da base artificial, o algoritmo evita naturalmente esses pontos de bloqueio, pelo menos até que se esteja muito próximo da solução ótima.

Descobrimos que, pelo menos no nosso caso, vale a pena inicializar o mestre com uma solução heurística, desde que também se incluam as variáveis artificiais. Só que essas variáveis devem receber o menor custo possível. Seja a_i a variável artificial correspondente à restrição i de (F2.2). Podemos colocar o custo de a_i como $max_k d_{ik}$ sem o risco dessa variável aparecer na solução ótima. Essas variáveis artificiais de baixo custo não apenas

reduzem o bloqueio nessa solução heurística (é verdade que essa solução não é muito boa, veja página 51), como aceleram a convergência do processo como um todo. Vale a pena manter as variáveis artificiais no mestre o tempo inteiro.

Para melhorar ainda mais esse efeito, pode-se até reduzir ainda mais o custo das variáveis artificiais a_i, por exemplo para o j-ésimo maior valor de d_{ik}. Só que agora não é mais garantido que essas variáveis não apareçam na solução ótima. Assim, temos que fazer um gerenciamento desses valores: sempre que tivermos indícios de que um certo a_i vai aparecer no ótimo, por exemplo quando as heurísticas não mais encontrarem colunas de custo reduzido negativo, aumentamos o custo de a_i. Deve-se escolher j de maneira que isso aconteça com pouca freqüência.

Apreçamento Exato

O problema de apreçamento da classe k pode ser formulado da seguinte maneira:

$$
(F2_{ap}) \begin{cases}
\text{Min} \quad \sum_{i \in V} (d_{ik} - \pi_i) x_i - \mu_k & & (1) \\
\left.\begin{array}{l} y_{ij} + y_{ji} \leq x_i \\ y_{ij} + y_{ji} \leq x_j \end{array}\right\} & \forall e = (i,j) \in E & (2) \\
\sum_{i \in V} y_{0i} \leq 1 & & (3) \\
\sum_{(j,i) \in \delta^-(i)} y_{ji} + y_{0i} = x_i & \forall i \in V & (4) \\
\sum_{(j,i) \in \delta^-(S)} y_{ji} + \sum_{i \in S} y_{0i} \geq x_d & \forall S \subset V, 1 < |S| \leq |V| \, ; \forall d \in S & (5) \\
x_i \in \{0,1\} & \forall i \in V & (6)
\end{cases}
$$

Esse problema, que corresponde a um subproblema de F1, pode ser resolvido por um *branch-and-cut* muito similar ao apresentado em 4.3. É interessante manter as restrições ($F2_{ap}$.5) geradas para um subproblema da classe k num *pool*, pois quando um novo problema da classe k for ser resolvido, elas têm grande chance de estarem violadas. Isso ocorre porque se trata do mesmo problema, apenas com custos ligeiramente diferentes.

Entretanto, conforme dito na seção 4.2.3, verificamos experimentalmente que resolver esses subproblemas à integralidade, na prática não ajuda a melhorar os limites duais obtidos. Dessa forma, esse trabalho de forçar a integralidade (a fase de *branching*) é perdido.

Assim, acabamos optando por resolver apenas a relaxação linear da formulação acima. Caso a solução seja fracionária, adicionamos ao mestre a correspondente coluna fracionária. Dessa forma, F2 sempre obterá exatamente os mesmos limites duais de F1, pois não teremos a convexificação parcial, mas uma decomposição Dantzig-Wolfe "pura".

A resolução exata dos subproblemas passa a ser polinomial. Mas como ela ainda é relativamente custosa na prática, deve-se usar esse procedimento de forma parcimoniosa.

Por curiosidade, resolvemos alguns problemas pequenos sem usar heurísticas, gerando todas as colunas por esse procedimento exato fracionário. No início da otimização, é relativamente comum a geração de colunas fracionárias. Porém, à medida que os valores das variáveis duais convergem para o ótimo, essas passam a ser mais raras e não mais ocorrem quando estamos no ótimo. As bases do mestre próximas à solução ótima não contêm colunas fracionárias. Essa experiência suporta a explicação (c) (página 46) para o fato de a convexificação parcial não melhorar F2, pois mostra que em geral, a aproximação ao poliedro inteiro dos subproblemas dada por P^k_{sub} não é tão perfeita assim (explicação (a)). Mas parece que a ligação entre os K subproblemas, feita por (F1.2), leva a uma estrutura de custos duais π ótimos (*shadow prices*), que fazem os subproblemas terem soluções inteiras.

Apreçamento Heurístico

Mesmo se for feita a simplificação de não resolver os subproblemas à integralidade, ainda é muito caro resolver muitos desses problemas exatos. É essencial termos boas heurísticas para resolver o SCMP correspondente a um apreçamento.

HEURÍSTICA DA CONTRAÇÃO DO CAMINHO MÍNIMO

1. **Pré-processamento** - Contraia todos os vértices não-positivos adjacentes, até que todos os vértices negativos estejam cercados por vértices positivos. O peso dos vértices resultantes é a soma dos pesos dos vértices originais contraídos. Se o peso $p(u)$ de algum desses vértices u for menor que μ_k, a coluna correspondente a u já pode ser adicionada ao mestre.

2. Escolha um vértice negativo u. Encontre um caminho de peso mínimo, passando por vértices positivos, entre u e os demais vértices negativos. Isso pode ser feito pelo algoritmo de Dijkstra. Seja $p(uv)$ o peso estritamente positivo de um caminho mínimo entre u e um certo vértice negativo v. Se $p(u) + p(v) + p(uv) < min\{p(u), p(v)\}$, contraímos u, v e os vértices do caminho mínimo, obtendo um novo vértice com peso $p(u) + p(v) + p(uv)$. Se esse peso for menor que μ_k, a coluna correspondente pode ser adicionada ao mestre.

3. Repita o passo 2 até que nenhum par de vértices negativos possa ser contraído.

Essa heurística mostrou-se razoavelmente boa para o nosso problema. Com ela, normalmente é possível obter colunas de custo reduzido negativo, sem resolver o apreçamento exato, até que se esteja próximo a uma solução ótima.

Entretanto, há alguns problemas:

- Em alguns casos a heurística não funciona tão bem e somos obrigados a resolver muitos apreçamentos exatos.
- Como não estamos colocando a cada iteração as colunas com custo reduzido mais negativo o número de iterações pode aumentar (em relação a só usar as colunas ótimas).
- Essa heurística é um pouco cara.

Gostaríamos de ter outras heurísticas para complementar o trabalho desta. Mas não conseguimos obter novas heurísticas significativamente melhores do que essa, sem recorrer a procedimentos computacionalmente pesados.

A solução para esse problema deve-se à seguinte observação: as colunas básicas do mestre tem custo reduzido zero. Será que não é eficiente tentar modificar localmente essas colunas para obter novas colunas de custo reduzido negativo ?

A resposta é sim. Por exemplo, pode-se facilmente verificar se a adição de um vértice i adjacente a uma coluna básica produz uma nova coluna factível de custo reduzido negativo. Da mesma forma, pode-se verificar se a remoção de algum vértice i que não desconecte o componente, produz uma nova coluna de custo reduzido negativo.

A introdução desse tipo de apreçamento heurístico representou uma grande melhoria no *branch-and-price*. Agora é possível se chegar numa solução muito próxima ao ótimo, sem recorrer ao apreçamento exato. O número de iterações também foi reduzido, pois estamos introduzindo mais colunas por iteração. Tudo isso sem onerar significativamente o apreçamento.

Descobrimos posteriormente que Sol e Savelsbergh (1994) [40] já usaram a mesma idéia com sucesso.

Gerenciamento de Colunas

Optamos pelo seguinte esquema:

A cada iteração colocamos no mestre as colunas de custo reduzido negativo encontradas pelos procedimentos de apreçamento acima descritos. Depois de resolver o novo PL, retiramos todas as colunas que ficaram com custo reduzido positivo, com exceção das artificiais, que são sempre mantidas. Não vale a pena guardar as colunas removidas num *pool*.

Resolvendo pelo Dual de F2

O dual da relaxação linear de F2 pode ser escrito como:

$$\begin{aligned} \text{Max} \quad & \sum_{i \in V} \pi_i - \sum_{k=1}^{K} \gamma_k \mu_k \\ & \sum_{i \in l} \pi_i - \mu_k \leq c_l \qquad \forall l \in L \\ & \pi_i \geq 0 \qquad\qquad\qquad \forall i \in V \end{aligned}$$

Esse PL pode ser resolvido de modo praticamente idêntico ao que usamos para resolver o primal. A única diferença é que as soluções geradas pelo apreçamento serão traduzidas em linhas e não colunas. Isso pode ser vantajoso ou não, dependendo do esquema de gerenciamento de cortes (colunas ou linhas) utilizado. Como o tempo de resolução de um PL depende mais do número de linhas que do número de colunas, teríamos uma vantagem quando trabalhássemos com poucos cortes no mestre.

Mas, no nosso caso há uma pequena vantagem que independe desse gerenciamento: as colunas correspondentes às variáveis artificiais, e que sempre vão estar no mestre, podem ser representadas como limites superiores das variáveis π. Ou seja, essas variáveis não vão pesar no PL. O dual também é menos suscetível a instabilidades numéricas do que o primal.

Algumas experiências comprovaram que, com o gerenciamento de cortes adotado, é realmente melhor trabalhar com o dual. Mas a melhoria obtida é pequena, mesmo porque a resolução de PLs não é o gargalo do nosso algoritmo.

Não foi possível embutir a resolução pelo dual na implementação sobre o ABACUS, pois a versão corrente desse sistema não previu a possibilidade de se resolver um problema de minimização, através de PLs de maximização.

Regra de Branching

Esse é um dos pontos mais críticos de um *branch-and-price*.

Fazer o *branching* sobre as variáveis λ de F2, teria graves inconvenientes. Em primeiro lugar, os subproblemas gerados seriam altamente desbalanceados. Fixar $\lambda_l = 1$ realmente reduz muito o espaço de soluções desse subproblema. Mas, dado o enorme número de variáveis , fixar $\lambda_l = 0$, restringe muito pouco o espaço de soluções do outro subproblema.

Outro problema grave é que essa regra destruiria a estrutura dos subproblemas de apreçamento. Ao fixar em 0 um componente l da classe k, não apenas retiramos l do mestre, mas temos que garantir que o subproblema da classe k não volte a gerar a coluna l. Mas agora, l provavelmente será uma coluna com custo reduzido negativo e terá forte tendência a voltar a aparecer, tanto no apreçamento exato quanto heurístico. Assim, temos que alterar a estrutura dos subproblemas, adicionando restrições para evitar que l apareça. À medida que vários $\lambda_l = 0$ forem sendo fixados em 0, o apreçamento de colunas vai ficando mais complicado.

Um dos esquemas mais utilizados em algoritmos de *branch-and-price* [42, 2] é usar a regra que Ryan e Foster (1981) [35] propuseram para o problema do *set-partitioning* (no problema destes autores não havia geração de colunas). No nosso caso, essa regra implicaria em gerar dois subproblemas; no primeiro deles, um certo vértice i deve obrigatoriamente estar no mesmo componente (e portanto na mesma classe) de um certo vértice j, no segundo i e j devem estar em componentes diferentes. Naturalmente i e j devem ser escolhidos de modo a cortar a solução fracionária λ corrente.

Essa regra sem dúvida é mais balanceada. Mas ela complica os problemas de apreçamento. Não é difícil forçar que dois vértices estejam no mesmo componente, seja no apreçamento exato ou no heurístico. Basta contrair esses vértices a um único vértice e resolver o SCMP resultante normalmente. Isso até ajuda na redução do tamanho do problema. Mas, por outro lado, forçar i e j a estarem em componentes diferentes leva a um SCMP com uma restrição adicional, e portanto a um apreçamento mais complicado.

Optamos por fazer o *branching* em cima das variáveis x de F1. Primeiro convertemos a solução fracionária λ numa solução equivalente (x, y). Para fixar um x_{ik} fracionário em 0, removemos do mestre todas as colunas da classe k que contem i. Para evitar que essas colunas voltem, removemos i do SCMP da classe k. O problema de apreçamento resultante pode ser resolvido pelos mesmos procedimentos anteriores, sendo até um pouco menor. Para fixar um x_{ik} em 1, procedemos indiretamente, fixando todos os $x_{ik'}$, $k' \neq k$ em 0.

Uma vantagem adicional dessa regra é que mantemos uma estreita relação com o *branch-and-cut* sobre F1. Por exemplo, a escolha da variável de *branching* x_{ik} pode usar o mesmo procedimento usado em F1. O mesmo vale para a fixação por implicação lógica, pois se x_{ik} está fixo em 1, e $\gamma_k = 1$, podemos restringir o subproblema da classe k a gerar colunas que contenham i. Essa fixação de um vértice simplifica a resolução do SCMP. [5]

A fixação por custo reduzido sobre as variáveis λ não deve ser usada, a fim de não destruir a estrutura do SCMP, pelos mesmos motivos que desaconselham o *branching* sobre λ. Mas, através dos x_{ik}, essa fixação pode ser implementada. Por exemplo, suponha que $x_{ik} = 0$. Resolvendo o apreçamento exato da classe k com a raiz i fixa, calculamos o mínimo custo reduzido de uma coluna da classe k que contenha i. Se esse custo, somado ao valor da relaxação linear de F2 corrente for maior do que a melhor solução inteira conhecida, x_{ik} pode ser fixada em 0.

Branching Precoce

Via de regra, usando apenas colunas geradas heuristicamente, somos capazes de chegar de modo relativamente rápido a uma solução próxima à solução ótima do subproblema de

[5] O SCMP com uma raiz fixa é equivalente à versão do problema de Steiner com peso nos vértices discutida no capítulo 6 de Magnanti e Wolsey (1994) [26]

um nó do *branch-and-bound*. Chamaremos essa solução de **solução heurística**. Entretanto, a partir da solução heurística, atingir a solução ótima e principalmente, provar a otimalidade, pode ser custoso. Isso ocorre, tanto pelo *tailing-off*, quanto pela necessidade de usar o apreçamento exato algumas vezes.

Assim, sempre que após a obtenção da solução heurística, estivermos perdendo muito tempo para achar a solução ótima de um nó, optamos por fazer um *branching* precoce. Verificamos experimentalmente, que caso a solução ótima de um nó seja inteira, na maioria das vezes a solução heurística coincidirá com a ótima. Isso é equivalente a dizer que se a solução heurística for fracionária, na maioria das vezes a solução ótima também o será. Ou seja, provavelmente teríamos que fazer esse *branching* de qualquer maneira.

Mas há uma outra razão para se querer achar a solução ótima do subproblema de um nó: a obtenção de um limite dual válido para esse nó. Esse limite pode ser útil p/ cortar esse nó, caso ele seja pior que o melhor limite primal conhecido para o problema geral. Mas podemos obter um limite dual válido antes da otimalidade.

Seja Z_{cor} o valor da solução corrente do mestre e sejam cr_k os valores correntes das solução exatas dos K subproblemas de apreçamento. O seguinte limite dual é válido para um subproblema de um nó (ver por exemplo, Vanderbeck [42]):

$$LD = Z_{cor} + \sum_{k=1}^{K} \gamma_k cr_k$$

Usando-se esse limite, é possível em alguns casos cortar um ramo da árvore de *branch-and-bound* sem necessariamente resolvê-lo otimamente.

4.4.4 Resultados Computacionais

Vamos aos novos resultados computacionais, que podem ser comparados com os da página 53, pois as instâncias são as mesmas.

Tabela 4.5: Instâncias randômicas, num grafo H de grade quadrado

$\|V\|$	K	$\gamma_k = 1$				$\gamma_k = 2$				$\gamma_k = 3$			
		Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs
25	3	15	10	3	69	3	2	1	25	2	1	1	20
36	3	192	88	9	205	13	6	1	46	7	4	1	23
49	3	-	565	17	334	135	66	7	99	51	28	5	74
64	3	-	-	-	-	419	147	7	181	112	56	7	92

Tabela 4.6: Instâncias randômicas, num grafo H obtido por *random_planar_graph*

$\|V\|$	K	$\gamma_k = 1$				$\gamma_k = 2$				$\gamma_k = 3$			
		Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs
25	3	19	8	1	51	3	2	1	22	2	1	1	14
36	3	113	68	9	148	18	6	1	35	6	4	1	27
49	3	-	477	13	290	209	48	5	79	43	22	3	60
64	3	-	-	-	-	381	275	11	204	156	73	7	81

Tabela 4.7: Instâncias com estrutura, num grafo H de grade quadrado

$\|V\|$	K	$\gamma_k = 1$				$\gamma_k = 2$				$\gamma_k = 3$			
		Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs	Tm	Tmd	Nós	PLs
100	5	-	220	3	83	310	109	1	59	139	56	1	43

Comparando-se com os resultados do *branch-and-cut* sobre F1, vê-se que o primeiro algoritmo foi mais eficiente na maioria dos casos.

Alguns comentários adicionais:

- Nas instâncias em que o limite dual obtido pela relaxação inicial foi igual ou quase igual ao ótimo, o algoritmo se comportou bem e às vezes até foi melhor do que o *branch-and-cut*. Mas quando isso não acontece, o desempenho é prejudicado. O problema é que para cada folha da árvore de *branch-and-bound* temos que resolver no mínimo K apreçamentos exatos, o que pode ser muito custoso.

- O desempenho do algoritmo é mais instável do que o *branch-and-cut* sobre F1. Algumas instâncias são rapidamente resolvidas, enquanto outras demoram muito. Por sinal, uma leve mudança em algum ponto do algoritmo, é capaz de alterar bastante o tempo de cpu da mesma instância. Isso se deve ao fato de que dependendo da seqüência de bases do mestre, pode-se ou não evitar as bases mais degeneradas, que são pontos de *tailing-off*.

Observamos, que na resolução da relaxação inicial, mais ou menos metade do tempo é gasta nas várias iterações heurísticas e metade nas raras iterações com apreçamento exato. Já após o *branching*, a convergência das iterações heurísticas é rápida, mas o tempo para se provar a otimalidade do nó, que depende de apreçamento exato, pouco muda. O resultado, é que quando há muitos *branchings*, o tempo do algoritmo é dominado pelo apreçamento exato.

4.5 Reforçando as Formulações

Os testes computacionais feitos com os algoritmos de *branch-and-bound* sobre F1 e F2 mostram que os limites duais obtidos por essas formulações são razoavelmente bons. Mas o *gap* entre esse limite e o valor da solução ótima, às vezes ainda é grande, principalmente nas instâncias difíceis de maior tamanho. Queremos encontrar novas classes de desigualdades válidas para o PAgC, a fim de reforçar as formulações F1 e F2 e reduzir esse *gap*.

4.5.1 Novas Classes de Desigualdades Fortes

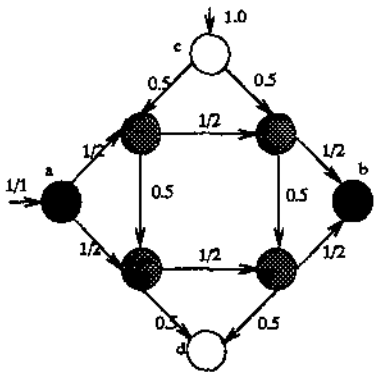


Figura 4.8: Solução fracionária de F1 ($K = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

Na figura 4.8 mostramos uma solução fracionária de F1. Os vértices a e b estão atribuídos à classe 1 e c e d à classe 2. Mas observando-se o grafo, nota-se que qualquer caminho de a até b separa c de d e vice-versa. Ou seja, não existem dois caminhos disjuntos nos vértices entre os pares de vértices (a, b) e (c, d). Assim, como $\gamma_1 = \gamma_2 = 1$, a seguinte desigualdade é válida para essa instância:

$$x_{a1} + x_{b1} + x_{c2} + x_{d2} \leq 3 \qquad (24)$$

Adicionando-se essa desigualdade à F1, teremos, nessa instância, soluções inteiras para qualquer valor da função objetivo.

Definimos P_{γ_1} como o poliedro inteiro do PAgC **para o caso em que todos os γ_k são iguais a 1**, ou seja, o fecho convexo dos vetores binários de atribuições x em cada classe induz no máximo um componente conexo em H. Temos então a seguinte classe de desigualdades válidas para P_{γ_1}:

Definição 4.1 *Sejam (s_1, t_1) e (s_2, t_2) dois pares de vértices distintos pertencentes à H, com a propriedade que não existem dois caminhos disjuntos nos vértices unindo s_1 a t_1 e s_2 a t_2. Sejam k_1 e k_2 duas classes distintas, k_1 e $k_2 \in \{1, \ldots, K\}$. Então a seguinte desigualdade será chamada de* **Desigualdade da Cruz***:*

$$x_{s_1 k_1} + x_{t_1 k_1} + x_{s_2 k_2} + x_{t_2 k_2} \leq 3$$

A desigualdade da cruz é claramente válida para P_{γ_1}. O apêndice A contém um estudo teórico dessa classe de desigualdades, em que se mostra que a desigualdade da cruz define uma faceta do poliedro P_{γ_1} em muitos casos importantes.

Introduziremos agora uma seqüência de classes de desigualdades válidas que generalizam a desigualdade da cruz.

Definição 4.2 *Sejam $(s_1, t_1), (s_2, t_2), \ldots, (s_q, t_q)$, q pares de vértices distintos em H, com a propriedade que não existem q caminhos disjuntos nos vértices unindo s_1 a t_1, s_2 a $t_2, \ldots,$ e s_q a t_q. Seja Q o conjunto de inteiros $\{1, \ldots, q\}$. Cada $q \in Q$ identifica uma classe distinta k_q. Então a seguinte desigualdade será chamada de* **Desigualdade da q-Cruz**:

$$\sum_{q \in Q} (x_{s_q k_q} + x_{t_q k_q}) \leq 2.|Q| - 1$$

A desigualdade da q-cruz é claramente válida para P_{γ_1} e generaliza a desigualdade da cruz, pois essa é o o caso particular da 2-cruz.

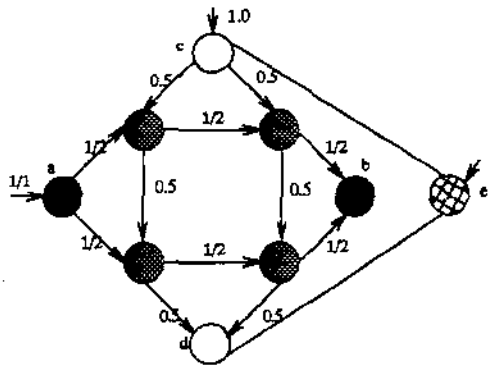


Figura 4.9: Solução fracionária de F1 ($K = 3$, $\gamma_1 = 1$, $\gamma_2 = 1$, $\gamma_3 = 1$).

A figura 4.9 mostra uma solução de F1 que não pode ser cortada por nenhuma desigualdade da q-cruz. Aqui, $K = 3$ e o vértice rotulado de e (hachurado) está atribuído à classe 3. A seguinte desigualdade válida corta essa solução:

$$x_{a1} + x_{b1} + x_{c2} + x_{d2} + x_{e3} \leq 4 \qquad (25)$$

A validade de (25) segue do fato de não existem dois caminhos unindo a a b e c a d em $H - e$.

Definição 4.3 *Seja R um subconjunto dos vértices de H. Sejam $(s_1, t_1), (s_2, t_2), \ldots, (s_q, t_q)$, q pares de vértices distintos em $H - R$, com a propriedade que não existem q caminhos*

disjuntos nos vértices em $H - R$ unindo s_1 a t_1, s_2 a t_2, ..., e s_q a t_q. Seja Q o conjunto de inteiros $\{1, \ldots, q\}$. Seja $\overline{Q}$ o conjunto de inteiros $\{q+1, \ldots, K\}$. Cada $q \in Q \cup \overline{Q}$ identifica uma classe distinta k_q. Então a seguinte desigualdade será chamada de **Desigualdade da q-Cruz em $H - R$:**

$$\sum_{q \in Q} (x_{s_q k_q} + x_{t_q k_q}) + \sum_{i \in R} \sum_{q \in \overline{Q}} x_{i k_q} \leq 2.|Q| + |R| - 1$$

Essa desigualdade é válida para P_{γ_1} e generaliza a q-cruz, pois esta é o caso particular em que R é vazio.

A classe de desigualdades da q-Cruz em $H - R$ tem a propriedade interessante de ser definida apenas nas variáveis de atribuição x. Isso implica que ela também pode ser adicionada de modo direto à formulação F2, sem complicar os procedimentos de apreçamento.

Por exemplo, para adicionar a desigualdade da cruz $x_{a1} + x_{b1} + x_{c2} + x_{d2} \leq 3$ na formulação F2 da instância da figura 4.8, introduzimos uma nova restrição. As colunas da classe 1 (resp. 2) que contiverem os vértices a e b (resp. c e d) terão coeficiente 2 nessa restrição. As colunas da classe 1 (2) que contiverem apenas a ou b (c ou d) terão coeficiente 1. As colunas da classe 1 (2) que não contiverem os vértices a e b (c e d) terão coeficiente 0. O lado direito da restrição continua sendo 3. Pode-se verificar que essa restrição corta a solução fracionária mostrada na figura 4.4.

Após a adição dessa nova restrição, teremos uma nova variável dual ν. O subproblema de apreçamento da classe 1 (2) incluirá novos termos $\nu.x_a + \nu.x_b$ ($\nu.x_c + \nu.x_d$). Mas isso apenas muda a função objetivo dos subproblemas. Como os procedimentos de apreçamento que apresentamos se aplicam a uma função objetivo qualquer, não haverá nenhuma complicação adicional.

Agora vamos generalizar essas classes de desigualdades para o PAgC geral, para quaisquer valores de γ_k. Seja P o poliedro formado pelo fecho convexo dos vetores (x, y), em que x é inteiro e γ-conexo, e (x, y) respeita as restrições de F1.

Definição 4.4 *Seja S um subconjunto dos vértices de H. Sejam $(s_1, t_1), (s_2, t_2), \ldots, (s_q, t_q)$, q pares de vértices distintos pertencentes à S, com a propriedade que não existem q caminhos disjuntos nos vértices em S unindo s_1 a t_1, s_2 a t_2, ..., e s_q a t_q. Seja Q o conjunto de inteiros $\{1, \ldots, q\}$. Cada $q \in Q$ identifica uma classe distinta k_q. Então a seguinte desigualdade será chamada de* **Desigualdade da q-Cruz Generalizada:**

$$\sum_{q \in Q} (x_{s_q k_q} + x_{t_q k_q}) \leq \sum_{q \in Q} \Big(\sum_{(j,i) \in \delta^-(S)} y_{ji}^{k_q} + \sum_{i \in S} y_{0_{k_q} i} \Big) + |Q| - 1$$

Essa desigualdade é válida para P e generaliza a desigualdade da q-Cruz em $H - R$, no seguinte sentido: suponha que todos os γ_k são iguais a 1 e que existe um vetor (x, y)

respeitando F1, em que x viole uma desigualdade da q-Cruz em $H - R$. O vetor (x, y) certamente também viola uma desigualdade da q-Cruz generalizada, com $S = H - R$.

Mas mesmo quando os γ_k são iguais a 1, essa nova classe de desigualdades pode ser mais forte. Na figura 4.10 vemos uma solução fracionária de F1 que respeita todas as desigualdades da q-Cruz em $H - R$, para qualquer R, mas viola uma 2-Cruz generalizada, com $S = \{a, b, c, d\}$. Adicionando-se essa desigualdade teremos uma solução inteira para qualquer função objetivo.

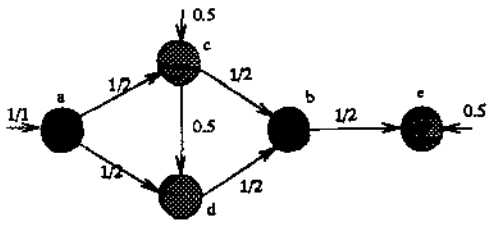


Figura 4.10: Solução fracionária de F1 ($K = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

4.5.2 Novos Subproblemas de Separação

O problema de separação da desigualdade da q-cruz (e portanto de suas generalizações), para um q qualquer, é NP-completo, pois imediatamente pode ser feita uma redução a partir do seguinte problema:

Problema dos Caminhos Conectantes Disjuntos
Instância: Grafo $H = (V, E)$; coleção de q pares de vértices disjuntos $(s_1, t_1), \ldots,$ $(s_q, t_q))$.
Questão: Existem q caminhos disjuntos nos vértices, ligando cada par (s_q, t_q), $1 \leq i \leq q$?

Esse problema é NP-completo, mesmo para grafos planares (Garey e Johnson [12], problema ND40).

Mas Shiloach (1980) [37] resolveu o caso $q = 2$, para um grafo H qualquer, em tempo polinomial. Esse resultado foi considerado surpreendente, pois achar dois pares de caminhos disjuntos nos vértices, em grafos direcionados, é NP-completo (Fortune, Hopcroft e Wylie (1978), citado em [37]).

O algoritmo de Shiloach tem complexidade $O(|V|.|E|)$ e nos permite separar a desigualdade da 2-cruz em tempo polinomial. Não implementamos a separação da 2-cruz para o caso geral, pois o algoritmo de Shiloach que aparece no artigo é extremamente complicado. O problema é dividido em dezenas de casos, que são tratados isoladamente. Vê-se que esse autor tinha como objetivo principal provar a polinomialidade do problema.

Mas o problema pode ser bem mais facilmente resolvido em grafos planares.

Proposição 4.4 *Seja F uma face de um grafo planar H. Suponha que s_1, s_2, t_1 e t_2 aparecerem em F nessa ordem cíclica. Essa condição é suficiente para afirmar que não existem dois caminhos disjuntos nos vértices ligando esses dois pares de vértices.*

É imediato verificar que isso é verdadeiro. Se houverem três caminhos disjuntos nos vértices ligando os dois pares, a condição também é necessária (Perl e Shiloach (1978) [32]). Os demais casos planares também são tratados em [32].

O nosso algoritmo de separação da desigualdade da cruz para grafos planares, usa a proposição acima. Para cada face F de H geramos uma "string" de tamanho máximo $|F|.K$ contendo os valores de cada $x_{ik} > 0$, na ordem em que os i aparecem na face. Temos agora o problema da identificação eficiente de um padrão cíclico nessa string. Resolvemos esse problema em $O(|F|.K^2)$.

Recentemente, Robertson e Seymour [34] obtiveram o seguinte resultado: existe um algoritmo polinomial de complexidade $O(|V|.|E|)$ que resolve o problema dos q caminhos disjuntos, num grafo qualquer, para qualquer q fixo. [6] Esse resultado tem uma grande importância teórica, mas não fornece um algoritmo prático para os casos em que $q \geq 3$. Mas existem boas heurísticas para se resolver esse problema quando $q \geq 3$, principalmente num grafo planar (Schrijver (1989) [36]). Essas heurísticas permitem determinar, em grande parte das instâncias, se os q caminhos disjuntos existem ou não (nas instâncias restantes, elas não são capazes de fornecer uma resposta). Essas heurísticas poderiam ser aproveitadas para a separação de desigualdades das classes que generalizam a desigualdade da cruz.

4.5.3 Resultados Computacionais

Incorporamos ao *branch-and-cut* sobre F1, o algoritmo de separação da desigualdade da cruz, para o caso em que H é planar, mencionado acima. Essa desigualdade só é válida para o caso em que os γ_k são todos iguais à 1, mas como vimos, esse é justamente o caso mais difícil. Todas as instâncias com que já vinhamos trabalhando usam grafos H planares.

Os resultados obtidos por esse *branch-and-cut* melhorado foram muito bons. É notável que, mesmo esse subcaso tão particular da desigualdade da q-cruz generalizada, é capaz de reforçar significativamente F1. Os limites duais obtidos foram bem melhorados, o que causou uma redução do número de nós de *branch-and-bound*, principalmente nas instâncias mais difíceis.

Mas descobrimos que essas desigualdades também servem para reduzir o *tailing-off* relacionado à separação de (F1.6). Como a separação da cruz é bastante eficiente, podemos executá-la em todas as iterações, juntamente com a separação de (F1.6), desde o início do

[6] A notação O esconde uma constante que cresce exponencialmente em função de q.

algoritmo. Achar uma desigualdade da cruz violada é uma garantia de que não vai haver *tailing-off* nessa iteração, pois ela obriga a uma mudança nas variáveis x.

Vamos aos novos resultados computacionais. Esses resultados podem ser comparados com os da página 53, pois as instâncias são as mesmas.

Tabela 4.8: Instâncias randômicas, num grafo H de grade quadrado

$\|V\|$	K	$\gamma_k = 1$			
		Tm	Tmd	Nós	PLs
25	3	5	4	3	11
36	3	29	26	7	32
49	3	231	164	15	79
64	3	-	1341	77	761

Tabela 4.9: Instâncias randômicas, num grafo H obtido por *random_planar_graph*

$\|V\|$	K	$\gamma_k = 1$			
		Tm	Tmd	Nós	PLs
25	3	6	4	1	14
36	3	41	31	9	29
49	3	305	73	3	34
64	3	-	632	23	311

Tabela 4.10: Instâncias com estrutura, num grafo H de grade quadrado

$\|V\|$	K	$\gamma_k = 1$			
		Tm	Tmd	Nós	PLs
100	5	241	113	1	36

Capítulo 5

Conclusões e Comentários Finais

Nesse capítulo faremos alguns comentários e conjecturas a partir dos resultados obtidos.

- A extensão direta do nosso trabalho seria aplicar os nossos métodos e algoritmos a problemas reais de classificação com conexidade. Só dessa forma, seria possível avaliar se os métodos que propomos realmente funcionam bem em aplicações práticas.

  Não fazia parte da nossa proposta de trabalho fazer esse tipo de avaliação prática. Isso poderia ser muito complicado. Um estudo sério desse tipo exige conhecimentos sólidos de estatística, bem como conhecimentos particulares às aplicações em questão.

- **Comparação** *branch-and-cut* × *branch-and-price*

  Os testes computacionais mostraram que mesmo com todos os melhoramentos feitos no algoritmo de *branch-and-price*, este ainda é menos eficiente do que o *branch-and-cut*.

  Achamos que o ponto chave que explica a desvantagem de F2 em relação a F1, está no fato de F2 não ser significativamente mais forte do que F1. Dessa forma, F2 praticamente se reduz a uma decomposição Dantzig-Wolfe de F1, uma técnica que sabidamente pode não ser eficiente. Além disso, um dos principais problemas da formulação F1, o *tailing-off*, é transportado para o apreçamento exato dos subproblemas. Dessa forma, não havia realmente muito espaço para obtermos um algoritmo mais eficiente.

  Mas, esses mesmos testes, mostram que a diferença de eficiência entre os dois algoritmos não é tão grande assim. Assim, nossas experiências de certa forma suportam a idéia de que algoritmos de *branch-and-price* podem ser uma boa opção quando:

  a) Resolver alguns poucos apreçamentos exatos é relativamente barato.

  b) O apreçamento pode ser resolvido em grande parte do tempo por boas heurísticas.

c) A formulação com um grande número de colunas é significativamente mais forte do que outros tipos de formulação disponíveis.

A formulação F2 não cumpre o item (c).

Mas, por esse raciocínio, o *branch-and-price* poderia ser uma excelente opção, por exemplo, para o problema mais complicado da extensão direta do K-medóide, descrito na seção 3.1. Nesse caso, as formulações disponíveis sobre as variáveis x_{ik} são fracas, grandes e simétricas. Mas existe uma formulação bem mais forte para esse problema, parecida com F2.

Aqui, uma coluna representaria um subgrafo separado em até γ_k componentes. O custo da coluna l seria dado por $\sum_{i \in l} d_{i\,r(l)}$, onde $r(l)$ é o representante da coluna, ou seja o vértice que minimiza essa soma. O subproblema de apreçamento exato seria mais complicado que o de F2, mas ainda relativamente tratável. Boas heurísticas teriam que ser construídas para esse subproblema.

- As heurísticas de apreçamento baseadas na busca local a partir das colunas básicas do mestre foram as principais responsáveis pelo melhoramento do *branch-and-price* sobre F2 em relação ao algoritmo básico. Savelsbergh e Sol [40] já tinham verificado que esse tipo de apreçamento pode ser muito útil, em outra aplicação do *branch-and-price.*

  O segundo maior responsável pela melhoria, foi o uso das variáveis artificiais de baixo custo. Também foi muito importante o uso de heurísticas de arredondamento, baseadas no valor corrente da solução do mestre (página 51).

  Acreditamos que é possível combinar essas idéias no seguinte contexto: quando temos um problema que tem uma formulação muito forte com um grande número de colunas, mas que o apreçamento exato dessas colunas seja muito caro, realmente intratável em grandes instâncias.

  É possível usar uma espécie de *branch-and-price* sem a resolução exata dos apreçamentos como uma boa heurística para esse problema. usando pesadamente heurísticas de apreçamento baseadas em busca local e manipulações nos custos das variáveis artificiais (aumentando progressivamente o valor dessas até que todas elas saiam das bases do mestre).

  Essa heurística poderia ser interpretada como um tipo de algoritmo de busca local moderna, como um *simulated annealing* ou um algoritmo genético, mas que ao invés de se guiar pelos custos das variáveis, seria guiado pelas informações mais globais dadas pelos custos reduzidos.

  É claro que tudo isso ainda não passa de especulação, mas mesmo no nosso caso, em que a formulação F2 não é tão forte assim, já verificamos que ao resolver o *branch-*

and-price sem os apreçamentos exatos obtemos boas soluções inteiras logo nos primeiros nós da árvore de *branch-and-bound*. Essas soluções inteiras, nem sempre são as soluções dos nós, mas a soluções obtidas pela heurística de arredondamento.

- A adição da desigualdade da cruz melhorou muito o nosso *branch-and-cut* sobre F1. Isso mais uma vez mostra a importância de se fortalecer uma formulação, obtendo novas classes de desigualdades válidas bem ajustadas ao problema particular tratado.

  Vale a pena comentar que em todas as soluções fracionárias obtidas por essa nova formulação, que tivemos a oportunidade de analisar manualmente, era evidente a presença de desigualdades da q-cruz generalizada violadas. É possível que a implementação de uma boa heurística para esse caso geral, leve a uma grande melhoria dos algoritmos que temos hoje.

Apêndice A

Um Estudo Poliédrico da Desigualdade da Cruz

No capítulo 4, verificamos experimentalmente que a desigualdade da cruz reforça significativamente F1 ou F2. Nesse apêndice, pretendemos estudar a qualidade dessas desigualdades de uma forma teórica.

Em algumas instâncias pequenas, pudemos comprovar manualmente que a desigualdade da cruz era uma faceta de P_{γ_1}. Gostaríamos de saber em que condições isso acontece. Uma das maiores dificuldades para se responder esse tipo de questão, está no fato de o poliedro P_{γ_1} não ter dimensão cheia.

Mas suponha que $K = 2$. Nesse caso, podemos substituir as $2.|V|$ variáveis x_{ik} por apenas $|V|$ variáveis x'_i, por exemplo, usando a convenção que $x'_i = 1$ se e somente se $x_{i1} = 1$, e $x'_i = 0$ se e somente se $x_{i2} = 1$. Definimos P_{sim} como o poliedro simplificado do PAgC para o caso em que todos os γ_k são iguais a 1 e $K = 2$, ou seja, o fecho convexo dos vetores binários x' em que os vértices em 1 e os vértices em 0 induzem no máximo um componente conexo em H.

Proposição A.1 *Se H for conexo e não tiver separadores, P_{sim} tem dimensão cheia.*

Prova: O vetor **0** pertence à P_{sim}. Um vértice i é um separador de H se e somente se $H - i$ não for conexo. Para cada vértice $i \in H$, o vetor de incidência $\chi^{\{i\}}$ (o vetor com $x'_i = 1$ e todos os demais componentes em 0) está em P_{sim}, desde que i não seja um separador de H. Temos então $|V| + 1$ vetores afim independentes em P_{sim}. □

Com a substituição de variáveis feita para definir P_{sim}, a desigualdade da cruz se torna:

$$x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$$

Chamaremos os vértices do conjunto $E = \{s_1, t_1, s_2, t_2\}$, ou seja, os vértices com coeficiente não-nulo numa certa desigualdade da cruz, de vértices especiais. Queremos descobrir em que condições essa desigualdade da cruz é uma faceta de P_{sim}.

Se H não for um grafo conexo, é trivial descrever P_{sim}. Se H estiver dividido em mais de 3 componentes conexos, P_{sim} é o poliedro vazio. Se H estiver dividido em dois componentes conexos, P_{sim} terá exatamente quatro soluções inteiras. A desigualdade da cruz não é uma faceta de P_{sim} em nenhum desses casos.

Diremos que um subconjunto de vértices $C \subseteq V$ de $G = (V, E)$ é **c-conexo** em G (complementarmente conexo em G) se e somente se o subgrafo induzido por C for conexo e o subgrafo complementar induzido por $V \setminus C$ também. Por facilidade de notação, às vezes diremos que um subgrafo de H é c-conexo, quando o conjunto de seus vértices for c-conexo. As soluções inteiras em P_{sim} são justamente os vetores de incidência de um subgrafo c-conexo.

Lema A.1 *Seja H um grafo conexo e sem separadores. Seja C um conjunto c-conexo em H, tal que $0 < |C| \leq |V|$. Seja x um vértice em C. Então é possível construir uma seqüência encaixada de subconjuntos de vértices de H, $C = C_1 \supset C_2 \supset \ldots \supset C_{|C|} = \{x\}$ tal que para cada i, $1 \leq i \leq |C|$, C_i é c-conexo em H.*

Prova: $C_{|C|} = \{x\}$ é c-conexo, pois x não é um separador de H. Suponha que C_i, $1 \leq i < |C|$, é c-conexo em H. Vamos formar um C_{i+1} c-conexo retirando um vértice de C_i diferente de x e adjacente a $H \setminus C_i$. Os vértices com essas características serão chamados de vértices candidatos. Como H não tem separadores, há pelo menos um vértice candidato. Vamos mostrar que pelo menos um dos vértices candidatos serve obter um C_{i+1} c-conexo em H.

Seja v_0 um desses candidatos. Se $C_i - v_0$ for conexo, v_0 será o escolhido. Caso contrário, $C_i - v_0$ está separado em j partes conexas $A_1^1, \ldots, A_j^1$, $j \geq 2$. Como C_i é conexo, existe pelo menos uma aresta entre cada A_j^1 e v_0. Como v_0 não é um separador de H, existe pelo menos uma aresta entre cada A_j^1 e $H \setminus C_i$. Ajuste a notação para que A_1^1 seja um componente que não contêm x.

Seja v_1 um vértice em A_1^1 adjacente a $H \setminus C_i$. Suponha que $A_1^1 - v_1$ está separado em $j \geq 0$ componentes, $A_1^2, \ldots, A_j^2$. Como v_1 não é separador de H, todos os componentes A_j^2 estão conectados, ou a v_0, ou a $H \setminus C_i$. Se todos eles estão conectados à v_0 (ou $j = 0$), podemos escolher v_1, pois $C_i - v_1$ será conexo. Caso contrário, existe algum componente, digamos A_1^2, não ligado a v_0, mas ligado a $H \setminus C_i$.

Seja v_2 um vértice de A_1^2 adjacente a $H \setminus C_i$, ...

Procedendo recursivamente dessa forma, obtemos uma seqüência v_l de vértices candidatos contidos em conjuntos cada vez menores A_1^l. Se os componentes de $A_1^l - v_l$ estão todos ligados a $C_i \setminus A_1^l$, v_l será o escolhido, e $C_{i+1} = C_i - v_l$ será c-conexo em H. Na pior das hipóteses, teremos que avançar até que $A_1^l = \{v_l\}$. □

Proposição A.2 *Suponha que H é conexo e não tem vértices separadores. Então a desigualdade da cruz $x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$ é uma faceta de P_{sim}.*

Prova: Seja $a^t x' \leq \alpha$ uma desigualdade que define uma faceta de P_{sim} nas condições acima e suponha que

$$F = \{x' \in R^{|V|} | \, x' \in P_{sim} \text{ e } x'(s_1+t_1-s_2-t_2) = 1\} \subseteq F_a = \{x' \in R^{|V|} | \, x' \in P_{sim} \text{ e } a^t x' = \alpha\}$$

Iremos mostrar que $a^t x \leq \alpha$ é um múltiplo escalar da desigualdade da cruz indicada, e portanto, essa é uma faceta de P_{sim}.

Como H não tem separadores, $H - s_1$ e $H - t_1$ são conexos. Assim, os vetores de incidência $\chi^{\{s_1\}}$ e $\chi^{\{t_1\}}$ pertencem a F e portanto a F_a. Logo, $a_{s_1} = a_{t_1} = \alpha$.

O ponto crítico da prova e que depende da análise de vários casos, é mostrar que $a_v = 0 \; ; \forall v \in V \setminus E$. Por enquanto suponha que isso é verdade.

Como H não tem separadores, $H - s_2$ e $H - t_2$ são conexos. Assim, os vetores de incidência χ^{H-s_2} e χ^{H-t_2} pertencem à F e portanto à F_a. Como $a_v = 0$, $\forall v \in V \setminus E$, então $a_{s_2} = a_{t_2} = -\alpha$. Isso mostra que $a^t x \leq \alpha$ realmente é um múltiplo escalar da desigualdade da cruz indicada.

Subproposição A.2.1 $a_v = 0 \; ; \forall v \in V \setminus E$.

Prova: Suponha que, para um certo $v \in V \setminus E$, exista um componente c-conexo C, que inclua v, inclua s_1 ou t_1 (mas não ambos) e não inclua s_2 e t_2. O vetor de incidência χ^C pertence à F e portanto à F_a. Mas, pelo lema anterior, é possível construir uma seqüência encaixada de $|C|$ vetores de incidência em F_a, com $C_{|C|} = \{s_1\}$ ou $C_{|C|} = \{t_1\}$. Isso mostra que $a_u = 0$ para todos os $|C|-1$ vértices não-especiais u em C, e em particular, que $a_v = 0$. Iremos explorar essa técnica para mostrar que a subproposição é verdadeira.

Seja C_{s1} um subgrafo c-conexo, contendo s_1, mas não contendo nenhum dos outros 3 vértices especiais, maximal. Como $\{s_1\}$ é c-conexo, temos certeza que esse subgrafo existe. Vamos mostrar que quase sempre existe um subgrafo C_{t1} c-conexo, contendo t_1 e todos os vértices em $H \setminus C_{s1}$, mas sem conter s_2 e t_2. Nesses casos, a subproposição segue, pois C_{s1} e C_{t1} cobrem todos os vértices não especiais. No único caso em que não é possível construir C_{t1}, mostraremos uma construção alternativa para mostrar que todos os a_v são 0.

Caso 1: Existe um vértice não-especial v adjacente a C_{s1}.

Como C_{s1} é maximal, $(H \setminus C_{s1}) - v$ deve estar separado em j partes conexas $A_1, ..., A_j$, $j \geq 2$. Como $H \setminus C_{s1}$ é conexo, existe pelo menos uma aresta entre cada A_j e v. Como v não é um separador de H, existe pelo menos uma aresta entre cada A_j e C_{s1}. j não pode ser maior do que 3, porque haveria um componente conexo sem vértices especiais, ligado a C_{s1}; e C_{s1} não seria maximal. Se $j = 3$, teríamos um vértice especial em cada A_j. Nesse caso teríamos um caminho entre s_2 e t_2 passando por v, que não cortaria um caminho entre s_1 e t_1, o que é proibido pela definição de desigualdade da cruz. Logo $j = 2$ e s_2 e t_2 estão em A_js diferentes. Ajuste a notação para que s_2 e t_1 estejam em A_1 e t_2 esteja

em A_2 (por simetria, ainda é possível trocar os nomes de s_2 com t_2). A figura A.1 mostra essa configuração. As elipses representam subgrafos conexos de H.

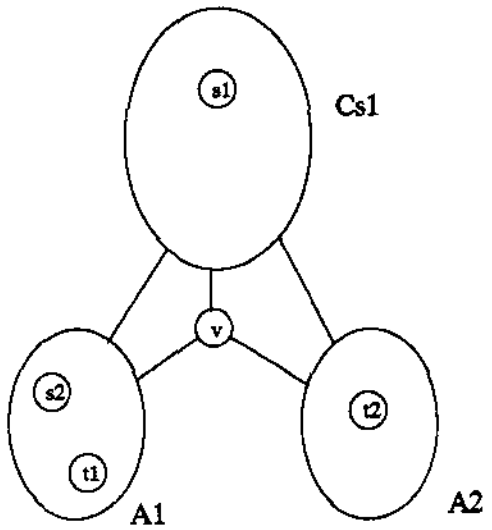


Figura A.1: Configuração de H no caso 1

Vamos examinar com mais detalhes a situação em A_2.

Subsubproposição A.2.1.1 *t_2 é adjacente a C_{s1}.*

Prova: Seja v_1 um vértice de A_2 adjacente a C_{s1}. Se v_1 é $t2$, a subsubproposição segue. Senão, v_1 separa A_2 em $j \geq 1$ componentes conexos. Seja A_2^2 o componente que contêm $t2$. A_2^2 não pode estar ligado a v, pois C_{s1} não seria maximal. Logo, como v_1 não é separador de H, A_2^2 está ligado a C_{s1}. Seja v_2 um vértice de A_2^2 adjacente a C_{s1}, ...

Procedendo recursivamente dessa forma, obtemos uma seqüência v_i de vértices adjacentes a C_{s1}, pertencentes a conjuntos cada vez menores A_2^i, e que contêm t_2. Quando v_i for t_2, a subsubproposição segue. Na pior das hipóteses, teremos que avançar até que $A_2^i = \{v_i\}$. □

t_2 separa A_2 em $j \geq 0$ componentes conexos. Como t_2 não é um separador de H, todos esses componentes estão ligados a v ou a C_{s1}. Afirmamos que todos eles estão ligados a v. Isso é verdade, pois se existisse algum desses componentes ligados a C_{s1} mas não a v, C_{s1} não seria maximal. A figura A.2 mostra a configuração detalhada de A_2. As elipses representam subgrafos conexos de A_2.

Agora vamos examinar a situação em A_1.

Subsubproposição A.2.1.2 *s_2 é adjacente a C_{s1}.*

Prova:

Caso a: t_1 é adjacente a C_{s1}.

t_1 separa A_1 em $j \geq 1$ componentes conexos. Seja A_1^2 o componente que contêm s_2. A_1^2 não pode estar ligado a v, pois senão teríamos um caminho entre s_2 e t_2 passando por

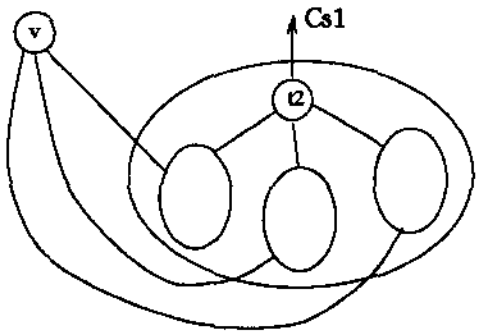


Figura A.2: Configuração detalhada de A_2 no caso 1

v, que não cortaria um caminho entre s_1 e t_1. Como t_1 não é separador, A_1^2 está ligado a C_{s1}.

Seja v_2 um vértice de A_1^2 adjacente a C_{s1}. Se $v_2 = s_2$, a subsubproposição segue. Senão, v_2 separa A_1^2 em $j \geq 1$ componentes conexos, todos ligados a C_{s1}, ...

Procedendo recursivamente dessa forma, obtemos uma seqüência v_i de vértices adjacentes a C_{s1}, contidos em conjuntos cada vez menores A_1^i, e que contêm s_2. Quando v_i for s_2, a subsubproposição segue. Na pior das hipóteses, teremos que avançar até que $A_1^i = \{v_i\}$.

Caso b: t_1 não é adjacente a C_{s1}.

Seja v_1 um vértice de A_1 adjacente a C_{s1}. Se v_1 é s_2, a subsubproposição segue. Senão, v_1 separa A_1 em A_j^2, $j \geq 1$ componentes conexos.

Subcaso b.1: s_2 e t_2 estão em componentes A_j^2 diferentes.

Seja A_1^2 o componente que contêm s_2. A_1^2 não pode estar ligado a v, pois senão teríamos um caminho entre s_2 e t_2 passando por v, que não cortaria um caminho entre s_1 e t_1 passando por v_1. Como v_1 não é separador, A_1^2 está ligado a C_{s1}. Procedendo recursivamente, a partir de A_1^2, de forma similar ao caso a, vemos que s_2 é adjacente a C_{s1}.

Subcaso b.2: s_2 e t_2 estão no mesmo componente A_j^2.

Seja A_j^2 o componente que contêm s_2 e t_2. A_j^2 não pode estar ligado a v, pois C_{s1} não seria maximal. Com v_1 não é um separador, A_j^2 está ligado a C_{s1}.

Seja v_2 um vértice de A_1^2 adjacente a C_{s1}. Se v_2 é s_2 a subsubproposição segue. Senão, v_2 separa A_1^2 em $j \geq 1$ componentes conexos A_j^3. Se s_2 e t_1 estão em componentes A_j^3 diferentes, podemos aplicar um procedimento similar ao do subcaso b.1. Se eles estiverem no mesmo componente A_1^3, procedemos recursivamente nesse mesmo subcaso b.2. Na pior das hipóteses teremos que avançar até que $A_1^i = \{s_2, t_1\}$. Como t_1 não é adjacente a C_{s1}, s_2 o será. □

Visto que s_2 é adjacente a C_{s1}, s_2 separa A_1 em $j \geq 1$ componentes conexos. Como s_2 não é um separador de H, todos esses componentes estão conectados ou a v, ou a C_{s1}.

Afirmamos que todos eles estão conectados a v. Se existisse algum desses componentes que não contêm t_1, conectados a C_{s1}, mas não a v, C_{s1} não seria maximal. Se o componente que contêm t_1 estiver ligado a C_{s1} mas não a v, haveria um caminho entre s_2 e t_2 passando por v, que não cortaria um caminho entre s_1 e t_1. A figura A.3 abaixo mostra a configuração detalhada de H nesse caso.

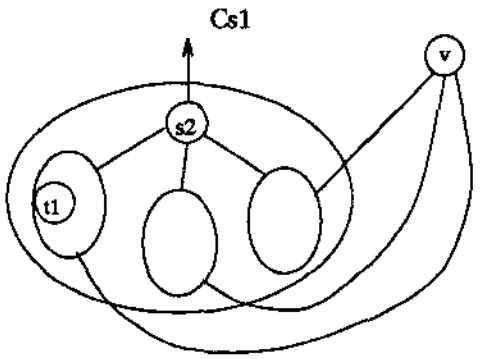


Figura A.3: Configuração detalhada de A_1 no caso 1

Definimos C_{t1} como $(A_1 \cup A_2 + v) - s_1 - t_1$. $C = C_{t1}$ é c-conexo e χ^C pertence a F_a. $C = C_{s1}$ também é c-conexo e χ^C pertence a F_a. Esses dois componentes cobrem todos os vértices não-especiais. Aplicando-se o lema, vemos que $a_v = 0$; $\forall v \in V \setminus E$ e subproposição segue.

Caso 2: Não existe um vértice não-especial v adjacente a C_{s1}.

Se todos os vértices não-especiais estão em C_{s1} a subproposição segue. Caso contrário, iremos construir um C_{t1} para "completar" C_{s1}, de modo similar ao que fizemos no caso 1.

Como H não tem separador, pelo menos 2 vértices especiais são adjacentes a C_{s1}.

Subcaso 2.1: Todos os três vértices especiais t_1, s_2 e t_2, são adjacentes a C_{s1}.

$(H \setminus C_{s1}) - t_1 - s_2 - t_2$ deve estar separado em j partes conexas $B_1, ..., B_j$, $j \geq 1$. Como H não tem separador, cada B_j está ligado a pelo menos dois vértices especiais. Mas B_j não pode estar ligado a s_2 e a t_2, pois haveria um caminho entre s_2 e t_2 passando por B_j, que não cortaria um caminho entre s_1 e t_1. Assim cada B_j está ligado a t_1.

Definimos C_{t1} como $(B_1 \cup ... \cup B_j) + t_1$. $C = C_{t1}$ é c-conexo e χ^C pertence a F_a. Aplicando-se o lema, vemos que $a_v = 0$; $\forall v \in V \setminus E$.

Subcaso 2.2: Existem dois vértice especiais adjacentes a C_{s1}.

Esses dois vértices devem ser s_2 e t_2, pois senão haveria um caminho entre s_2 e t_2, que não cortaria um caminho entre s_1 e t_1.

$(H \setminus C_{s1}) - s_2 - t_2$ deve estar separado em j partes conexas $B_1, ..., B_j$, $j \geq 1$. Seja B_1 o componente que contêm B_1. Se $j = 1$, definimos C_{t1} como B_1. Mas caso $j > 1$, ficamos bloqueados na situação mostrada na figura A.4.

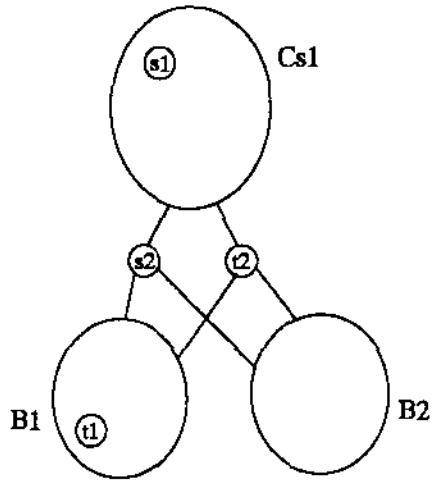


Figura A.4: Situação em que não é possível construir C_{t1} no subcaso 2.2

Evidentemente não é possível construir um C_{t1} com as propriedades desejadas, que juntamente com C_{s1} cubra todos os vértices não-especiais. Mas, notamos que esse bloqueio só pode acontecer se: (i) Não existir um caminho entre s_1 e t_1 em $H - s_2 - t_2$; (ii) Existir um caminho entre s_2 e t_2 em $H - s_1 - t_1$ (o caminho que passa por B_2).

Podemos, sem perda de generalidade, inverter os papéis dos pares (s_1, t_1) e (s_2, t_2) e repetir, desde o início, todo o procedimento de prova feito até agora.

Isso é possível, pois a desigualdade da cruz original em P_{γ_1} é simétrica. Ao fazer a substituição de variáveis para de definir P_{sim}, usamos uma convenção que quebrou arbitrariamente essa simetria. Mas nada nos impede de usarmos a convenção oposta e ainda termos a mesma desigualdade. A inversão é apenas um truque para facilitar a prova. Mesmo sem ela, seria possível mostrar que todos os a_v são 0 nos componentes B_j, $j > 1$; manipulando subgrafos c-conexos que contêm s_1, t_1 e, ou s_2 ou t_2 (mas não ambos). De fato, após a inversão, estaremos trabalhando com os subgrafos complementares a estes.

A figura abaixo dá um exemplo da inversão.

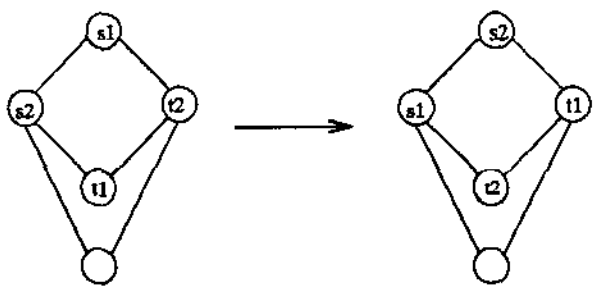


Figura A.5: Inversão dos pares (s_1, t_1) e (s_2, t_2)

Com a inversão, agora temos certeza que existe um caminho entre s_1 e t_1 em $H - s_2 - t_2$. Logo não ficaremos bloqueados nesse subcaso 2.2 e $a_v = 0$; $\forall v \in V \setminus E$. □

□

Esse resultado já mostra que a desigualdade da cruz é uma faceta em grande parte das instâncias que ocorrem na prática. Vamos agora caracterizar as condições em que a desigualdade da cruz é uma faceta de P_{sim} quando H é conexo mas tem separadores. Vamos começar com as condições em que a desigualdade da cruz não é faceta.

Proposição A.3 *Suponha que H é conexo e v é um separador de H. Se um dos componentes conexos de $H - v$ não contêm vértices especiais, então a desigualdade da cruz $x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$ não é uma faceta de P_{sim}.*

Prova: Seja u um vértice adjacente a v, num componente conexo A_1 de $H - v$ que não contêm vértices especiais. Afirmamos que a desigualdade da cruz é dominada pela seguinte desigualdade válida:

$$x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} + x'_v - x'_u \leq 1 \qquad (1)$$

Mesmo se v for um vértice especial, a desigualdade não muda. Nesse caso, deve-se somar 1 ao coeficiente original desse vértice.

Suponha que v não é s_2 ou t_2. Nesse caso, em todo C c-conexo tal que χ^C satisfaz a desigualdade da cruz na igualdade, $\chi^C_v = \chi^C_u$ e portanto χ^C também satisfaz (1) na igualdade. Mas $\chi^{H \setminus A_1}$ satisfaz (1) na igualdade, mas não satisfaz a desigualdade da cruz na igualdade. Logo a desigualdade da cruz é dominada por (1).

Se v for s_2 ou t_2, o resultado segue por simetria, fazendo-se a inversão dos pares. □

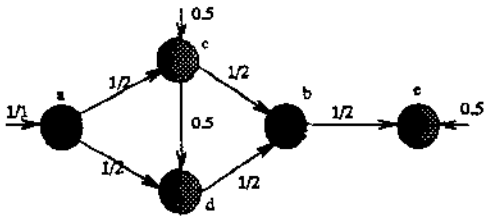


Figura A.6: Solução fracionária de F1 ($K = 2$, $\gamma_1 = 1$, $\gamma_2 = 1$).

A figura A.6 mostra uma solução fracionária de F1 que não é cortada por nenhuma desigualdade da cruz, mas que é cortada pela seguinte faceta de P_{γ_1}.

$$x_{a1} + 2x_{b1} + x_{c2} + x_{d2} + x_{e2} \leq 4 \qquad (2)$$

Essa faceta pode ser obtida pelo *lifting* da desigualdade da cruz envolvendo a, b, c e d indicado na proposição A.3.

Proposição A.4 *Suponha que H é conexo, v é um vértice especial e um separador de H e não estamos nas condições da proposição anterior. Então a desigualdade da cruz $x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$ não é uma faceta de P_{sim}.*

Prova: Suponha que v é s_2 (por simetria isso é sempre possível). Afirmamos que a desigualdade da cruz é dominada pela seguinte desigualdade válida:

$$x'_{s_1} + x'_{t_1} - x'_{s_2} \leq 1 \qquad (3)$$

Se s_1 e t_1 estiverem no mesmo componente conexo de $H - v$, como não estamos nas condições da proposição anterior, t_2 obrigatoriamente deverá estar em outro componente conexo de $H - v$. Mas nesse caso, haveria caminhos disjuntos entre esses pares de vértices. Logo s_2 separa s_1 de t_1 e (3) é uma desigualdade válida.

Claramente, (3) domina a desigualdade da cruz. □

Proposição A.5 *Suponha que que H é conexo, v é um separador de H que separa s_1 de t_1 e s_2 de t_2, e não estamos nas condições das duas proposições anteriores. Então a desigualdade da cruz $x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$ não é uma faceta de P_{sim}.*

Prova: Afirmamos que a desigualdade da cruz é dominada pela seguinte desigualdade válida:

$$x'_{s_1} + x'_{t_1} - x'_{v} \leq 1 \qquad (4)$$

Nas condições da proposição, em todo C c-conexo tal que χ^C satisfaz a desigualdade da cruz na igualdade, $\chi^C_v = \chi^C_{s_2} + \chi^C_{t_2}$ e portanto χ^C também satisfaz (4) na igualdade. Mas χ^H satisfaz (4) na igualdade, mas não satisfaz a desigualdade da cruz na igualdade. Logo a desigualdade da cruz é dominada por (4). □

Com os três resultados já obtidos até agora para o caso de H conexo, mas com separadores, podemos afirmar o seguinte:

Proposição A.6 *Suponha que H é conexo e que a desigualdade da cruz $x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$ é uma faceta de P_{sim}. Então não há separadores v que dividam $H - v$ em mais de 3 componentes conexos. Além disso, se v separa $H - v$ em 2 componentes conexos, há 3 vértices especiais em um dos dois componentes conexos de $H - v$.*

Prova: Seja v um vértice separador de H. Pela proposição A.4, sabemos que v não é um vértice especial. Pela proposição A.3, sabemos que cada componente conexo de $H - v$ deve conter pelo menos um vértice especial. Pelas proposição A.5, vemos que $H - v$ não pode estar separado em 4 ou mais componentes conexos. Se $H - v$ estiver separado em 3 componentes conexos, os vértices especiais s_1 e t_1 (ou s_2 e t_2) não podem estar no mesmo componente conexo, ou teríamos caminhos disjuntos entre os dois pares de vértices. Assim, v separaria os dois pares de vértices, o que é proibido pela proposição A.5. A primeira afirmação segue.

A segunda afirmação segue diretamente da proposição A.5. □

Um grafo H com separadores pode ser decomposto em componentes biconexos. Os separadores são justamente os "pontos de articulação" comuns a dois ou mais desses componentes biconexos. Por essa razão, os componentes biconexos são definidos como um conjunto de arestas.

Suponha que todos os separadores v em H, dividem $H - v$ em dois componentes conexos. Construímos a árvore T da decomposição biconexa de H da seguinte forma: cada nó de T corresponde a um componente biconexo de H. As arestas de T correspondem aos componentes biconexos de H com um vértice separador em comum.

Vamos agora mostrar as condições em que desigualdade da cruz é uma faceta de P_{sim}, quando H é conexo, mas tem separadores. Antes de mais nada, precisamos saber a dimensão do poliedro nessas condições.

Proposição A.7 *Suponha que H é conexo e que todos os seus separadores v, dividem $H - v$ em dois componentes conexos. Então P_{sim} tem dimensão cheia.*

Prova: O vetor 0 está em P_{sim}. Para cada vértice $i \in H$ não separador, o vetor com $\chi^{\{i\}}$ está em P_{sim}.

Escolha um nó da árvore T da decomposição biconexa de H para ser a raiz R (R corresponde a um conjunto de arestas em H). Seja v um vértice que separa H nos componentes conexos A_1 e A_2. Seja A_1 o componente que contêm ou é adjacente as arestas de R. O vetor de incidência χ^{A_2+v} está em P_{sim}.

Temos então $|V| + 1$ vetores afim independentes em P_{sim}. □

Proposição A.8 *Suponha que H é um grafo conexo. Suponha que para qualquer v separador de H, as seguintes propriedades são verdadeiras: (i) v não é um vértice especial; (ii) todos os componentes conexos de $H - v$ contêm vértices especiais; (iii) v não separa s_1 de t_1 e s_2 de t_2. Então a desigualdade da cruz $x'_{s_1} + x'_{t_1} - x'_{s_2} - x'_{t_2} \leq 1$ é uma faceta de P_{sim}.*

Esboço de Prova: A prova completa que encontramos para esse caso é muito trabalhosa, mas as idéias são basicamente as mesmas usadas na proposição A.2. Optamos por apresentá-la de forma resumida.

Como na proposição A.2, o objetivo é mostrar que qualquer faceta de P_{sim} da forma $a^t x' = \alpha$, que inclua a desigualdade da cruz, é um múltiplo escalar dessa desigualdade. Ainda é verdade que $a_{s_1} = a_{t_1} = \alpha$. O passo crítico, e que depende de uma análise de vários casos, é mostrar que $a_v = 0 \ ; \forall v \in V \setminus E$. Com esse resultado mostra-se que $a_{s_2} = a_{t_2} = -\alpha$.

Para mostrar que $a_v = 0 \ ; \forall v \in V \setminus E$, temos que analisar a estrutura de H. Seja T a árvore da decomposição biconexa de H. Os nós de T correspondem a conjuntos de arestas em T, mas também correspondem a conjuntos de vértices em H. A diferença é que

cada aresta pertence a exatamente um componente biconexo, mas os vértices separadores pertencem a dois componentes biconexos. Mas, por (i), todos os vértices especiais estão associados a um único componente biconexo. Assim, podemos falar sem ambiguidade no componente biconexo que contêm um certo vértice especial.

Por (ii), sabemos que todas as folhas de T devem conter um vértice especial. Isso mostra que T é uma árvore com no máximo 4 folhas.

Caso 1: Existe um par, s_1 e t_1, ou s_2 e t_2, no mesmo componente biconexo.

Suponha que s_2 e t_2 estão no mesmo componente biconexo A_1 de T. T é uma árvore de grau máximo 2, ou seja, uma linha. Além disso, s_1 e t_1 estão nos componentes biconexos de grau 1 dessa árvore (é possível que A_1 seja um componente biconexos de grau 1). A figura A.7 abaixo mostra a configuração básica de H no caso 1. Aqui, os círculos grandes representam os componentes biconexos e as intersecções entre eles representam os vértices separadores.

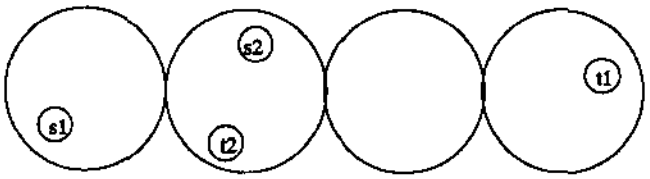


Figura A.7: Possível configuração de H no caso 1

Podemos usar técnicas semelhantes às da proposição A.2 para construir seqüências encaixadas de conjuntos c-conexos, contendo s_1 ou t_1, mas não contendo s_2 e t_2, de forma a cobrir todos os vértices não especiais v. Dessa forma, mostra-se que $a_v = 0 \; ; \forall v \in V \setminus E$.

Caso 2: Não estamos nas condições do caso 1, mas existem dois vértices especiais, no mesmo componente biconexo.

Suponha que s_1 e s_2 estão no mesmo componente biconexo A_1 de H. T também é uma árvore de grau máximo 2, ou seja, uma linha. Além disso, t_1 e t_2 estão nos componentes biconexos de grau 1 dessa árvore.

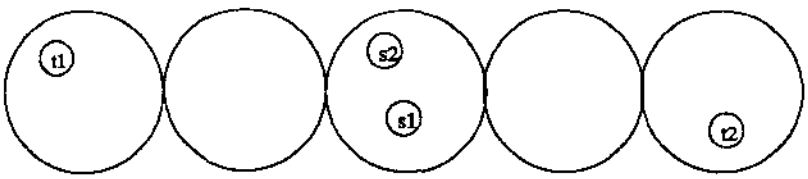


Figura A.8: Possível configuração de H no caso 2

Esse caso é um pouco mais complicado. Seja v o separador contido em A_1 que isola t_2 dos demais vértices especiais. Podemos usar técnicas semelhantes às do caso 1 para construir seqüências encaixadas de conjuntos c-conexos, contendo s_1 ou t_1, mas não contendo s_2 e t_2, cobrindo todos os vértices não especiais v contidos no componente conexo de $H - v$ que não contêm t_2.

Mas para os vértices não especiais do componente conexo de H v que contêm t_2, e para v, temos que construir seqüências encaixadas de conjuntos c-conexos contendo s_1, t_1, e s_2, mas não contendo t_2. Ao contrário do que acontece na proposição A.2, aqui não adianta inverter os pares de vértices.

Caso 3: Cada vértice especial está num componente biconexo distinto.

É possível mostrar que nesse caso, H deve ter uma configuração semelhante a da figura A.9.

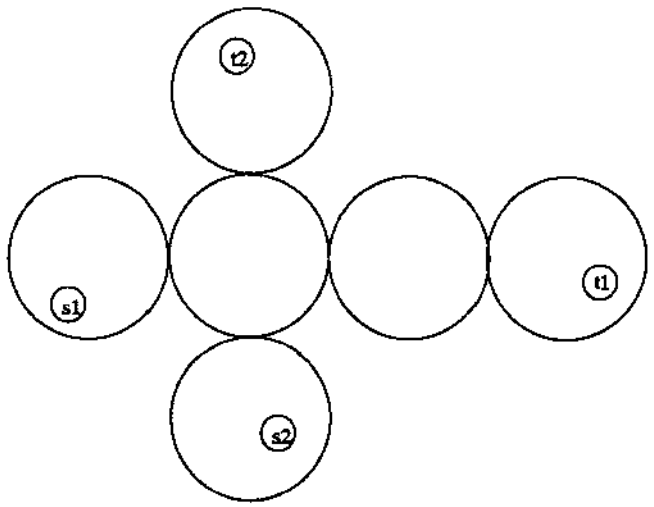


Figura A.9: Possível configuração de H no caso 3

Nesse caso construímos as seqüências encaixadas de forma similar ao caso 2. □

A.1 Comentários

- Nesse apêndice, estudamos as condições em que a desigualdade da cruz é uma faceta, para o caso $K = 2$. A dificuldade de estender esse estudo para um valor de K qualquer, é trabalhar com um poliedro que não é de dimensão cheia. É possível que essa dificuldade técnica possa ser contornada trabalhando-se diretamente com a definição de faceta, ou seja, procurando-se achar $\dim(P_{\gamma_1})$ vetores afim independentes em P_{γ_1} que satisfaçam a desigualdade da cruz na igualdade.

  Conforme apresentando no capítulo 4, a desigualdade da cruz é apenas um caso particular da desigualdade da q-cruz, que por sua vez é um caso particular da desigualdade da q-cruz em $H - R$. Todas essas desigualdades são válidas para P_{γ_1}. A dificuldade de estender o estudo que fizemos para essas classes é maior. Teríamos que entender melhor as propriedades combinatórias do problema, sob pena de termos uma explosão no número de casos a serem tratados.

Mas, a classe de desigualdades mais interessante são as desigualdades da q-cruz generalizadas. Essa classe generaliza todas as classes anteriores para o PAgC com valores quaisquer de γ_k. A dificuldade de estender o estudo que fizemos para essa classe seria ainda maior. O problema é que essa classe é válida para o poliedro aumentado P, definido nas variáveis x e y.

De qualquer maneira, o fato de a desigualdade da cruz ser uma faceta de P_{sim} na grande maioria dos casos, é um argumento a favor da força das demais classes de desigualdades.

- A desigualdade da q-cruz generalizada não apenas generaliza a desigualdade da cruz, mas é mais forte do que esta. Na figura A.6 mostramos uma solução fracionária que podia ser cortada por uma desigualdade da cruz após a aplicação do *lifting* indicado na proposição A.3. Essa mesma solução fracionária pode ser cortada diretamente por uma desigualdade da 2-cruz generalizada, o que dizer que essa desigualdade, de certa forma, já incorpora esse *lifting*.

Bibliografia

[1] M. Andberg, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[2] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh e P. Vance, *Branch-and-Price: Column Generation for Solving Huge Integer Programs*, Operations Research, a ser publicado, 1996.

[3] J. Beasley, *A Note on Solving Large p-median Problems*, European Journal of Operational Research, 21, 270-273, 1985.

[4] R. Bellman e S. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, New Jersey, 1962.

[5] J. Bourjolly, G. Laporte e J. Rousseau, *Decoupage Electoral Automatise*, INFOR, 19(2), May, 1981.

[6] N. Christofides e J. Beasley, *A Tree-search Algorithm for the p-median Problem*, European Journal of Operational Research, 10, 196-204, 1982.

[7] CPLEX 3.0, *Using the CPLEX Callable Library*, CPLEX Optimization, Inc., 1994.

[8] M. Delattre e P. Hansen, *Bicriterion Cluster Analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI2, 277-291, 1980.

[9] D. Erlenkotter, *A Dual-Based Procedure for Uncapacited Facility Location*, Operations Research, 26, 1590-1602, 1978.

[10] B. Everitt, *Cluster Analysis*, Edward Arnold Press, London, 1993.

[11] C. Ferreira e Y. Wakabayashi, *Combinatória Poliédrica e Planos-de-Corte Faciais*, Texto da 10ª Escola de Computação, IC-UNICAMP, 1996.

[12] M. Garey e D. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[13] A. Geoffrion, *Lagrangean Relaxation for Integer Programming*, Mathematical Programming Study 2, 82-114, 1974.

[14] A. Gordon, *Classification: Methods for the Exploratory Analysis of Multivariate Data*, Chapman and Hall, New York, 1981.

[15] P. Hansen, B. Jaumard, B. Simeone e V. Doring, *Maximum Split Clustering Under Connectivity Constraints*, Les Cahiers du GERAD, G-93-06, Montréal, 1993.

[16] P. Hansen e B. Jaumard, *Computational Methods in Clustering from a Mathematical Programming Viewpoint* , Les Cahiers du GERAD, G-95-29, Montréal, 1995.

[17] L. Hubert, *Monotone Invariant Clustering Procedures*, Psychometrika, 38(1), 47-62, 1973.

[18] A. Jain e R. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, New Jersey, 1988.

[19] E. Johnson, A. Mehotra e G. Nemhauser, *Min-Cut Clustering*, Report COC-93-21, Georgia Institute of Technology, Atlanta, 1993.

[20] M. Jünger, G. Reinelt, S. Thienel, *Practical Problem Solving with Cutting Plane Algorithms*, In W. Cook, L. Lovász e P. Seymour, editors, *Combinatorial Optmization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, p. 111-152, American Mathematical Society, 1995.

[21] T. Kariya e B.Sinha, *Robustness of Statistical Tests*, Academic Press, Boston, 1989.

[22] L. Kaufman e P. Rousseeuw, *Finding Groups in Data: an introduction to cluster analysis*, John Wiley, New York, 1990.

[23] M. Korkel, *On the Exact Solution of Large-scale Simple Plant Location Problems*, European Journal of Operational Research, 39, 157-173, 1989.

[24] A. Lucena e J. Beasley, *A Branch and Cut Algorithm for the Steiner Problem in Graphs*, Report, Management School, Imperial College, 1992.

[25] A. Lucena e J. Beasley, *Branch and Cut Algorithms for Integer Programming*, Capítulo 5 de "Advances in Linear and Integer Programming", Oxford University Press, 1996.

[26] T. Magnanti e L. Wolsey, *Optimal Trees*, CORE Discussion Paper 9426, Université Catholique de Louvain, Louvain-la-Neuve, 1994.

[27] M. Maravale e B. Simeone, *A Spanning Tree Heuristic for Regional Clustering*, Research Report 29 (serie A), Dipartimento di Statistica, Probabilitáe Statistiche Applicate, Universita degli studi de Roma "La Sapienza", 1990.

[28] F. Murtagh, *A Survey of Algorithms for Contiguity-constr... lated Problems*, The Computer Journal, 28-1, 1985.

[29] S. Näher e C. Uhrig, *The LEDA User Manual Version R ...* für Informatik, Saarbrücken, 1994.

[30] G. Nemhauser e L. Wolsey, *Integer and Combinatorial Optim...* York, 1988.

[31] M. Padberg e G. Rinaldi, *A Branch-and-Cut Algorithm fo... Scale Symmetric TSPs*, SIAM Review, 33:60-100, 1991.

[32] Y. Perl e Y. Shiloach, *Finding Two Disjoint Paths Between... a Graph*, Journal of the ACM, Vol. 25, No.1, Janeiro 1978.

[33] M. Rao, *Cluster Analysis and Mathematical Programming*... tistical Association, 66, 622-626, 1971.

[34] N. Robertson e P. Seymour, *An Outline of a Disjoint Pa...* L. Lovász, H. Prömel, A. Schrijver (Eds.) Paths, Flows an... Verlag, Berlim, 1989.

[35] D. Ryan e B. Foster, *An Integer Programming Approach to ...* Computer Scheduling of Public Transport Urban Passenge... duling, North-Holland, 269-280, Amsterdam, 1981.

[36] A. Schrijver, *Homotopic Routing Methods*, B. Korte, L. Le... rijver (Eds.) Paths, Flows and VLSI-Layout, Springer-Verl...

[37] Y. Shiloach, *A Polynomial Solution to The Undirected Tw...* of the ACM, Vol. 27, No.3, Julho 1980.

[38] R. Singleton e W. Kautz, *Minimum Squared Error Clust...* Research Institute, 1965.

[39] H. Späth, *Cluster Analysis Algorithms*, Ellis Horwood, Chi...

[40] M. Sol e M. Savelsbergh, *A Branch-and-Price Algoritm fo... with Time Windows*, Report COC-94-06, Georgia Institut... 1994.

[41] S. Thienel, *ABACUS - A Branch-And-Cut System*, PhD th... 1995.

[42] F. Vanderbeck, *Decomposition and Column Generation for Integer Programs*, PhD Thesis, Université Catholique de Louvain, Louvain-la-Neuve, 1994.

[43] H. Vinod, *Integer Programming and the Theory of Grouping*, Journal of American Statistical Association, 64, 506-517, 1969.