Algoritmos para Problemas de Classificação e Particionamento em Grafos

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Luís Augusto Angelotti Meira e aprovada pela Banca Examinadora.

Campinas, 13 de Dezembro de 2007.

Flávio Keidi Miyazawa (Orientador)

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

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues CRB8a / 2116

Meira, Luís Augusto Angelotti
M478a Algoritmos para problemas de classificação e particionamento em grafos / Luís Augusto Angelotti Meira -- Campinas, [S.P. :s.n.], 2007.

Orientador : Flávio Keidi Miyazawa
Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Algoritmos - Programação (Matemática). 3. Programação inteira. I. Miyazawa, Flávio Keidi. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Algorithms for classification and partitioning in graphs.

Palavras-chave em inglês (Keywords): 1. Combinatorial optimization. 2. Algorithms – Mathematical programming. 3. Integer programming.

Área de concentração: Otimização Combinatória

Titulação: Doutor em Ciência da Computação

Banca examinadora:

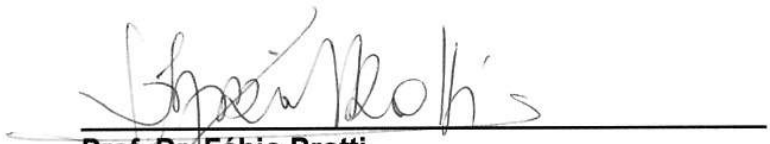
Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)
Prof. Dr. Cid Carvalho de Souza (IC-UNICAMP)
Profa. Dra. Cristina Gomes Fernandes (IME-USP)
Prof. Dr. Fábio Protti (NCE-UFRJ)
Prof. Dr. Orlando Lee (IC-UNICAMP)

Data da defesa: 13-12-2007

Programa de pós-graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

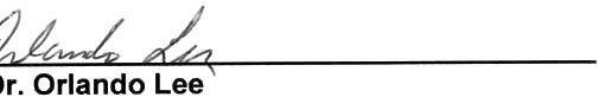
Tese Defendida e Aprovada em 13 de dezembro de 2007, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Fábio Protti
NCE - UFRJ.

Profª. Drª. Cristina Gomes Fernandes
IME - USP.

Prof. Dr. Cid Carvalho de Souza
IC - UNICAMP.



Prof. Dr. Orlando Lee
IC - UNICAMP.

Prof. Dr. Flávio Keidi Miyazawa
IC - UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos para Problemas de Classificação e Particionamento em Grafos

Luís Augusto Angelotti Meira*

Dezembro de 2007

Banca Examinadora:

- Flávio Keidi Miyazawa (Orientador)
- Cid Carvalho de Souza
  IC-UNICAMP.
- Cristina Gomes Fernandes
  IME-USP.
- Fábio Protti
  NCE-UFRJ.
- Orlando Lee
  IC-UNICAMP.

*Pesquisa financiada em parte por FAPESP, CNPq, ProNEx–FAPESP/CNPq e Receita Federal.

Resumo

O trabalho desenvolvido neste doutorado consistiu em conceber algoritmos para uma série de problemas NP-difíceis sob a abordagem de aproximabilidade, complementado com resultados heurísticos e também de programação inteira. O estudo foi focado em problemas de classificação e particionamento em grafos, como classificação métrica, corte balanceado e clusterização. Houve um equilíbrio entre teoria e aplicabilidade, ao obter-se algoritmos com bons fatores de aproximação e algoritmos que obtiveram soluções de qualidade em tempo competitivo. O estudo concentrou-se em três problemas: o Problema da Classificação Métrica Uniforme, o Problema do Corte Balanceado e o Problema da Localização de Recursos na versão contínua.

Inicialmente trabalhamos no Problema da Classificação Métrica Uniforme, para o qual propusemos um algoritmo $O(\log n)$-aproximado. Na validação experimental, este algoritmo obteve soluções de boa qualidade em um espaço de tempo menor que os algoritmos tradicionais.

Para o Problema do Corte Balanceado, propusemos heurísticas e um algoritmo exato. Experimentalmente, utilizamos um resolvedor de programação semidefinida para resolver a relaxação do problema e melhoramos substancialmente o tempo de resolução da relaxação ao construir um resolvedor próprio utilizando o método de inserção de cortes sobre um sistema de programação linear.

Finalmente, trabalhamos com o problema de Localização de Recursos na variante contínua. Para este problema, apresentamos algoritmos de aproximação para as métricas l_2 e l_2^2. Este algoritmo foi aplicado para obter algoritmos de aproximação para o problema k-Means, que é um problema clássico de clusterização. Na comparação experimental com uma implementação conhecida da literatura, os algoritmos apresentados mostraram-se competitivos, obtendo, em vários casos, soluções de melhor qualidade em tempo equiparável.

Os estudos relativos a estes problemas resultaram em três artigos, detalhados nos capítulos que compõem esta tese.

Abstract

We present algorithms for combinatorial optimization NP-hard problems on classification and graph partitioning. The thesis concerns about theory and application and is guided by an approximation algorithms approach, complemented with heuristics and integer programming. We proposed good approximation factor algorithms as well as algorithms that find quality solutions in competitive time. We focus on three problems: the Metric Labeling Problem, the Sparsest Cut Problem and the Continuous Facility Location Problem.

For the Metric Labeling Problem, we proposed an $O(\log n)$-approximation algorithm. In the experimental analysis, this algorithm found high quality solutions in less time than other known algorithms.

For the Sparsest Cut Problem we proposed heuristics and an exact algorithm. We built an SDP Solver to the relaxed formulation using a semi-infinity cut generation over linear programming. This approach considerably reduces the time used to solve the semidefinite relaxation compared to an open source semidefinite programming solver.

Finally, for the Continuous Facility Location Problem we present approximation algorithms to the l_2 and l_2^2 distance function. These algorithms are used to obtain approximation algorithms to the k-Means Problem, which is a basic clustering problem. The presented algorithms are competitive since they obtain in many cases better solutions in equivalent time, compared to other known algorithms.

The study of these problems results in three papers, which are detailed in chapters that make this thesis.

Agradecimentos

Certa vez, na sala do café, encontrei um professor em dia atribulado que desatou seu queixume: “Aproveite bem. O doutorado é a melhor fase.” Se isto é verdade, será o tempo que dirá. Somente ele, cada vez mais apressado, poderá dar a dimensão correta desta época da vida.

Foram anos de trabalho, umas vezes solitário, na sala 71. Os teoremas se enroscavam final de semana a fora e, segunda pela manhã, explicava em detalhes a meu orientador. Com olhar atento me questionava, e o processo se reiniciava. A Flávio Miyazawa, por ser orientador, amigo e exemplo, por ter ensinado com calma e respeito, meus sinceros agradecimentos.

Mas, para chegar neste ponto, tive apoio incondicional de meus pais e de minha família. Ao meu pai, João Martinho, que desde pequeno me incentivou a estudar e me passou o gosto pela matemática; a minha mãe, Emília, que sempre esteve presente com seu amor; a minhas irmãs, Lilian e Liziane, irmãs e amigas; a vocês quatro, mais que agradeço, dedico este doutorado.

Outras pessoas fizeram parte de minha vida nesta época. A meus colegas de república, Wilson e Anderson, a meus amigos de graduação, em especial ao Fábio, ao Paulinho e ao Ivo, a meus colegas de mestrado, em especial ao Cláudio, a meus irmãos de orientação, Evandro, Eduardo, André e Carlos, aos meus amigos de doutorado, em especial à Cândida e ao Gustavo, a meus colegas de Harpia, Norton, Rezende e os demais. Ao Gate, da cantina, aos professores que muito me ensinaram e com quem pude ter rica convivência nestes anos de Unicamp, Cid, Bauzer, Orlando, Dahab, Cristina, Tomasz, Lucchesi, Baranauskas, Meidanis, Rezende, Edmundo, Guido, Islene, Anido, Rodolfo e Siome. A todos vocês meu muito obrigado.

Quero agradecer a Antônio Carvalho, o Dr. Antônio, pelo apoio e fortalecimento para enfrentar as dificuldades e também expressar meu sentimento de gratidão pela Unicamp, esta mãe rigorosa me fez crescer, por Campinas, que me acolheu, e por todos aqueles que fizeram parte desta caminhada.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Nesta tese investigamos alguns problemas de Otimização Combinatória, voltados para classificação e particionamento de elementos. Para entender melhor a natureza destes problemas, descreveremos algumas situações onde eles ocorrem.

Suponha que desejamos descobrir regiões críticas na malha viária da grande São Paulo e tudo que possuímos é um mapa com a descrição das ruas e a densidade populacional. O problema do corte balanceado estudado nesta tese constitui-se na formulação natural para encontrar regiões suscetíveis a engarrafamentos em uma malha.

Dado um grafo G, com conjunto de vértices V e conjunto de arestas E, a solução para o problema do corte balanceado é a partição dos vértices em dois grupos. Duas medidas são feitas sobre esta partição. Uma função mede o peso das arestas entre as partições e outra função mede o equilíbrio entre o tamanho dos dois conjuntos, multiplicando seus tamanhos. A função objetivo a ser minimizada consiste na divisão do peso das arestas entre as partes pelo produto do tamanho das partes. Veja Figura 1.1.

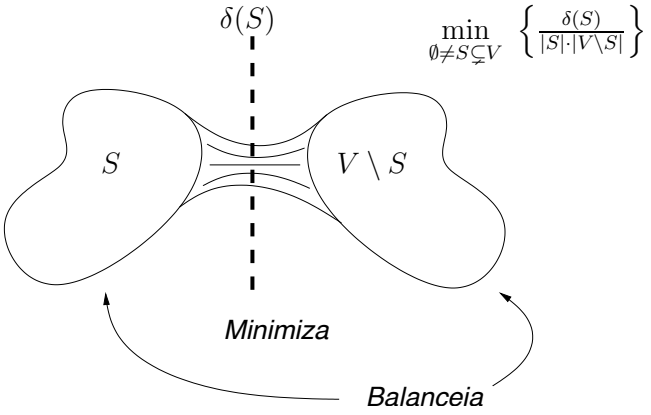


Figura 1.1: Problema do Corte Balanceado em um grafo $G(V, E)$.

Se quebrarmos a malha viária de São Paulo em regiões, conectarmos regiões vizinhas por arestas contendo a capacidade das ruas entre elas e ponderarmos o peso de acordo com sua população, algoritmos desenvolvidos para o corte balanceado devolverão uma partição das regiões em dois grupos que terão a característica de serem grandes em relação ao fluxo possível entre eles.

Nesta tese também investigamos problemas de clusterização e de localização de recursos, que são básicos dentro da otimização combinatória. Toda vez que a quantidade de informação é demasiada, um algoritmo de clusterização consegue extrair os representantes mais significativos. Por exemplo, uma imagem com um milhão de cores pode ser comprimida em outra com 512 cores. Encontrar as 512 cores que causam menor distorção na qualidade da imagem é um problema de clusterização. Já um problema de situar fornecedores como postos de correios, bancos, hospitais, pontos de informação, servidores, etc, podem ser naturalmente descritos como um problema de localização de recursos.

Na Figura 1.2, mostramos um exemplo de problema de clusterização, o k-Means. Neste problema temos um conjunto de pontos no plano e procuramos por k centros dentro do domínio de forma que a soma das distâncias entre cada ponto e o centro mais próximo seja minimizada. A função distância para este problema é tradicionalmente a euclidiana ao quadrado, também conhecida como l_2^2.

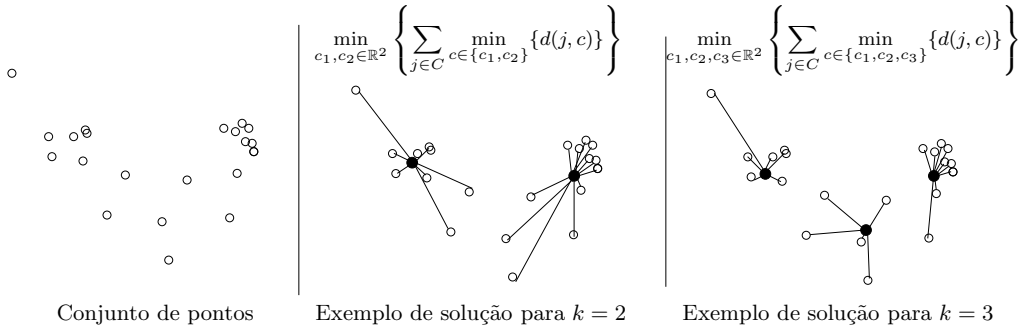


Figura 1.2: Problema k-Means.

Nesta tese também investigamos problemas de classificação, como o da Classificação Métrica Uniforme. Em um problema de classificação tradicional, deseja-se conectar cada um dos n objetos a uma das m classes. Essa classificação deve ser consistente com alguns dados observados, os quais incluem a relação entre os pares de objetos a serem classificados e a relação entre objetos e classes.

Na Figura 1.3 ilustramos o problema. Temos uma série de animais representando os objetos do problema: gato, morcego, baleia, etc, e quatro classes: mamíferos, aves, répteis e peixes, que apesar de serem agora classes bem definidas, poderiam estar em algum momento indefinidas. Existe um custo de atribuição entre cada objeto e cada classe.

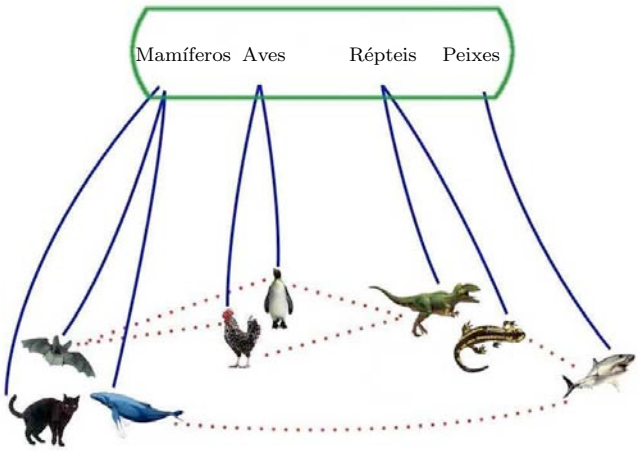


Figura 1.3: Problema da Classificação Métrica Uniforme.

Uma baleia, por exemplo, tem um custo de atribuição baixo com a classe mamíferos e um custo alto com a classe aves. O custo de atribuição é pago toda vez que um objeto é atribuído a uma classe. Existe também o custo de associação entre quaisquer dois objetos. Ele guarda a informação de similaridade entre os objetos. Toda vez que dois objetos vão para classes diferentes, o custo de associação entre eles é pago.

O objetivo do problema Classificação Métrica Uniforme *(Uniform Metric Labeling Problem - UMLP)* consiste em encontrar uma atribuição dos objetos em classes de forma a minimizar a soma dos custos de atribuição e de associação.

Os problemas investigados nesta tese estão na classe NP-difícil. Para tais problemas não se espera que existam algoritmos eficientes que os resolvam. A existência ou não de algoritmos exatos e polinomiais é uma das questões que permeia a Ciência da Computação há mais de 35 anos. Assumindo a conjectura de que tais algoritmos não existem, ou simplesmente assumindo que $P \neq NP$, algoritmos exatos implicam uma complexidade de tempo supra-polinomial para qualquer problema NP-difícil. Sendo assim, a tese foi principalmente guiada pelo desenvolvimento de *Algoritmos Aproximados*, uma área que tem se revelado crescentemente promissora dentro da Ciência da Computação.

1.1 Algoritmos de Aproximação

Nesta seção será dada uma breve introdução a algoritmos de aproximação, mostrando-se a notação utilizada e também conceitos básicos sobre o tema.

Dado um algoritmo $\mathcal{A}$ para um problema de minimização, seja I uma instância para esse problema e $\mathcal{A}(I)$ o valor da solução devolvida pelo algoritmo $\mathcal{A}$ aplicado à instância I.

Seja OPT(I) o valor de uma solução ótima de I. Um algoritmo tem um fator de aproximação α, ou é α-aproximado, se $\mathcal{A}(I)/\text{OPT}(I) \leq \alpha$ para toda instância I. No caso dos algoritmos probabilísticos, considera-se a desigualdade $E[\mathcal{A}(I)]/\text{OPT}(I) \leq \alpha$, onde a esperança $E[\mathcal{A}(I)]$ é tomada sobre todas as escolhas aleatórias feitas pelo algoritmo. É importante ressaltar que algoritmos de aproximação, considerados neste trabalho, têm complexidade de tempo polinomial, a menos que seja explícito o contrário. Algoritmos probabilísticos podem ser considerados com complexidade de tempo polinomial quando a esperança do tempo de execução for polinomial. Definições e conceitos para problemas de maximização podem ser apresentados de maneira análoga, porém tais problemas não são investigados nesta tese.

Ao se elaborar um algoritmo aproximado, um passo importante é buscar uma prova de seu fator de aproximação. Outro aspecto que merece ser ressaltado é a necessidade de verificar se o fator de aproximação α demonstrado é o melhor possível para o algoritmo. Para tanto, deve-se encontrar uma instância cuja razão entre o valor da solução obtida pelo algoritmo e valor de uma solução ótima é igual, ou tão próxima quanto se queira, de α. Nesse caso, dizemos que o fator de aproximação do algoritmo é justo, não podendo ser melhorado. Outro algoritmo pode ter um fator de aproximação mais próximo de 1, desde que não entre em contradição com limites de inaproximabilidade.

Quando a existência de um fator de aproximação α implicar a quebra de uma conjectura bem estabelecida como, por exemplo, que $P \neq NP$, diz-se que este valor α é um limite de inaproximabilidade.

Quanto menor o fator de aproximação de um algoritmo para um problema de minimização, maior a garantia de que o valor da solução obtida seja próximo ao de uma solução ótima. Logo, fatores menores são mais desejados. Para que um algoritmo possa ser aplicado a problemas reais, é desejável também que possua complexidade de tempo descrita por um polinômio de grau baixo.

Em alguns casos é possível obter um algoritmo polinomial $(1+\epsilon)$-aproximado para qualquer $\epsilon > 0$ fixo. A esta família de algoritmos damos o nome de PTAS (Polinomial Time Approximation Scheme). Um PTAS com complexidade de tempo polinomial também em $1/\epsilon$ é chamado de FPTAS (Fully Polinomial Time Approximation Scheme).

1.2 Problemas Investigados e Resultados

Obtivemos resultados para um problema de classificação, conhecido como Classificação Métrica Uniforme, para o Problema do Corte Balanceado e para o Problema de Localização de Recursos em sua versão contínua. Os resultados nesse último problema foram aplicados para obtenção de um algoritmo para o k-Means. As próximas seções detalham estes problemas e os respectivos resultados.

1.2.1 Problema da Classificação Métrica Uniforme

Esta tese investigou um problema de classificação, conhecido como Classificação Métrica Uniforme. Ele consiste em classificar um conjunto de objetos em classes de forma a minimizar custos. Sobre uma classificação são calculadas duas funções: a primeira mede o custo de atribuição entre objetos e classes; a segunda mede o custo para separar objetos que possuam similaridade. O objetivo do problema é encontrar uma classificação dos objetos em classes que minimize a soma dos custos de atribuição e de separação. A seguir descrevemos o problema de maneira formal.

Seja P o conjunto de objetos, L o conjunto de classes, $w : P \times P \to \mathbb{R}^+$ uma função peso, $d : L \times L \to \mathbb{R}^+$ uma função de distância e $c : P \times L \to \mathbb{R}^+$ uma função de custo de atribuição. Uma *classificação* de P em L é uma função $\phi : P \to L$. O *custo de atribuição* de uma classificação ϕ é a soma $\sum_{i \in P} c(i, \phi(i))$ e o *custo de separação* de uma classificação ϕ é a soma $\sum_{i,j \in P} d(\phi(i), \phi(j))w(i, j)$. A função w indica a força de relacionamento entre dois objetos, e a função d indica a similaridade entre duas classes. O custo de uma classificação ϕ é a soma dos custos de atribuição com os custos de separação. Definimos os tamanhos dos conjuntos P e L como n e m, respectivamente.

O Problema da Classificação Métrica (*Metric Labeling Problem - MLP*) consiste em encontrar uma classificação de objetos em classes com custo total mínimo. Neste problema, toda vez que dois objetos vão para classes diferentes, paga-se o custo de separação vezes a distância entre as classes. O foco de nossa pesquisa foi o problema da *Classificação Métrica Uniforme* (*Uniform Metric Labeling Problem - UMLP*), no qual as distâncias valem 1 para classes diferentes, e 0 caso contrário.

O problema MLP tem várias aplicações, muitas descritas por Kleinberg e Tardos [33]. Algumas dessas aplicações ocorrem em processamento de imagens [4, 13, 18], biometria [3], classificação de textos [8], *etc.* Um exemplo de aplicação é a restauração de imagens corrompidas por ruído. Neste caso, a imagem pode ser vista como uma grade de pixels, cada pixel é um objeto que deve ser classificado com uma cor. O custo de atribuição é dado pela similaridade entre a nova cor e a antiga, e o custo de separação é dado pela diferença entre a cor do pixel e a cor de seus vizinhos.

O Problema da Classificação Métrica, mesmo restrito a distâncias uniformes (ULP), generaliza o *Multiway Cut Problem*, um problema conhecido que é NP-difícil [14].

O MLP foi primeiramente introduzido por Kleinberg e Tardos [33]. Eles apresentaram um algoritmo $O(\log m \log \log m)$-aproximado. Para o ULP obtiveram um algoritmo 2-aproximado. Para ambos os problemas, a técnica utilizada foi arredondamento probabilístico sobre a solução de um programa linear. Esse programa linear tem muitas variáveis e restrições, $\Omega(m\,n^2)$ restrições e $\Omega(m\,n^2)$ variáveis, para ser usado em instâncias grandes.

Recentemente, Gupta e Tardos [24] apresentaram uma formulação para o problema da classificação truncada, um problema de classificação métrica onde as classes são in-

teiros positivos e a distância métrica entre as classes i e j é dada pela norma linear truncada, $d_{ij} = \min\{M, |i-j|\}$, onde M é o máximo valor permitido. Eles apresentaram um algoritmo 4-aproximado para a classificação métrica truncada e 2-aproximado para o ULP. O algoritmo gera uma instância para o problema da rede de fluxo onde os pesos das arestas vêm do custo de atribuição e separação do problema original. O grafo resultante tem $O(n^2m)$ arestas e um algoritmo para o corte mínimo é aplicado $O((m/M)(\log Q_0 + \log \varepsilon^{-1}))$ vezes para se obter uma $(4+\varepsilon)$-aproximação, onde Q_0 é o custo da solução inicial. Esse algoritmo também tem alta complexidade para ser usado em instâncias práticas.

Chekuri, Khanna, Naor e Zosin [11] desenvolveram uma nova formulação em programação linear que é melhor para o caso não uniforme. Porém, para o caso uniforme ela mantém a 2-aproximação e tem uma complexidade de tempo mais alta, que deriva do fato de resolver um programa linear grande, com $\Omega(n^2m^2)$ restrições e $\Omega(n^2m^2)$ variáveis.

Em nossa pesquisa, foi apresentado um algoritmo aproximado mais rápido para o Problema da Classificação Métrica Uniforme e foi provado que ele é uma $8 \log n$-aproximação com complexidade de tempo estimada em $O(mn^{36})$. Além disso, foi comparado o desempenho prático desse algoritmo com os algoritmos baseados em programação linear de Chekuri *et al.* [11] e Kleinberg e Tardos [33]. Apesar do novo algoritmo ter um maior fator de aproximação, os testes computacionais indicaram que as soluções obtidas tiveram desvio menor que 60% em relação a um limitante do ótimo em instâncias geradas aleatoriamente. Ademais, o algoritmo obteve soluções para instâncias grandes numa menor quantidade de tempo.

Também foi provado que a análise do fator de aproximação do algoritmo proposto é justa, a menos de um fator constante.

Elaboramos para este problema um artigo em conjunto com Evandro C. Bracht [6], o qual foi aceito no *Workshop on Experimental and Efficient Algorithms (WEA)* e cuja versão estendida foi publicada no *ACM Journal of Experimental Algorithmics (JEA)* [5]. O Capítulo 2 traz este artigo.

1.2.2 Problema do Corte Balanceado

No início deste capítulo descrevemos informalmente o Problema do Corte Balanceado. Este problema consiste em, dado um grafo $G(V, E)$, encontrar um conjunto não vazio $S \subset V$ de forma a minimizar a função objetivo. Seja $\delta(S)$ o conjunto de arestas com exatamente uma extremidade em S, seja $\mathcal{C}(S) = \sum_{e \in \delta(S)} c_e$ onde $c_e \in \mathbb{R}^+$ é um custo para cada aresta $e \in E$. A medida de distribuição do corte é $\mathcal{W}(S) = w(S) \cdot w(V \setminus S)$ onde $w(C) := \sum_{v \in C} w_v$ e $w_v \in \mathbb{R}^+$ é um peso para cada vértice $v \in V$. O objetivo deste problema consiste em encontrar um corte não vazio $S \subset V$ tal que $\frac{\mathcal{C}(S)}{\mathcal{W}(S)}$ seja mínimo. Denotamos

por n o número de vértices em V.

O primeiro algoritmo aproximado para o problema do Corte Balanceado consiste em uma $O(\log n)$-aproximação devido ao estudo de Leighton e Rao [36] publicado em 1988. Recentemente, o interesse no problema tem crescido. Para o caso em que todos os custos das arestas estão em $\{0, 1\}$, Arora, Rao e Vazirani [2] apresentaram uma $O(\sqrt{\log n})$-aproximação baseada em programação semidefinida. Para o caso geral, Chawla, Gupta e Räque [10] apresentaram uma $O(\log^{3/4} n)$-aproximação, enquanto o melhor algoritmo atualmente é uma $O(\sqrt{\log n} \log \log n)$-aproximação feita por Arora, Lee e Naor [1].

Devanur, Khot, Saket e Vishnoi [15] apresentaram uma instância com gap de integralidade de $\Theta(\log \log n)$ para a relaxação semidefinida considerada nesta tese. Shmoys [43] apresenta uma coletânea sobre algoritmos de aproximação para problemas de corte e suas aplicações para divisão e conquista.

Em nosso trabalho sobre este problema, analisamos uma relaxação conhecida descrita por Arora *et al.* [2]. Esta relaxação é baseada em programação semidefinida. Apresentamos um algoritmo exato e heurísticas baseadas nesta relaxação. A relaxação foi testada em instâncias de tamanho pequeno a moderado e devolveu valores muito próximos à solução ótima. O algoritmo exato obteve soluções para instâncias de tamanho pequeno a moderado e as melhores heurísticas obtiveram valores ótimos ou quase ótimos para todas as instâncias testadas. A relaxação semidefinida nos forneceu um limitante $\frac{C}{W}$ e cada heurística produziu um corte S com valor $\frac{\mathcal{C}(S)}{\mathcal{W}(S)}$. Provamos, para cada heurística, que $\mathcal{C}(S)$ é no máximo um fator de C ou $\mathcal{W}(S)$ é pelo menos um fator W.

Não encontramos algoritmos aproximados para o problema. Recentemente foi provado que aproximar o problema do Corte Balanceado dentro de qualquer fator constante, assumindo a Conjectura *Unique Games* de Khot [9], implica $P = NP$.

Na parte experimental, resolvemos a relaxação semidefinida a partir de um resolvedor próprio para o Problema do Corte Balanceado utilizando a técnica de inserção de cortes semidefinidos sobre programação linear. Para resolver a programação linear foi usado o resolvedor profissional XPRESS. Nós mostramos que a estratégia proposta teve um desempenho melhor que um resolvedor de programação semidefinida conhecido.

Como resultado secundário, provamos a existência de uma solução limitada quando a esperança de ambas as partes da fração de variáveis aleatórias são limitadas. É possível aplicar esta estratégia a outros problemas com fração na função objetivo. Lembrando que não é sempre verdade que $E[X/Y] = E[X]/E[Y]$, este resultado tem uma utilidade geral.

Elaboramos um artigo sobre este problema, o qual está submetido para uma revista da área. O Capítulo 3 traz este artigo.

1.2.3 Problemas k-Means e Localização de Recursos

Trabalhamos, por fim, em problemas de clusterização. Esta classe de problemas consiste em agrupar elementos em conjuntos de forma a minimizar a soma da distorção entre cada elemento e o centro do respectivo conjunto. Para tanto, consideramos uma versão contínua do Problema de Localização de Recursos sem Capacidades e sua aplicação para o Problema k-Means. Nestes dois problemas temos um conjunto de clientes $C \subset \mathbb{R}^q$. Procuramos encontrar um conjunto não vazio $F \subset \mathbb{R}^q$ para conectar cada cliente à facilidade mais próxima em F. No Problema de Localização de Recursos Contínuo, o valor de uma solução é o custo total de conexão dos clientes com a respectiva facilidade, mais o custo de abrir as facilidades, que é $f \cdot |F|$, onde f é o custo para se abrir uma facilidade. No Problema k-Means, o conjunto F deve ter exatamente k facilidades e o custo de uma solução é o custo total de conectar os clientes à respectiva facilidade. Em ambos os casos, procuramos soluções de custo mínimo.

O Problema de Localização de Recursos (Sem Capacidades) e Problemas de Clusterização têm muitas aplicações práticas [17] e têm sido largamente investigados na literatura. As aplicações aparecem em muitas áreas como pesquisa operacional, mineração de dados, recuperação da informação, processamento de imagens, compressão de dados, aplicações web, etc. Nosso principal interesse no estudo deste problema foi o desenvolvimento de algoritmos aproximados.

O primeiro algoritmo de aproximação para o Problema de Localização de Recursos é um algoritmo guloso com fator de aproximação $O(\log n)$, onde n é o número de facilidades e clientes na instância de entrada, descrito por Hochbaum [26] em 1982. Este resultado é o melhor possível, assumindo $P \neq NP$. Para o caso métrico, o primeiro fator de aproximação constante é 3,17, descrito por Shmoys, Tardos e Aardal [44]. Existem muitas outras técnicas e limitantes na literatura. Uma das técnicas principais consiste no uso de programação linear e no uso da abordagem primal-dual. Neste sentido, Jain e Vazirani [30] apresentaram um algoritmo primal-dual 3-aproximado para o Problema de Localização de Recursos. O melhor fator de aproximação é de 1,52 vindo de um algoritmo proposto por Mahdian, Ye e Zhang [39] que usa a estratégia primal-dual com LP revelador de fator e técnica de escala de custo. Considerando resultados negativos, Guha e Khuller [23] mostraram que qualquer algoritmo de aproximação deve ter um fator de aproximação de pelo menos 1,463, assumindo $NP \subsetneq DTIME[n^{O(\log \log n)}]$. O Problema de Localização de Recursos também é um bom exemplo do impacto prático da busca por melhores fatores de aproximação. Por exemplo, o algoritmo 1,61-aproximado apresentado por Jain, Mahdian, Markakis, Saberi e Vazirani [28] obteve soluções quase sem erro na maioria dos casos, indicando que fatores de aproximação são também uma boa medida para o comportamento prático dos algoritmos.

O problema de clusterização mais famoso é o Problema k-Means que é NP-difícil mesmo para $k = 2$ [16]. Um dos algoritmos mais clássicos para o Problema k-Means é o algoritmo de Lloyd [38], para o qual existem implementações rápidas [31]. Apesar deste algoritmo possuir um bom comportamento prático, ele não tem fator de aproximação ou convergência em tempo polinomial. De fato, existem exemplos para os quais o algoritmo obtém clusteres arbitrariamente ruins. Considerando algoritmos de aproximação, Matousek [40] apresentou um esquema de aproximação polinimial para o k-Means quando q, ϵ e k são constantes. Existem muitos outros esquemas de aproximação polinomiais sob estas condições. Mais recentemente, Kumar, Sabharwal e Sen [34] apresentaram um esquema de aproximação quando k e ϵ são constantes, independentemente de q. Quando ϵ e q são constantes, Kanungo, Netanyahu, Piatko, Silverman e Wu [32] apresentaram um algoritmo $(9+\epsilon)$-aproximado. Infelizmente, a maioria dos algoritmos acima mencionados são muito complicados ou tem alta complexidade para valores pequenos de ϵ. Em [30], Jain e Vazirani descreveram um algoritmo de aproximação primal-dual com fator 108 para o Problema k-Means (independente de k e q). Apesar deste algoritmo ter um alto fator, ele pode ser usado na prática por sua baixa complexidade de tempo.

Apresentamos resultados de aproximação para o caso contínuo do Problema de Localização de Recursos ao adaptar algoritmos apresentados em [28, 30]. Obtivemos um algoritmo $(1{,}861+\epsilon)$-aproximado para a métrica l_2 e um algoritmo $(9+\epsilon)$-aproximado com distâncias l_2^2. Para o problema l_2-Clustering obtivemos um algoritmo $(6+\epsilon)$-aproximado. Para o Problema k-Means, propusemos um algoritmo similar ao apresentado em [30] com fator de aproximação $54 + \epsilon$. Todos estes resultados são para q fixo. Nossa contribuição foi um método para extrair de um número exponencial de centróides um subgrupo polinomial que garantiu praticamente a mesma qualidade da solução. Nós reportamos resultados computacionais para o Problema k-Means, mostrando que para valores pequenos de q o algoritmo produz soluções próximas ou superiores a algoritmos conhecidos para o problema. Para as mesmas instâncias, o algoritmo em alguns casos devolveu soluções melhores que as obtidas pela combinação de Lloyd e outros algoritmos de busca local [31]. Por fim, implementamos algumas variantes dos algoritmos e fizemos uma análise empírica.

Elaboramos para este problema um artigo que foi aceito no *23rd Annual ACM Symposium on Applied Computing (SAC 2008)*. O Capítulo 4 traz uma versão estendida deste artigo.

Chapter 2

A Greedy Approximation Algorithm for the Uniform Metric Labeling Problem Analyzed By a Primal-Dual Technique

Abstract

We consider the uniform metric labeling problem. This NP-hard problem considers how to assign objects to labels respecting assignment and separation costs. The known approximation algorithms are based on solutions of large linear programs and are impractical for moderated and large size instances. We present an $8 \log n$-approximation algorithm that can be applied to large size instances. The algorithm is greedy and is analyzed by a primal-dual technique. We implemented the presented algorithm and two known approximation algorithms and compared them at randomized instances. The gain of time was considerable with small error ratios. We also show that the analysis is tight, up to a constant factor.

2.1 Introduction

In a traditional classification problem, we wish to assign each of n objects to one of m labels (or classes). This assignment must be consistent with some observed data that includes pairwise relationships among the objects to be classified. More precisely, the classification problem can be defined as follows. Let P be a set of objects, L a set of labels, $w : P \times P \to \mathbb{R}^+$ a symmetric weight function, $d : L \times L \to \mathbb{R}^+$ a symmetric distance function and $c : P \times L \to \mathbb{R}^+$ an assignment cost function. A *labeling* of P over

L is a function $\phi : P \to L$. The *assignment cost* of a labeling ϕ is the sum $\sum_{i \in P} c(i, \phi(i))$ and the *separation cost* of a labeling ϕ is the sum $\sum_{i,j \in P} d(\phi(i), \phi(j))w(i, j)$. The function w indicates the strength of the relation between two objects, and the function d indicates the similarity between two labels. The cost of a labeling ϕ is the sum of the assignment cost and the separation cost. The *Metric Labeling Problem (MLP)* consists on finding a labeling of the objects into labels with minimum total cost. Throughout this paper, we denote the size of the sets P and L by n and m, respectively.

We focus our attention on the *Uniform Metric Labeling Problem (UMLP)* where the distance $d(i, j)$ is 1 if $i \neq j$, and 0 otherwise, for all $i, j \in L$.

The MLP has several applications, many listed by Kleinberg and Tardos [33]. Some applications occur in image processing [4, 13, 18], biometrics [3], text categorization [8], etc. An example of application is the restoration of images degenerated by noise. In this case, an image can be seen as a grid of pixels, each pixel is an object that must be classified with a color. The assignment cost is given by the similarity between the new and old coloring, and the separation cost given by the color of a pixel and the color of its neighbors.

The Metric Labeling Problem generalizes the Multiway Cut Problem, a known NP-hard problem [14], and was first introduced by Kleinberg and Tardos [33] that present an $O(\log m \log \log m)$-approximation algorithm for the MLP, and a 2-approximation algorithm for the UMLP using the randomized rounding technique over a solution of a linear program. This linear program is too large to be used in large size instances. It has $\Theta(m\,n^2)$ constraints and $\Theta(m\,n^2)$ variables.

Because the UMLP generalizes the Multiway Cut Problem, it is also NP-hard.

More recently, Gupta and Tardos [24] present a formulation for the truncated labeling problem, a metric labeling problem where the labels are positive integers and the metric distance between labels i and j is given by the truncated linear norm, $d_{ij} = \min\{M, |i - j|\}$, where M is the maximum value allowed. They present an algorithm that is a 4-approximation algorithm for the truncated labeling problem and a 2-approximation for the UMLP. This algorithm generates a network flow problem instance where the weights of edges come from the assignment and separation costs of the original problem. The resulting graph has $O(n^2m)$ edges and the *Min Cut* algorithm is applied $O((m/M)(\log Q_0 + \log \varepsilon^{-1}))$ times in order to obtain a $(4 + \varepsilon)$-approximation, where Q_0 is the cost of the initial solution. This is also a high time complexity to be used in practical instances.

Chekuri, Khanna, Naor and Zosin [11] developed a new linear programming formulation that is better for the non-uniform case. However, for the uniform case it maintains a 2-approximation factor and has a higher time complexity. This is a consequence of solving a large linear program with $\Theta(n^2m^2)$ constraints and $\Theta(n^2m^2)$ variables.

In this paper, we present a fast approximation algorithm for the Uniform Metric Labeling Problem and prove that it is an $8 \log n$-approximation algorithm. We also compare the practical performance of this algorithm with the linear programming based algorithms of Chekuri et al. [11] and Kleinberg and Tardos [33]. Although this algorithm has a higher approximation factor, the computational tests indicated that the obtained solutions have values that are within 60% of the optimum. Moreover, the algorithm could obtain solutions for large instances in small amount of time.

In Section 2.2, we present an approximation factor proof for a greedy algorithm for the Set Cover Problem via a primal-dual analysis. Then we analyze the case when the greedy choice is relaxed to an approximated one. In Section 2.3, we present a general algorithm for the Uniform Metric Labeling Problem using an algorithm for the Set Cover Problem. This algorithm uses as a subroutine an approximation algorithm for the Quotient Cut Problem. In Subsection 2.3.3 we show that the analysis is tight, up to a constant factor. In Section 2.4, we compare the presented algorithm with linear programming based algorithms. In Section 2.5, we present the concluding remarks. Finally, in Section 2.6, we describe some future works, and, in Section 2.7, we present the acknowledgements.

2.2 A Primal-Dual Analysis for a Greedy Algorithm for the Set Cover Problem

The Set Cover Problem is a well-known optimization problem, that generalizes many others. An instance of this problem consists of: a set $E_{\text{SC}} = \{e_1, e_2, \ldots, e_n\}$ of elements, a family of subsets $\mathcal{U}_{\text{SC}} = \{U_1, U_2, \ldots, U_m\}$, where $U_j \subseteq E_{\text{SC}}$, and a cost w_j, for each $j \in \{1, \ldots, m\}$. A set $S \subseteq \{1, \ldots, m\}$ such that $\cup_{j \in S} U_j = E_{\text{SC}}$ is a set cover of E_{SC}. The goal of the problem is to find a set cover $Sol \subseteq \{1, \ldots, m\}$ of E_{SC} that minimizes $\sum_{j \in Sol} w_j$.

In [12], Chvátal presents a greedy H_g-approximation algorithm for the Set Cover Problem, where g is the number of elements in the largest set in $\mathcal{U}_{\text{SC}}$ and H_g is the value of the harmonic function of degree g. We denote this algorithm by Greedy. This algorithm iteratively chooses the set with minimum amortized cost, that is the cost of the set divided by the number of non-covered elements in the set. Once a set is chosen to be in the solution, all the elements in this set are considered as covered. For more details on algorithm Greedy, see Chapter 3 of book [45].

The Greedy algorithm can be rewritten as a primal-dual algorithm, with similar set of events. To present this algorithm, we first consider a formulation for the Set Cover Problem using binary variables x_j for each set U_j, where $x_j = 1$ if and only if U_j is chosen

ALGORITHM PD-SC($E_{\mathrm{SC}}, \mathcal{U}_{\mathrm{SC}}, w$)
Input: Set of elements E_{SC}, family of sets $\mathcal{U}_{\mathrm{SC}}$, each set $U_j \in \mathcal{U}_{\mathrm{SC}}$ with weight w_j.
Output: A set cover $Sol \subseteq \{1, \ldots, m\}$ of E_{SC}.

1. $U \leftarrow E_{\mathrm{SC}}$; $T \leftarrow 0$; $Sol \leftarrow \emptyset$.
2. Let $\alpha_e \leftarrow 0$, for all $e \in E_{\mathrm{SC}}$.
3. While $U \neq \emptyset$ do
4. increase T and α_e for each $e \in U$, uniformly, until we have
$$\sum_{e \in U_j \cap U} \alpha_e = w_j, \text{ for some } j.$$
5. $Sol \leftarrow Sol \cup \{j\}$.
6. $U \leftarrow U \setminus U_j$.
7. Return *Sol*.

Figure 2.1: Primal-Dual Algorithm for the Set Cover Problem.

in the solution. The formulation consists on finding x that

$$\begin{aligned} \text{minimize} \quad & \sum_j w_j x_j & & & \\ \text{s.t.} \quad & \sum_{j: e \in U_j} x_j & \geq & 1 & \forall e \in E_{\mathrm{SC}}, \\ & x_j & \in & \{0,1\} & \forall j \in \{1, \ldots, m\}, \end{aligned}$$

and the dual of the relaxed version consists on finding α that

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E_{\mathrm{SC}}} \alpha_e & & & \\ \text{s.t.} \quad & \sum_{e \in U_j} \alpha_e & \leq & w_j & \forall j \in \{1, \ldots, m\}, \\ & \alpha_e & \geq & 0 & \forall e \in E_{\mathrm{SC}}. \end{aligned}$$

The primal-dual algorithm, which we denote by PD-SC, uses a set U containing the elements not covered in each iteration, and a variable T. The algorithm do not need T, but it is used in the analysis as a notion of time associated with each event. At each iteration, the algorithm increases the variable T and the (dual) variables α_e uniformly, for each element in U, until the inequality correspondent to a set U_j becomes tight. At this point, the algorithm chooses j and declares each element of U_j as covered. Algorithm PD-SC is presented in Figure 2.1.

We first present a proof of the approximation factor of algorithm PD-SC using a factor revealing primal-dual analysis. The idea of the proof is proposed in Jain, Mahdian, Markakis, Saberi and Vazirani [28], Section 9, and described in [45]. For completeness,

we included the proof here, adapting the presentation in such a way that we can easily extend Theorem 3 to Theorem 5.

Lemma 1 *The sequence of events executed by algorithms* Greedy *and* PD-SC *is the same.*

Proof. Note that the value of α_e when $\sum_{e \in U_j \cap U} \alpha_e = w_j$ is equal to the amortized cost. Because α_e grows uniformly, it is clear that algorithm PD-SC, at each iteration, chooses a set with minimum amortized cost. □

Lemma 2 *Let $U_j = \{e_1, e_2, \ldots, e_k\}$ and α_i be the dual variable associated to e_i, generated by algorithm* PD-SC. *If $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_k$, then $\sum_{i=l}^{k} \alpha_l \leq w_j$, for any l.*

Proof. At the moment just before $T = \alpha_l$, all α_i, $l \leq i \leq k$, have the same value, α_l, and $e_l, \ldots, e_k$ are all uncovered. Suppose the lemma is false. In this case, $\sum_{i=l}^{k} \alpha_l > w_j$ and, in an instant strictly before α_l, the set U_j would enter in the solution and all of its elements would have $\alpha < \alpha_l$. This is a contradiction, because U_j has at least one element greater than or equal to α_l. Therefore, the lemma is valid. □

Algorithm PD-SC returns a primal solution *Sol*, with value $val(Sol) := \sum_{j \in Sol} w_j$, such that $val(Sol) = \sum_{e \in E_{\mathrm{SC}}} \alpha_e$. Note that α may not be dual feasible. If there exists a value γ such that α/γ is dual feasible, i.e., $\sum_{e \in U_j} \alpha_e/\gamma \leq w_j$ for each $j \in \{1, \ldots, m\}$, then, by the weak duality theorem, $val(Sol) \leq \gamma OPT$. The idea of the proof is to find a value γ for which α/γ is dual feasible.

Theorem 3 *The* PD-SC *Algorithm, for the Set Cover Problem, is an H_g-approximation algorithm, where g is the size of a largest set in $\mathcal{U}_{\mathrm{SC}}$.*

Proof. Consider an arbitrary set $U_j = \{e_1, \ldots, e_k\}$ with k elements and cost w. Let α_i be the dual variable associated with element e_i for $i = 1, \ldots, k$. Without loss of generality, assume that $\alpha_1 \leq \ldots \leq \alpha_k$.

If γ is a value such that α/γ is dual feasible, then

$$\sum_{i=1}^{k} \alpha_i/\gamma \leq w. \tag{2.1}$$

Thus, a sufficient condition is

$$\gamma \geq \frac{\sum_{i=1}^{k} \alpha_i}{w}. \tag{2.2}$$

Applying Lemma 2 for each value of l, we have

$$\begin{cases} l=1, & k\alpha_1 \leq w, & \Rightarrow & \alpha_1/w \leq 1/k, \\ l=2, & (k-1)\,\alpha_2 \leq w, & \Rightarrow & \alpha_2/w \leq 1/(k-1), \\ & & \vdots & \\ l=k, & \alpha_k \leq w, & \Rightarrow & \alpha_k/w \leq 1. \end{cases}$$

Summing up the inequalities above we have

$$\frac{\sum_{i=1}^{k}\alpha_i}{w} \leq H_k. \tag{2.3}$$

Therefore, when $\gamma = H_g$, we obtain that α/γ is dual feasible. □

Now, let us consider a small modification in the previous algorithm. Instead of choosing a set with minimum amortized cost, we choose a set $U_{j'}$ with amortized cost at most f times greater than the minimum. That is, if U_{j*} is a set with minimum amortized cost in a given iteration, then the following inequality is valid

$$\frac{w_{j'}}{|U_{j'} \cap U|} \leq f\frac{w_{j*}}{|U_{j*} \cap U|}.$$

This modification can be understood, in the primal-dual version of the algorithm, as a permission that $\sum_{e\in U_j\cap U}\alpha_e$ becomes greater than w_j by a factor of at most f. We denote the algorithm with this modification by $\mathcal{A}_f$.

Lemma 4 *Let $U_j = \{e_1, e_2, \ldots, e_k\}$ and α_i be the time variable associated to the item e_i generated by algorithm $\mathcal{A}_f$. If $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_k$, then $\sum_{i=l}^{k}\alpha_l \leq f\,w_j$, for any l.*

Proof. Suppose the lemma is false. Then, for some l, $\sum_{i=l}^{k}\alpha_l > f\,w_j$ and, in an instant $T < \alpha_l$, the set U_j would have entered in the solution, that is a contradiction. □

The following theorem can be proved analogously to Theorem 3.

Theorem 5 *Algorithm $\mathcal{A}_f$ is an $f\,H_g$-approximation, where g is the size of a largest set in $\mathcal{U}_{\mathrm{SC}}$.*

2.3 A New Algorithm for the Uniform Metric Labeling Problem

The algorithm for the UMLP uses some ideas presented by Jain et al. [28] for the Facility Location Problem. To present these ideas, we use the notion of a star. A *star* is a

connected graph where only one vertex, denoted as *center* of the star, can have degree greater than one. Jain et al. [28] present an algorithm that iteratively selects a star with minimum amortized cost, where the center of each star is a facility.

In the Uniform Metric Labeling Problem, we can consider a labeling $\phi : P \to L$ as a set of stars, each one with a label in the center. Thus a star for UMLP is a pair (l, U_S), where $l \in L$ and $U_S \subseteq P$. The algorithm for the UMLP iteratively selects stars of reduced amortized cost, until all objects have been covered.

Given a star $S = (l, U_S)$ for the UMLP, where $l \in L$ and $U_S \subseteq P$, we denote by $\mathcal{C}(S)$ the cost of the star S, which is defined as

$$\mathcal{C}(S) = c(U_S) + \frac{1}{2}w(U_S),$$

where $c(U_S) = \sum_{u \in U_S} c_{ul}$ and $w(U_S) = \sum_{u \in U_S, v \in (P \setminus U_S)} w_{uv}$. That is, $\mathcal{C}(S)$ is the cost to assign each element of U_S to l plus half the cost to separate each element of U_S from each element of $P \setminus U_S$. We pay just half of the separation cost, because the other half will appear when we analyze the elements in $P \setminus U_S$ against the elements in U_S.

Denote by $\mathcal{S}_{\text{UMLP}}(L, P)$ the set of all possible stars of an instance. The algorithm uses U as the set of unassigned objects, $(E_{\text{SC}}, \mathcal{S}_{\text{SC}}, w')$ as an instance for the Set Cover Problem, $\mathcal{U}$ as a collection and ϕ the labeling being produced. Without loss of generality, we consider $\mathcal{S}_{\text{SC}}$ as a collection and given a set $U_S \in \mathcal{S}_{\text{SC}}$, we can obtain the associated star $S = (l, U_S)$, and *vice versa*. We describe the algorithm more formally in Figure 2.2.

ALGORITHM GUL$(L, P, c, w, \mathcal{A}_{\text{SC}})$
Input: Set L of labels; set P of objects; assignment costs c_{ui} for $i \in L$ and $u \in P$;
separation costs w_{uv} for $u, v \in P$; and an algorithm $\mathcal{A}_{\text{SC}}$ for Set Cover.
Output: A labeling ϕ of P.

1. $E_{\text{SC}} \leftarrow P$.
2. $\mathcal{S}_{\text{SC}} \leftarrow \{U_S : S = (l, U_S) \in \mathcal{S}_{\text{UMLP}}(L, P)\}$.
3. $w'_S \leftarrow \mathcal{C}(S) + \frac{1}{2}w(U_S)$, for all $S = (l, U_S) \in \mathcal{S}_{\text{UMLP}}(L, P)$.
4. $\mathcal{U} \leftarrow \mathcal{A}_{\text{SC}}(E_{\text{SC}}, \mathcal{S}_{\text{SC}}, w')$, let $\mathcal{U} = \{U_{S_1}, U_{S_2}, \ldots, U_{S_t}\}$.
5. $U \leftarrow P$.
6. For $k \leftarrow 1$ to t do
7. Let $l \in L$ be such that $S_k = (l, U_{S_k})$;
8. $\phi(i) \leftarrow l$, for all $i \in U_{S_k} \cap U$;
9. $U \leftarrow U \setminus U_{S_k}$.
10. Return ϕ.

Figure 2.2: Greedy algorithm for the Uniform Metric Labeling Problem.

2.3.1 Analysis of the Algorithm

To analyze the algorithm we use the following notation:

- $val_{\mathrm{UMLP}}(\phi)$: the value, in the UMLP, of a labeling ϕ.
- $val_{\mathrm{SC}}(\mathcal{U})$: the value, in the Set Cover Problem, of a collection $\mathcal{U}$.
- ϕ_{OPT}: an optimum labeling for UMLP.
- $\mathcal{U}_{\mathrm{OPT}}$: an optimum solution for the Set Cover Problem.
- $SC(\phi)$: a collection $\{U_{S_1}, U_{S_2}, \ldots, U_{S_k}\}$ related to a labeling $\phi = \{S_1, S_2, \ldots, S_k\}$.

Lemma 6 *If ϕ is the solution returned by algorithm* GUL *and $\mathcal{U}$ the solution returned by algorithm $\mathcal{A}_{\mathrm{SC}}$ at Step 4 of algorithm* GUL*, then*

$$val_{\mathrm{UMLP}}(\phi) \leq val_{\mathrm{SC}}(\mathcal{U}).$$

Proof.

$$\begin{aligned} val_{\mathrm{UMLP}}(\phi) &= \sum_{S\in\phi} \mathcal{C}(S) = \sum_{S\in\phi} \left(c(U_S) + \frac{1}{2} w(U_S) \right) && (2.4)\\ &\leq \sum_{U_S\in\mathcal{U}} (c(U_S) + w(U_S)) && (2.5)\\ &= \sum_{U_S\in\mathcal{U}} \left(\mathcal{C}(S) + \frac{1}{2} w(U_S) \right) \\ &= \sum_{U_S\in\mathcal{U}} w'_S = val_{\mathrm{SC}}(\mathcal{U}). \end{aligned}$$

Equality (2.4) is valid because the stars in ϕ are disjoint and this can be checked by counting. Inequality (2.5) is valid because an assignment cost in $val_{\mathrm{UMLP}}(\phi)$ also appears in $val_{\mathrm{SC}}(\mathcal{U})$,

$$\sum_{S\in\phi} c(U_S) \leq \sum_{U_S\in\mathcal{U}} c(U_S),$$

and the separation cost w_{uv} for any u and v such that $\phi(u) \neq \phi(v)$, that appears in $val_{\mathrm{UMLP}}(\phi)$ must appear, once or twice, in $val_{\mathrm{SC}}(\mathcal{U})$,

$$\sum_{S\in\phi} \frac{1}{2} w(U_S) = \sum_{u<v:\phi(u)\neq\phi(v)} w_{uv} \leq \sum_{U_S\in\mathcal{U}} w(U_S).$$

□

Lemma 7 $val_{\mathrm{SC}}(\mathcal{U}_{\mathrm{OPT}}) \leq 2val_{\mathrm{UMLP}}(\phi_{\mathrm{OPT}})$.

Proof.

$$\begin{aligned} 2val_{\mathrm{UMLP}}(\phi_{\mathrm{OPT}}) &= 2\sum_{S\in\phi_{\mathrm{OPT}}} \mathcal{C}(S) \geq \sum_{S\in\phi_{\mathrm{OPT}}} \left(\mathcal{C}(S) + \frac{1}{2}w(U_S)\right) && (2.6)\\ &= \sum_{t\in SC(\phi_{\mathrm{OPT}})} w'_t \\ &\geq \sum_{t\in\mathcal{U}_{\mathrm{OPT}}} w'_t && (2.7)\\ &= val_{\mathrm{SC}}(\mathcal{U}_{\mathrm{OPT}}), \end{aligned}$$

where inequality (2.6) is valid because $\mathcal{C}(S) \geq \frac{1}{2}w(U_S)$ and inequality (2.7) is valid because $SC(\phi_{\mathrm{OPT}})$ is a solution for the Set Cover Problem, but not necessarily with optimum value. □

Theorem 8 *If I is an instance for the UMLP, ϕ is the labeling generated by algorithm* Gul *when running on I and ϕ_{OPT} is an optimum labeling for I then*

$$val_{\mathrm{UMLP}}(\phi) \leq 2\beta\, val_{\mathrm{UMLP}}(\phi_{\mathrm{OPT}}),$$

where β is the approximation factor of algorithm $\mathcal{A}_{\mathrm{SC}}$ used at Step 4 of algorithm Gul.

Proof. Let $\mathcal{U}$ be the solution returned by algorithm $\mathcal{A}_{\mathrm{SC}}$ (Step 4 of algorithm Gul) and $\mathcal{U}_{\mathrm{OPT}}$ an optimal solution for the corresponding Set Cover instance. In this case, we have

$$\begin{aligned} val_{\mathrm{UMLP}}(\phi) &\leq val_{\mathrm{SC}}(\mathcal{U}) && (2.8)\\ &\leq \beta\, val_{\mathrm{SC}}(\mathcal{U}_{\mathrm{OPT}}) && (2.9)\\ &\leq 2\beta\, val_{\mathrm{UMLP}}(\phi_{\mathrm{OPT}}), && (2.10) \end{aligned}$$

where inequality (2.8) is valid by Lemma 6, inequality (2.9) is valid because $\mathcal{U}$ is found by a β-approximation algorithm, and inequality (2.10) is valid by Lemma 7. □

The algorithm PD-SC cannot be directly applied in Step 4 of algorithm Gul because it is polynomial in the size $\mathcal{S}_{\mathrm{SC}}$ but exponencial in the size of P. To obtain an $8\log n$-approximation algorithm for the UMLP we need to present an algorithm $\mathcal{A}_{\mathrm{SC}}$ that is a $4\log n$-approximation algorithm for Set Cover. The algorithm is based on an approximation algorithm for the Quotient Cut Problem, which we describe in the following subsection.

2.3.2 Algorithm $\mathcal{A}_{\mathrm{SC}}$

In this section we show how to obtain a greedy algorithm for Set Cover without the explicit generation of all possible sets. The algorithm basically generates a graph G_l for each possible label $l \in L$ and obtains a set U_S with reduced amortized cost. The set which must enter in the solution is the smallest one considering the sets U_S from G_l for each $l \in L$.

Consider a label $l \in L$. We wish to find a set U_S that minimize $w_S/|U_S \cap U|$, where U is the set of unassigned objects in the iteration. Denote by G_l the complete graph with vertex set $V(G_l) = P \cup \{l\}$ and edge costs w_{G_l} defined as follows

$$\begin{aligned} w_{G_l}(u,v) &:= w_{uv} \qquad \forall u, v \in P, u \neq v, \\ w_{G_l}(l,u) &:= c_{ul} \qquad \forall u \in P. \end{aligned}$$

In other words, the cost of an edge between the label and an object is the assignment cost and the cost of an edge between two objects is the separation cost.

Lemma 9 *For each set $C \subseteq P$, the cost $c(C) = \sum_{u \in C, v \notin C} w_{G_l}(u,v)$ has value w'_S, where w'_S is the cost attributed in Step 3 of algorithm* Gul *to the set indexed by star $S = (l, C)$.*

Proof. The lemma can be proved by counting. In the Set Cover Problem, w'_S is equal to

$$\mathcal{C}(S) + \frac{1}{2} w(U_S),$$

that is equal to the cost of the cut C. See Figure 2.3. □

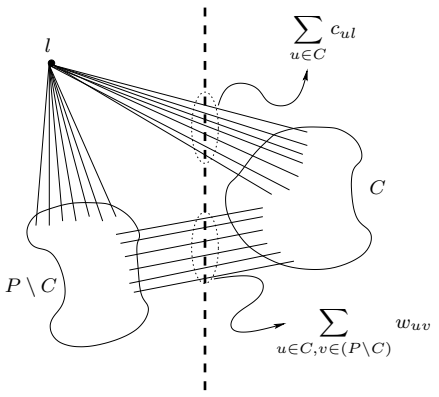


Figure 2.3: Example of a cut C that has the same cost w'_S that the star $S = (l, C)$.

The problem of finding a set with smallest amortized cost $c(C)/|C|$ in G_l is the Minimum Quotient Cut Problem, which can be defined as follows.

QUOTIENT CUT PROBLEM (QC): Given a graph G, weights $w_e \in \mathbb{R}^+$ for each edge $e \in E(G)$, and costs $c_v \in \mathbb{Z}^+$ for each vertex $v \in V(G)$, find a cut C that minimizes $w(C)/\min\{\pi(C), \pi(\bar{C})\}$, where $w(C) := \sum_{u \in C, v \notin C} w_{uv}$ and $\pi(C) := \sum_{v \in C} c_v$.

If we define a cost function c_l in each vertex of G_l as

$$c_l(v) = \begin{cases} 1 & \text{if } v \in P \cap U, \\ 0 & \text{if } v \in P \setminus U, \\ n & \text{if } v = l, \end{cases}$$

where U contains the elements not covered in a given iteration, then a cut $C := \min_{l \in L}\{QC(G_l, w_l, c_l)\}$, returned by some f-approximation algorithm for the QC Problem, corresponds to a set U_S, $S = (l, C)$, that is at most f times greater than a set with minimum amortized cost. Algorithm $\mathcal{A}_f$, as stated in Theorem 5, can be implemented choosing this set instead of a set with minimum amortized cost. The following result follows from Theorem 5 and Theorem 8.

Theorem 10 *If there exists an f-approximation algorithm for the Quotient Cut Problem with time complexity $T(n)$, then there exists a $2f\,H_n$-approximation algorithm for the UMLP, with time complexity $O(n\,m\,T(n))$.*

The Quotient Cut Problem is NP-hard and the best approximation algorithm has an approximation factor of 4 due to Freivalds [20]. This algorithm has experimental time complexity $O(n^{1.6})$ when the degree of each vertex is a small constant and $O(n^{2.6})$ for dense graphs. Although this algorithm has polynomial time complexity estimated experimentally, it is not proved to run in polynomial time in the worst case.

Theorem 11 *There is an $8 \log n$-approximation Fm for UMLP, with time complexity estimated in $O(m\,n^{3.6})$.*

It is important to observe that the size of an input I for UMLP, $size(I)$, is $O(n(n+m))$, thus, the estimated complexity of our algorithm is $O(size(I)^{2.3})$.

2.3.3 The Approximation Factor is $\Theta(\log n)$

In this subsection we will show that the approximation factor is tight, up to a constant factor. We present a family of instances where the factor given by the algorithm is H_n. We suppose that $\mathcal{A}_{\text{SC}}$ is an algorithm which selects a set with minimum amortized cost at each iteration.

Given a positive integer n and positive values ϵ_1 and ϵ_2, where $0 < \epsilon_1 \ll \epsilon_2 \ll 1/n^2$, define an instance $\mathcal{I}_n = (L, P, c, w)$ for UMLP as follows

- $L = \{1, \ldots, n\}$,
- $P = \{1, \ldots, n\}$,
- for each $i \in P$ and $l \in L$, define c_{il} as follows

$$c_{il} = \begin{cases} \epsilon_1 & \text{if } i \in \{1, \ldots, n-1\} \text{ and } l = i, \\ \epsilon_2 & \text{if } i \in \{1, \ldots, n-1\} \text{ and } l = n, \\ 1 & \text{if } i = n \text{ and } l = n, \\ \infty & \text{otherwise,} \end{cases}$$

- for each $i, j \in P$, define w_{ij} as follows

$$w_{ij} = \begin{cases} \dfrac{1}{n-i+1} & \text{if } i \in \{1, \ldots, n-1\} \text{ and } j = n, \\ 0 & \text{otherwise.} \end{cases}$$

Instance $\mathcal{I}_n$ is presented in Figure 2.4. In the next theorem we show that algorithm Gul can obtain a solution of $\mathcal{I}_n$ with a factor of $\Theta(\log n)$ of the optimum.

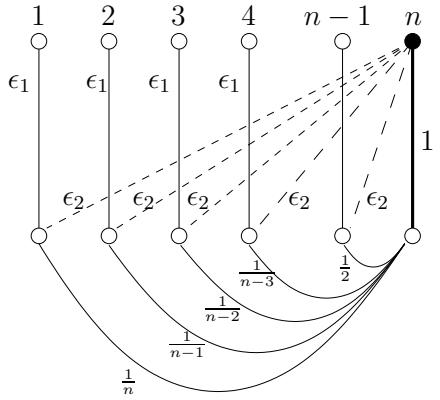


Figure 2.4: Instance $\mathcal{I}_n$.

Theorem 12 *If algorithm* Gul *uses a greedy algorithm as subroutine for the Set Cover Problem, then* $\frac{\text{Gul}(\mathcal{I}_n)}{\text{OPT}(\mathcal{I}_n)} \to H_n$ *as* $\epsilon_1, \epsilon_2 \to 0$*, where* $\text{OPT}(\mathcal{I}_n)$ *is the value of an optimum solution of* $\mathcal{I}_n$ *for UMLP.*

Proof. The optimal solution for instance $\mathcal{I}_n$ is the star (n, P) with cost $1 + \epsilon_2(n-1)$. Now, we will prove that the solution obtained by algorithm Gul can be formed by the stars $(1, \{1\}), (2, \{2\}), \ldots, (n, \{n\})$ whose cost is $H_n + \epsilon_1(n-1)$.

The theorem can be proved by induction on the number of iterations.

Given a star $S = (l, U_S)$, for $l \in L$ and $U_S \subseteq P$, let $W_i(l, U_S)$ denote the amortized cost of S at iteration i. That is, $W_i(l, U_S) = \frac{w'_S}{|U_S \cap U|}$, where U is the set of unassigned objects at iteration i.

Although the number of stars may be large, we can consider few stars, as shown by the next fact.

Fact 13 *All stars with bounded cost are of the form*

- $(i, \{i\})$ *for* $i = 1, \ldots, n$;
- (n, U_S) *for any set* $U_S \subseteq P,\ U_S \neq \emptyset$.

Consider the first iteration. In this case we will prove that star $(1, \{1\})$ has the smallest amortized cost. The following inequalities are immediate:

$$W_1(1, \{1\}) < W_1(2, \{2\}) < \ldots < W_1(n, \{n\}). \tag{2.11}$$

$$W_1(i, \{i\}) < W_1(n, \{i\}) \text{ for all } i \in \{1, \ldots, n-1\}. \tag{2.12}$$

From inequalities (2.11) and (2.12) we can conclude that $W_1(1, \{1\})$ is smaller than or equal to the amortized cost of all stars containing one object.

Because $W_1(1, \{1\}) = 1/n + \epsilon_1$ and $W_1(n, P) = 1/n + \frac{n-1}{n}\epsilon_2$, we have

$$W_1(1, \{1\}) < W_1(n, P). \tag{2.13}$$

Consider stars with objects Q such that $n \in Q \subseteq P$. Because adding an object $i \in P \setminus Q$ to Q decreases the separation cost and it does not incur a significant increase in the assignment cost (just ϵ_2 is added), we have

$$W_1(n, P) \leq W_1(n, Q). \tag{2.14}$$

From inequalities (2.13) and (2.14) we can conclude that any star that contains the object n has cost greater than $W_1(1, \{1\})$.

Now, consider stars with object set $Q \subseteq P \setminus \{n\}, Q \neq \emptyset$. A minimality argument says that

$$W_1(n, \{i\}) \leq W_1(n, Q), \text{ for } i = \min\{Q \cap U\}. \tag{2.15}$$

Inequality (2.15) is valid, because

$$W_1(n, \{i\}) = \min_{i' \in Q \cap U} \{\frac{1}{n - i' + 1}\} + \epsilon_2$$

and

$$W_1(n, Q) \geq \text{average}_{i' \in Q \cap U}\{\frac{1}{n - i' + 1}\} + \epsilon_2.$$

From the previous inequalities we can conclude that all stars have an amortized cost greater than $W_1(1, \{1\})$. Thus star $(1, \{1\})$ enters in the solution obtained by algorithm GUL in the first iteration.

Updating U, the set of unassigned objects, we note an invariant property. In the second iteration it is clear that

$$W_2(2, \{2\}) < W_2(3, \{3\}) < \ldots < W_2(n, \{n\}) \tag{2.16}$$

and because $W_2(2, \{2\}) = 1/(n-1) + \epsilon_1$ and $W_2(n, P) = 1/(n-1) + \frac{n-1}{n-1}\epsilon_2$ we have

$$W_2(2, \{2\}) < W_2(n, P). \tag{2.17}$$

In a similar way, we can conclude that all stars have an amortized cost greater than $W_2(2, \{2\})$ in the second iteration. Thus the greedy algorithm will select the star $(2, \{2\})$ to enter in the solution.

The induction step is straightforward. Therefore, the solution produced by algorithm GUL consists of $\{\phi(1) = 1, \phi(2) = 2, \ldots, \phi(n) = n\}$. □

2.4 Computational Experience

We performed several tests with the presented algorithm, denoted by GUL, the algorithm presented by Kleinberg and Tardos [33], denoted by KT, and the algorithm presented by Chekuri et al. [11], denoted by $CKNZ$. In the implementation of algorithm GUL, we do not consider intersections among sets in the solution of the subroutine $\mathcal{A}_{\text{SC}}$ in the Step 4 of the algorithm. The factor 2 that appears in Lemma 7 was diminished in the experiments when the assigned objects are removed from the current graph in the Quotient Cut subroutine of GUL.

We observe that the computational resources needed to solve an instance with the presented algorithm are very small compared with the other two algorithms cited above. All the implemented algorithms and the instances are available in the web [7]. The tests were performed in an Athlon XP with 1.2 Ghz and 700 MB of RAM and the linear programs were solved by the Xpress-MP solver [42].

We considered five different sets of instances. We denote these sets by *A, B, C, D* and *E*.

The instances of set A have the following characteristics: The number of objects is given by $n = \lceil \frac{2k}{3} \rceil$ and the number of labels by $m = \lfloor \frac{k}{3} \rfloor$ for different values of $k = n + m$. The assignment costs are given by $c_{ui} = \mathit{random}(1000)$ for all $u \in P$ and $i \in L$, and

separation costs by $w_{uv} = random(10)$ for all $u, v \in P$, where $random(i)$ returns a random value between 0 and i.

The largest instance we could solve with algorithm $CKNZ$ has 46 objects and 24 labels in 16722 seconds. The time is basically that of solving the linear program. When the primal-dual algorithm is tested with this instance, we obtained a solution that is 0.25% worse in 7 seconds. Concerning algorithm KT, the largest instance that we could test has 80 objects and 40 labels and it took 15560 seconds. This is basically the time to solve the associated linear program. For this instance, algorithm GUL obtained a solution that is 1.01% worse in 73 seconds. The major limitation to use KT was the time complexity, while the major limitation to use algorithm $CKNZ$ was the time and memory complexities. It is important to note that algorithm GUL does not use a linear programming solver.

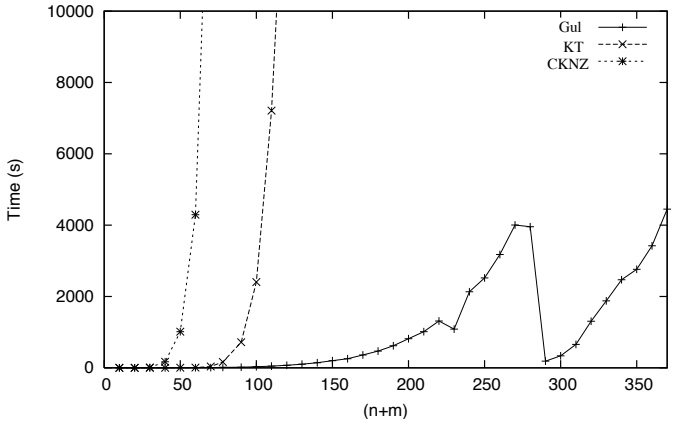


Figure 2.5: Running times for instances of set A.

In Figure 2.5 we can observe a strong reduction in the function time associated with the GUL algorithm when $n + m$ achieves 290. In this case, we observed that when the separation costs are big, there is a high probability that the size of the star becomes big. To larger instances of type A, the number of objects assigned to a label in each iteration is also big and the running time is reduced. The average number of objects assigned per iteration just before the time reduction was 1.2, that is near to the worst case, while for the instance just after the decrease, the average number of objects assigned per iteration was 32.3.

For all comparable instances of set A, algorithm GUL produced solutions that are at most 1.27% worse than the value of the linear programming relaxation.

The instances of set B were generated as follows. We maintain the same ratio between the number of classes and objects. The assignment costs are given by $c_{ui} = random(5n)$

and the separation costs by $w_{uv} = random(5)$ for all $u, v \in P$ and $i \in L$, where $n = |P|$.

In these instances, the expected ratio between assignment costs and separation costs keeps constant while varying $n+m$. Thus the size of the star that enters in the solution is almost the same and the time complexity changes smoothly with the size of the instances.

The largest instance that we could solve with algorithm $CKNZ$ has 46 objects and 24 labels, and spent 18720 seconds. Memory was the main restriction to solve large instances with this algorithm. When we tested algorithm GUL with this instance, we obtained a solution that is 1.25% worse in 7 seconds. The largest instance solved by KT had 120 objects and 65 labels in 44400 seconds. For this instance, algorithm GUL produced a solution that is 0.48% worse in 68 seconds. The worst ratio of a solution produced by GUL, when compared with the KT relaxation, was 11.53%. Considering all instances of this set, the number of objects per star was 1.03 on the average. The running times for this set of instances can be compared in Figure 2.6.

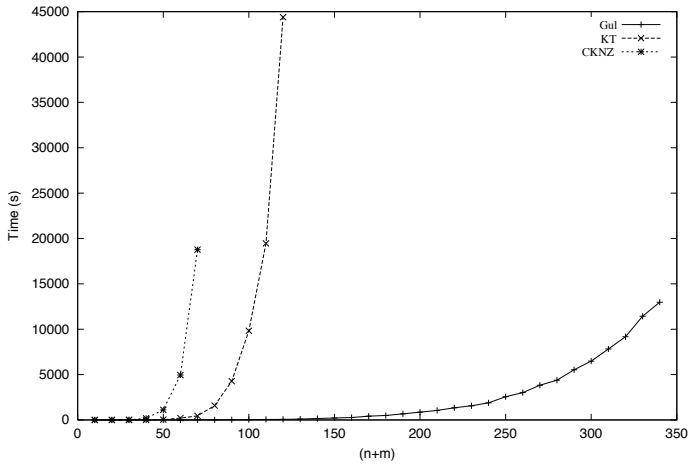


Figure 2.6: Running times for instances of set B.

To generate the instances of set C, we fix the number of labels (objects) to 40 and vary the number of objects (labels) from 5 to 65. The assignment and separation costs were generated in the same way as done for the instances of set B. Because algorithm $CKNZ$ spent more time, we did not use this algorithm for the remaining tests.

In Table 2.1 we show the results for the instances of set C. The column *Ratio*(%) corresponds to the value $(\text{GUL}(I)/KT(I) - 1) \cdot 100$, for each instance I. The largest error ratio was 1.25%. When the number of labels is fixed and the number of objects increases, the gain in time of algorithm GUL, compared to algorithm KT, was considerable.

On the average, algorithm GUL associated 1.05 objects in each iteration. When the number of objects, n, increases, algorithm KT takes substantially more time, compared

Instance		Time (s)		
Objects (n)	Labels (m)	GUL	KT	Ratio (%)
40	5	0	1	0.18
40	10	2	3	0.23
40	15	3	6	0.67
40	20	4	9	0.29
40	25	5	12	0.47
40	30	5	23	0.19
40	35	6	30	0.66
40	40	7	36	0.27
40	45	8	49	**1.25**
40	50	9	56	0.78
40	55	10	75	0.77
40	60	11	94	0.44
40	65	11	68	0.37
5	40	0	1	0
10	40	1	0	0
15	40	0	1	0
20	40	1	2	0.87
25	40	2	3	0.83
30	40	3	7	0.46
35	40	4	17	0.70
45	40	11	77	0.62
50	40	16	131	0.56
55	40	22	400	0.38
60	40	26	912	0.47
65	40	33	2198	0.60

Table 2.1: Results of KT and GUL over the instances of set C

to GUL. This confirms the time complexity of algorithm KT (for solving a linear program with $O(n^2m)$ constraints by $O(n^2m)$ variables) and algorithm GUL, experimentally estimated in $O(m\,n^{3.6})$.

For all instances in set D, the number of objects is $n = 40$ and the number of labels $m = 20$. The assignment costs are given by $c_{ui} = \mathit{random}(1000)$, and the separation costs are given by $w_{uv} = \mathit{random}(10\rho)$, where ρ is an integer varying from 1 to 100. The comparison of algorithms GUL and KT is presented in Table 2.2.

Because the separation costs increase as ρ increases, larger stars become promising. When big stars enter in the solution, the number of iterations decreases. Notice that the hardest instance occurs in the transition between big stars and small stars. When the separation costs are big ($\rho > 80$), the problem becomes easy, because it consists in con-

Time (s)		Ratio (%)	ρ	Number of iterations
GUL	KT			
3	5	0.34	1	36
4	7	0.39	5	37
4	7	0.57	10	38
4	11	0.75	15	39
4	11	0.43	20	38
3	25	0.66	25	36
3	21	0.47	30	37
4	23	0.51	35	38
3	35	0.51	40	37
4	41	0.88	45	36
4	67	0.68	50	36
4	114	1.06	55	36
4	168	1.75	60	36
4	203	1.26	65	32
4	439	2.26	70	32
4	831	4.32	75	29
4	1182	13.33	80	23
2	790	0	85	1
0	549	0	90	1
0	262	0	55	1
0	114	0	100	1

Table 2.2: Result of KT and GUL over the instances of set D.

necting all objects to one label. When $\rho = 80$, the corresponding instance was "difficult" for algorithm GUL, because the approximation factor increased, and was also "difficult" for algorithm KT, because its running time also increased. We can conclude, from this experiment, that the ratio between assignment and separation costs is an important characteristic in hard instances.

We observe that less than 1% of the instances of this set led to linear programs with fractional solutions, both for algorithm KT and $CKNZ$. The larger ratio between the value of the solution produced by algorithm KT and its associated linear program was 24.17% using 1020 seconds. In this case, algorithm GUL produced a solution within a ratio of 7.15% in 4 seconds. The average error for the instances where the linear programming solution was not integral was 7.02% and the average running time was 836 seconds to KT. The average error ratio of algorithm GUL when applied to the same instances was 5.8% in 3.02 seconds, on the average.

For all instances in the sets $A, \ldots, D$, the associated graph is complete. That is, each object has a separation cost with all other objects and each label has an assignment cost

with each object.

Based on the instances of set D, we generate a set E of instances associated with sparse graphs, where the degree of each vertex, in P, is 2, on the average. More precisely, these instances have 20 objects and 10 labels, and 40 edges on the average. The assignment costs are given by $c_{ui} = random(20)$ and the separation cost by $w_{uv} = random(8)$. On the average, 8% of these instances produced linear programs which lead to fractional solutions. The average (largest) error ratio of a solution produced by algorithm KT, from the fractional solution, was 6.47% (21.34%). The largest error ratio of a solution produced by algorithm GUL was 50.89% and the average error ratio was 13.46%.

The computational results we have obtained with these instances gave us more knowledge to generate hard instances. This led us to the instance $\mathcal{I}_n$ which proves that GUL analysis is tight up to a constant factor (see Section 2.3.3).

To illustrate the applicability of the primal-dual algorithm in practical instances, we have applied the algorithm for the image restoration problem with an image degenerated by noise. The image has pixels in gray scale and dimension 60x60 with 3600 objects to be classified in two colors, black and white. To define the assignment and separation costs, we consider that each color is an integer between 0 and 255. Each pixel corresponds to an object. The assignment cost of an object u to a label i is given by $c_{ui} = |clr(u) - clr(i)|$, where $clr(u)$ is the actual color of u and $clr(i)$ is the color associated to label i. The separation cost of an object u and an object v is given by $w_{uv} = 255 - |clr(u) - clr(v)|$.

The following images, Figures 2.7–2.10, present the results obtained applying the primal-dual algorithm. In each figure, the image on the left is obtained inserting some noise and the image on the right is the image obtained after the classification of the objects.

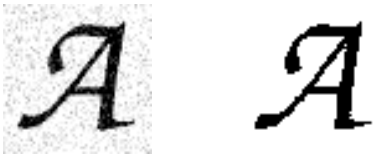


Figure 2.7: Image with 25% of noise. Time needed is 288s.

Figure 2.8: Image with 50% of noise. Time needed is 319s.

Figure 2.9: Image with 75% of noise. Time needed is 511s.

Figure 2.10: Image with 100% of noise. Time need is 973s.

We note that images with more noise have more pixels with different colors (labels). This explains the higher running time for these instances.

Although these times are large for image processing applications, it illustrates the performance of algorithm GUL and solution quality. In real instances, it is possible to define a small window over larger images, and the processing time may decrease. As one can see, the classification problem is more general and the primal-dual algorithm is appropriate for moderate and large size instances.

2.5 Concluding Remarks

We presented a primal-dual approximation algorithm with approximation factor $8 \log n$ for the Uniform Metric Labeling Problem. We compared the primal-dual algorithm with linear programming based approximation algorithms. We proved that our analysis is tight, up to a constant factor. The previous approximation algorithms for this problem have high time complexity and are adequate only for small and moderate size instances. In all experimental tests, the presented algorithm obtained solutions within 60% of the optimum and could obtain solutions for moderate and large size instances.

2.6 Future Works

A natural continuity to this work would be the use of GUL algorithm in real instances, as well the use of machinery learning and others tools to create good costs to this instances.

As theoretical future work, the method used in this chapter could be applied to others problems.

2.7 Acknowledgemets

We would like to thank K. Freivalds for making available his code for the Quotient Cut Problem and Cristina G. Fernandes for helpful suggestions to this article.

Chapter 3

Semidefinite Programming Based Algorithms for the Sparsest Cut Problem

Abstract

In this paper we analyze a known relaxation for the Sparsest Cut Problem based on positive semidefinite constraints, and present a branch and bound algorithm and heuristics based on this relaxation. The relaxation and the algorithms were tested on small and moderate sized instances. The relaxation leads to values very close to the optimum solution values. The exact algorithm could obtain solutions for small and moderate sized instances, and the best heuristics obtained optimum or almost optimum solutions for all tested instances. The semidefinite relaxation gives a lower bound $\frac{C}{W}$ and each heuristic produces a cut S with a ratio $\frac{c_S}{w_S}$, where either c_S is at most a factor of C or w_S is at least a factor of W. We solve the semidefinite relaxation using a semi-infinite cut generation with a commercial linear programming package adapted to the sparsest cut problem. We show that the proposed strategy leads to a better performance compared to the use of a known semidefinite programming solver.

3.1 Introduction

Since the work of Goemans and Williamson [22], which presents an approximation algorithm for the Max-Cut Problem, the use of semidefinite programming has increased and it turns out to be an important technique to obtain good relaxations and algorithms for combinatorial optimization problems [22, 25, 50].

The Sparsest Cut Problem is NP-hard and has applications in image segmentation [46], the metric labeling problem [5], and a natural application in graph conductance.

In this paper we analyze the relaxation of the Sparsest Cut Problem formulation presented by Arora, Rao and Vazirani [2]. We propose four heuristics and an exact branch and bound algorithm for the Sparsest Cut Problem, based on semidefinite programming relaxation.

These algorithms were tested on a set of random instances and, in all tests, they produced optimal or almost optimal solutions.

Although we could not present approximation factors to the heuristics, we proved good characteristics for each one. We note that it is NP-hard to approximate the Sparsest Cut problem within any constant factor, assuming the Unique Games Conjecture of Khot [9]. We proved the existence of a bounded solution for the Sparsest Cut Problem when the expectation of both parts of the ratio are bounded. It is possible to apply this strategy to other problems with a ratio in the objective function. Considering that it is not always true that $E[X/Y] = E[X]/E[Y]$, this result has a general utility. See more details in Section 3.4.1.

The first approximation algorithm for the Sparsest Cut Problem has an $O(\log n)$ factor due to Leighton and Rao [36] in 1988. Recently, the interest in the Sparsest Cut Problem has increased. When all edge costs are in $\{0, 1\}$ and vertex weights are 1, Arora et al. [2] present an $O(\sqrt{\log n})$-approximation algorithm based on a semidefinite program. For the general case, Chawla, Gupta and Räque [10] present an $O(\log^{3/4} n)$-approximation algorithm, while the current best known algorithm is an $O(\sqrt{\log n} \log \log n)$-approximation algorithm due to Arora, Lee and Naor [1].

Devanur, Khot, Saket and Vishnoi [15] present a $\Theta(\log \log n)$ integrality gap instance for the considered semidefinite programming relaxation. Shmoys [43] presents a survey on approximation algorithms for cut problems and their application to divide-and-conquer.

3.2 Problem Definition

Given a graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, a weight function $w : V \to \mathbb{R}^+$, and a set $S \subset V$, $S \neq \emptyset$, we denote by $\delta(S)$ the set of edges with exactly one extremity in S, $\mathcal{C}(S)$ the sum $\sum_{e \in \delta(S)} c_e$, $\mathcal{W}(S)$ the product $w(S)w(V \setminus S)$, where $w(C) := \sum_{v \in C} w_v$, and $\rho(S)$ the value $\mathcal{C}(S)/\mathcal{W}(S)$. The Sparsest Cut Problem can be defined as follows:

> Sparsest Cut Problem (SC): Given a graph $G = (V, E)$, costs $c_e \in \mathbb{R}^+$ for each edge $e \in E$, and weights $w_v \in \mathbb{R}^+$ for each vertex $v \in V$, find a cut $S \subset V, S \neq \emptyset$, that minimizes $\rho(S)$. See Figure 3.1.

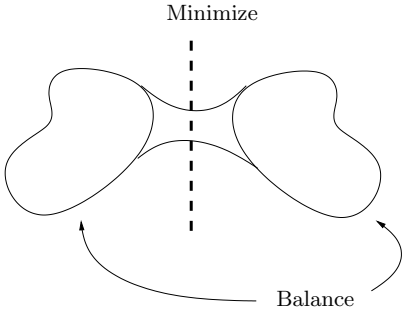


Figure 3.1: The objective of the Sparsest Cut Problem is to find a small balanced cut.

In the unweighted version, which we denote by USC Problem, we have $c_e = 1$ for each $e \in E$, and $w_v = 1$ for each v in V.

Another problem that is strongly related to the Sparsest Cut Problem is the Quotient Cut Problem, which can be defined as follows:

> Min Quotient Cut Problem: Given a graph $G = (V, E)$, costs $c_e \in \mathbb{R}^+$ for each edge $e \in E$, and weights $w_v \in \mathbb{R}^+$ for each vertex $v \in V$, find a cut $S \subset V, S \neq \emptyset$, that minimizes $\mathcal{C}(S)/\min\{w(S), w(V \setminus S)\}$.

In [2], the Min Quotient Cut is also referred to as Sparsest Cut. In terms of approximability, an α-approximation algorithm for the Sparsest Cut (Quotient Cut) Problem is a 2α-approximation algorithm for the Quotient Cut (Sparsest Cut) Problem.

The following lemma can be deduced from the work of Leighton and Rao [36]. Because the proof is not straightforward, we present it here.

Lemma 3.2.1 *An α-approximation algorithm for the Sparsest Cut (Quotient Cut) Problem is a 2α-approximation algorithm for the Quotient Cut (Sparsest Cut) Problem.*

Proof. Let $\mathcal{A}_1$ be an α-approximation algorithm and I an instance for the Sparsest Cut Problem. Let S be the cut produced by the algorithm $\mathcal{A}_1(I)$. Without loss of generality, assume that $w(S) \leq w(V)/2$ and $w(V \setminus S) \geq w(V)/2$. Because $\mathcal{A}_1$ is an α-approximation algorithm, we have

$$\frac{\mathcal{C}(S)}{w(S)w(V \setminus S)} \leq \alpha \frac{\mathcal{C}(S')}{w(S')w(V \setminus S')}, \quad \forall S' \subset V, S' \neq \emptyset.$$

That is,

$$\frac{\mathcal{C}(S)}{w(S)} \leq \alpha \frac{\mathcal{C}(S')w(V \setminus S)}{w(S')w(V \setminus S')}, \quad \forall S' \subset V, S' \neq \emptyset$$

$$\begin{aligned} &\leq \alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}} \frac{w(V)}{\max\{w(S'), w(V \setminus S')\}}, \quad \forall S' \subset V, S' \neq \emptyset \\ &\leq 2\alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}}, \quad \forall S' \subset V, S' \neq \emptyset, \end{aligned} \tag{3.1}$$

where inequality (3.1) is valid because $w(V) \leq 2\max\{w(S'), w(V \setminus S')\}$.

Now, suppose that $\mathcal{A}_2$ is an α-approximation algorithm and I an instance for the Quotient Cut Problem. Let S be the cut produced by $\mathcal{A}_2(I)$. Without loss of generality, we suppose that $w(S) \leq w(V)/2$ and $w(V \setminus S) \geq w(V)/2$. Thus

$$\begin{aligned} \frac{\mathcal{C}(S)}{w(S)} &= \frac{\mathcal{C}(S)}{\min\{w(S), w(V \setminus S)\}} \\ &\leq \alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}}, \quad \forall S' \subset V, S' \neq \emptyset. \end{aligned} \tag{3.2}$$

That is,

$$\begin{aligned} \frac{\mathcal{C}(S)}{w(S)w(V \setminus S)} &\leq \alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}\ w(V \setminus S)}, \quad \forall S' \subset V, S' \neq \emptyset \\ &= \alpha \frac{\mathcal{C}(S')}{w(S')w(V \setminus S')} \frac{\max\{w(S'), w(V \setminus S')\}}{w(V \setminus S)}, \quad \forall S' \subset V, S' \neq \emptyset \quad (3.3) \\ &\leq 2\alpha \frac{\mathcal{C}(S')}{w(S')w(V \setminus S')}, \quad \forall S' \subset V, S' \neq \emptyset. \quad (3.4) \end{aligned}$$

Equality (3.3) is obtained multiplying the previous fraction by $\frac{\max\{w(S'), w(V \setminus S')\}}{\max\{w(S'), w(V \setminus S')\}}$. Inequality (3.4) is valid because $\max\{w(S'), w(V \setminus S')\} \leq w(V) \leq 2w(V \setminus S)$. □

3.3 Semidefinite Programming Formulation and Relaxation

Given an instance (G, c, w) for the Sparsest Cut Problem, there is a variable v_i for each $i \in V_G$. This variables will value either r or $-r$, indicating if the vertex is in one side or the other of the cut.

For each vertex $i \in V_G$, the fact that v_i has a value r or $-r$ allows us to scale v_i in a way that the denominator of the objective function ratio becomes 1 without changing the objective function. Note that $|v_i - v_j|^2$ is non zero if and only if edge (i, j) is considered in the ratio.

$$
\begin{aligned}
&\rho_F = \min \sum_{i<j,(i,j)\in E} c_{ij}|v_i - v_j|^2 \\
&\text{s.t.} \\
&\qquad |v_i - v_j|^2 \;\leq\; |v_i - v_k|^2 + |v_k - v_j|^2, \quad \forall i,j,k \in V, \\
&\sum_{i<j} w_i w_j |v_i - v_j|^2 \;=\; 1, \qquad\qquad (F) \\
&\qquad\qquad v_i^2 \;=\; r^2, \quad \forall i \in V, \\
&\qquad\qquad r \;\geq\; 0, \\
&\qquad\qquad v_i, r \;\in\; \mathbb{R}, \quad \forall i \in V.
\end{aligned}
$$

Lemma 3.3.1 *F is a* formulation *for the Sparsest Cut Problem.*

Proof. Let I be an instance for the Sparsest Cut Problem, $\mathcal{O}^* \subset V$ be an optimum solution to I and ρ_F be the value of the objective function of F solved with instance I. Now, we prove that a solution for one problem leads to a solution for the other, with the same value.

Given a cut S, let $r = \frac{1}{\sqrt{4|S||V\setminus S|}}$, $v_i = r$ for each $i \in S$ and $v_i = -r$ for each $i \in V \setminus S$. The obtained attribution is feasible to F and has cost $\rho(S)$ for unweighted graphs. This is valid for any set S, including $\mathcal{O}^*$. In the weighted version, the same conclusion is valid for $r = \frac{1}{\sqrt{4\mathcal{W}(S)\mathcal{W}(V\setminus S)}}$.

To conclude that F is a formulation it is sufficient to show that any solution has an associated cut with the same cost. If $\mathcal{V} = (v, r)$ is a solution to F, we may obtain an associated cut $S_\mathcal{V} = \{i : v_i = -r\}$. As $\mathcal{V}$ is feasible, $r = \frac{1}{\sqrt{4|S_\mathcal{V}||V\setminus S_\mathcal{V}|}}$ (in the weighted version $r = \frac{1}{\sqrt{4\mathcal{W}(S_\mathcal{V})\mathcal{W}(V\setminus S_\mathcal{V})}}$) and $\rho_F(\mathcal{V})$ values $\rho(S_\mathcal{V})$. □

For the rest of this chapter, we consider $n = |V|$. If we relax v_i to an n-dimensional vector in $\mathbb{R}^n$, we have a relaxation, which we denote by $\vec{R}$. To write the relaxation $\vec{R}$ as a semidefinite program, we observe that $|v_i - v_j|^2 = (v_i - v_j)^2 = v_i^2 - 2v_iv_j + v_j^2$ and $x_{ij} = v_iv_j$. For further information about semidefinite programming see [25, 27, 35]. In the following we present the (weighted) relaxation $\vec{R}$ and its corresponding semidefinite program R, where v_0 is the n-dimensional vector associated with an arbitrary vertex $v_0 \in V_G$ and $X \succeq 0$ means that X is positive semidefinite.

$$\vec{R}$$

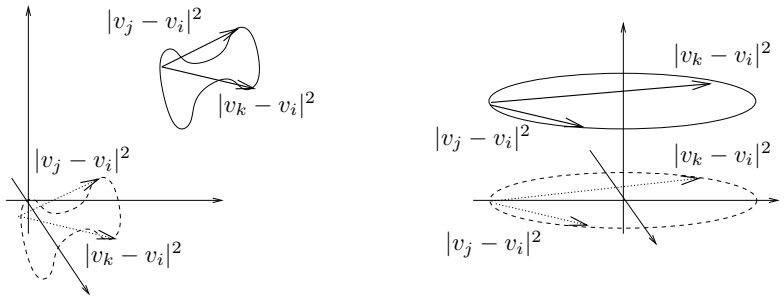
$$\begin{aligned}
\text{minimize } & \sum_{i<j} c_{ij}|v_i - v_j|^2 \\
\text{such that } & \\
& |v_i - v_j|^2 \leq |v_i - v_k|^2 + |v_k - v_j|^2 \quad \forall i,j,k \in V, \\
& \sum_{i<j} w_i w_j |v_i - v_j|^2 = 1, \\
& |v_i| = |v_0| \quad \forall i \in V, \\
& v_i \in \mathbb{R}^n.
\end{aligned}$$

$$R$$

$$\begin{aligned}
\text{minimize } & \sum_{i<j} c_{ij}(x_{ii} + x_{jj} - 2x_{ij}) \\
\text{such that } & \\
& x_{ik} + x_{kj} - x_{ij} \leq x_{kk} \quad \forall i,j,k \in V, \\
& \sum_{i<j} w_i w_j (x_{ii} + x_{jj} - 2x_{ij}) = 1, \\
& x_{ii} = x_{00} \quad \forall i \in V, \\
& X \succeq 0.
\end{aligned}$$

We denote by $\vec{R}_-$ and R_- the corresponding relaxations $\vec{R}$ and R without constraints $|v_i| = |v_0|$ and $x_{ii} = x_{00}$. For unweighted graphs, Arora et al. [2] prove an integrality gap of $O(\sqrt{\log n})$ for the relaxation $\vec{R}_-$.

Remark: Note that $\vec{R}_-$ has only relative distances between vectors in the objective function and in all constraints. This implies that the objective function value does not change when the vectors v_j, for $j \in V$, get the same translation. On the other hand, in the relaxation R, there may exist translations that do not change the objective function. See Figure 3.2.



Translations in R_- Translations in R

Figure 3.2: Translations in relaxations $\vec{R}_-$ and $\vec{R}$.

Definition 1 *An optimal solution X^* of R_- or R is said to be* small *if it is optimal and minimizes $\sum_{i \in V} |v_i|$.*

To convert a non-small solution of R_- to a small one with the same objective value, it is sufficient to translate its barycenter to the origin. In the relaxation R it is sufficient to translate the hyperplane of small dimension that contains all vectors of the solution to the origin.

Given an instance I for the Sparsest Cut Problem, we denote by $R_-(I)$ and $R(I)$ the relaxations R_- and R defined with the corresponding values of I.

3.4 Heuristics for the Sparsest Cut Problem

In this section we propose some heuristics for the Sparsest Cut Problem. These heuristics receive as an input parameter a feasible small solution $X^* = (x^*_{ij})$ to relaxation $R_-(G, c, w)$ or $R(G, c, w)$ and then apply some rounding strategy to obtain a feasible solution.

In the following we present the heuristics *H1*, *H2*, *H3* and *H4*. The heuristics *H1* and *H2* use a value $\xi(X^*, i, j)$ defined as $\xi(X^*, i, j) = \frac{x^*_{ij}}{\sqrt{x^*_{ii}x^*_{jj}}} = \cos(v_i, v_j)$. All the heuristics may generate an empty cut. Note that a smaller value for $\xi(X^*, i, j)$ means that vectors v_i and v_j are more separated and choosing a pair of vertices (s, t) such that $\xi(X^*, s, t)$ is minimum minimize the probability of empty cuts.

HEURISTIC *H1*$(G = (V, E), c, w, X^*)$

1. Let $s, t \in V$ where $\xi(X^*, s, t)$ is minimum.
2. $S \leftarrow \emptyset$
3. For each $u \in V$ do
4. select a random number α uniformly distributed in $[-1, 1]$
5. if $\xi(X^*, s, u) \geq \alpha$ then
6. $S \leftarrow S \cup \{u\}$.
7. Return(S).

HEURISTIC *H2*$(G = (V, E), c, w, X^*)$

1. Let $s, t \in V$ where $\xi(X^*, s, t)$ is minimum.
2. $S \leftarrow \emptyset$
3. Select a random number α uniformly distributed in $[-1, 1]$.
4. For each $u \in V$ do
5. if $\xi(X^*, s, u) \geq \alpha$
6. $S \leftarrow S \cup \{u\}$.
7. Return(S).

HEURISTIC *H3*$(G = (V, E), c, w, X^*)$

1. Let $\mathcal{V} = (v_i)$ be the Cholesky decomposition of X^* ($X^* = \mathcal{V}^T\mathcal{V}$).
2. Let $S \leftarrow \emptyset$
3. Select a random vector $r \in \mathbb{R}^n$ uniformly distributed in the unit sphere.
4. For each $u \in V$ do
5. if $v_u \cdot r \geq 0$
6. $S \leftarrow S \cup \{u\}$.
7. Return(S).

HEURISTIC $H4(G=(V,E),c,w,X^*)$

1. Let $\mathcal{V}=(v_i)$ be the Cholesky decomposition of X^* ($X^*=\mathcal{V}^T\mathcal{V}$).
2. Let $A \leftarrow \emptyset$ and $B \leftarrow \emptyset$.
3. Select two vectors $r_1, r_2 \in \mathbb{R}^n$ uniformly distributed in the unit sphere.
4. For each $u \in V$ do
5. if $v_u \cdot r_1 >= 0$ and $v_u \cdot r_2 >= 0$ then $A \leftarrow A \cup \{u\}$
6. if $v_u \cdot r_1 < 0$ and $v_u \cdot r_2 < 0$ then $B \leftarrow B \cup \{u\}$.
7. Let G' be a graph obtained from G contracting A and B to vertices a and b, resp.
8. Let S' be a cut with minimum cost separating a and b in G'.
9. Let S be the set of vertices in G corresponding to S'.
10. Return(S).

3.4.1 Some Properties of the Heuristics

Let $\mathcal{P}$ be a minimization problem with domain $\mathcal{D}$ and objective function $f(S)=\frac{g(S)}{h(S)}$, where $g(S)>0$ and $h(S)>0$ for each $S \in \mathcal{D}$.

Lemma 3.4.1 *Let $\mathcal{A}$ be a randomized algorithm for $\mathcal{P}$. Given an instance I of $\mathcal{P}$ and a solution S produced by $\mathcal{A}(I)$, denote by G_S the random variable $g(S)$ and H_S the random variable $h(S)$ and let $\mathcal{E}=\frac{E[G_S]}{E[H_S]}$. Then, there exists a solution S' with $f(S') \leq \mathcal{E}$.*

Proof. We have

$$\frac{E[G_S]}{E[H_S]} = \mathcal{E} \Rightarrow E[G_S] - \mathcal{E}E[H_S] = 0. \tag{3.5}$$

By the linearity of the expectation, we have

$$E[G_S - \mathcal{E}H_S] = 0.$$

This proves the existence of at least one solution S^{gap} such that

$$g(S^{gap}) - \mathcal{E}h(S^{gap}) \leq 0.$$

That is, $f(S^{gap}) \leq \mathcal{E}$. □

Fact 3.4.2 *Given a randomized algorithm $\mathcal{A}$ to problem USC, let C_S and W_S be the random variables $\mathcal{C}(S)$ and $\mathcal{W}(S)$, where S is the cut produced by $\mathcal{A}$ over instance I. Let S^* be an optimal cut of I, $f^*=\rho(S^*)$ and $\beta \geq 1$ be such that $\frac{E[C_S]}{E[W_S]}=\beta f^*$. If Z is the random variable $C_S - \beta f^* W_S$ then $\Pr[Z \geq f^*\frac{n}{b}] \leq p$ for any $p \in (0,1)$ where $n=|V|$ and $b=\frac{p}{(\beta-1)(1-p)n}$.*

Proof. The expectation $E[Z]$ is equal to $E[C_S] - f^*\beta E[W_S]$. Substituting β we obtain that $E[Z] = 0$. We also have that $1 \cdot (n-1) \leq W_S \leq n^2$.

Since f^* is a lower bound for the value of any solution, we have $\frac{C_S}{W_S} \geq f^*$ and therefore,

$$\begin{aligned} Z &= C_S - \beta f^* W_S \\ &\geq -(\beta - 1) f^* W_S \\ &\geq -(\beta - 1) f^* n^2. \end{aligned}$$

For the sake of contradiction, suppose that $\Pr[Z \geq f^* \frac{n}{b}] > p$. In this case, we obtain that

$$\begin{aligned} E[Z] &\geq -(\beta - 1) f^* n^2 \Pr[Z < f^* \tfrac{n}{b}] + f^* \tfrac{n}{b} \Pr[Z \geq f^* \tfrac{n}{b}] \\ &> -(\beta - 1) f^* n^2 (1 - p) + f^* \tfrac{n}{b} p \\ &= 0. \end{aligned}$$

This contradiction finishes the proof. □

Lemma 3.4.3 *Given a polynomial time randomized algorithm $\mathcal{A}$ to problem USC, let C_S and W_S be the random variables $\mathcal{C}(S)$ and $\mathcal{W}(S)$, where S is the cut produced by $\mathcal{A}$ over instance I. Let S^* be an optimal cut of I and β be such that $\frac{E[C_S]}{E[W_S]} = \beta\rho(S^*)$. Then it is also possible to obtain a cut S' with $\rho(S') \leq (1+\epsilon)\beta\rho(S^*)$ in polynomial time with high probability, for any $\epsilon > 0$.*

Proof. Let $f^* = \rho(S^*)$. Consider a pair (p, b) that respects Fact 3.4.2 in such a way that $b = 2/\epsilon$ for any $\epsilon > 0$. With probability greater than $1 - p$, we have that

$$Z = C_S - \beta f^* W_S \leq f^* \frac{n}{b}.$$

Because $1 \cdot (n-1) \leq W_S$ and $n \geq 2$ we have $\frac{n}{b} \leq 2\frac{W_S}{b}$. That is, with probability greater than $1 - p$, we have

$$\mathcal{C}(S) - \beta f^* \mathcal{W}(S) \leq f^* \frac{n}{b} \leq 2\frac{f^* \mathcal{W}(S)}{b}.$$

Thus, with probability greater than $1 - p$,

$$\begin{aligned} \frac{C_S}{W_S} &\leq \left(\beta + \frac{2}{b}\right) f^* = (\beta + \epsilon) f^* \\ &\leq (1+\epsilon)\beta\rho(S^*), \end{aligned} \tag{3.6}$$

where inequality (3.6) is valid because $\beta \geq 1$.

As $b = 2/\epsilon$, we have $p = \dfrac{1}{1 + \frac{\epsilon}{2(\beta-1)}{n}}$. If we get the solution S' with minimum value from n executions, the probability that inequality (3.6) is not satisfied for S' is less than

or equal to $p^n \leq \mathbf{e}^{-\frac{\epsilon}{2(\beta-1)}}$, where $\mathbf{e}$ is the base of the natural logarithm. Therefore, if β and ϵ are constant, the probability to obtain the desired cut is greater than a positive constant. We let to the interested reader to produce an algorithm that finds such a cut with high probability. □

Although we do not have the equality $E[A/B] = E[A]/E[B]$ in general, lemmas 3.4.1 and 3.4.3 describe a way to contour this difficulty for the Unweighted Sparsest Cut.

For the remaining of this section, we denote by $X^* = (x^*_{ij})$ a small optimum solution for $R(G, c, w)$ and by s and t two vertices where $\xi(X^*, s, t)$ is minimum. Given a solution X^* for relaxation R, we denote by Y^* the solution X^* scaled by a constant factor k such that $Y^* = kX^*$ and $Y^*_{ii} = 1$ for each vertex i.

Given a feasible solution X for R and $\mathcal{V} = (v)$ its corresponding solution for $\vec{R}$, denote by

$$\mathcal{C}(X) = \frac{1}{4}\sum_{i<j} c_{ij}(x_{ii} + x_{jj} - 2x_{ij}) = \frac{1}{4}\sum_{i<j} c_{ij}|v_i - v_j|^2,$$

$$\mathcal{W}(X) = \frac{1}{4}\sum_{i<j} w_i w_j(x_{ii} + x_{jj} - 2x_{ij}) = \frac{1}{4}\sum_{i<j} w_i w_j|v_i - v_j|^2,$$

and $\rho(X) = \mathcal{C}(X)/\mathcal{W}(X)$.

Observe that $\mathcal{C}(Y^*) = \sum_{i<j} c_{ij}\frac{1-y^*_{ij}}{2}$, $\mathcal{W}(Y^*) = \sum_{i<j} w_i w_j \frac{1-y^*_{ij}}{2}$ and $y^*_{ij} = \cos(v_i, v_j) = y^*_{ji}$.

Lemma 3.4.4 *The value $\rho(Y^*)$ is a lower bound for $\rho(S)$, for any set $S \subset V$, $S \neq \emptyset$.*

Proof. Let X^* be an optimum solution to R. As R is a relaxation of formulation F, $\rho(X^*)$ is a lower bound to the value of any optimum solution for the Sparsest Cut Problem. We have $\mathcal{C}(Y^*) = k\mathcal{C}(X^*)$, and $\mathcal{W}(Y^*) = k\mathcal{W}(X^*)$. Therefore, $\rho(Y^*) = \rho(X^*)$. □

We first consider that $\xi(X^*, s, t) = -1$. In this case, all heuristics generate non-empty cuts. We further present some ways to keep the same results when this condition is not true.

If each one of the four heuristics receives X^* as an input parameter, then they partially respect Lemma 3.4.1, but we could not obtain an approximation algorithm. In what follows, we show the strength and the weakness of each heuristic.

First consider heuristics *H1* and *H2*. These heuristics are, in some way, symmetric. While the heuristic *H1* guarantees that the expectation of $\mathcal{W}(S)$ is lower bounded by a constant factor of $\mathcal{W}(Y^*)$ (see Lemma 3.4.6), the heuristic *H2* guarantees that the expectation of $\mathcal{C}(S)$ is upper bounded by a constant factor of $\mathcal{C}(Y^*)$ (see Lemma 3.4.7).

It is also possible to obtain X^* for which Lemma 3.4.1 does not hold for heuristics *H1* and *H2*.

Fact 3.4.5 *If a and b are real numbers such that $|a| \leq 1$ and $|b| \leq 1$, then $\frac{1-ab}{2} \geq \frac{1}{2}\min\{\frac{1-a}{2}+\frac{1-b}{2}, \frac{1+a}{2}+\frac{1+b}{2}\}$.*

Proof. The proof is divided in two cases, depending on the sign of a and b.

Case 1: a and b have the same sign. Using the fact that $(a-b)^2 \geq 0$, we have

$$\begin{aligned} -ab &\geq -\frac{1}{2}(a^2+b^2), \\ &\geq \frac{-(|a|+|b|)}{2}, \\ &= \min\{\frac{a+b}{2}, \frac{-(a+b)}{2}\}. \end{aligned}$$

Case 2: a and b have different signs: The proof of this case follows from the fact that $-ab \geq 0 \geq \min\{\frac{a+b}{2}, \frac{-(a+b)}{2}\}$, which gives us that

$$\frac{1-ab}{2} \geq \frac{1}{2}\min\{\frac{1-a}{2}+\frac{1-b}{2}, \frac{1+a}{2}+\frac{1+b}{2}\}.$$

□

The following lemma shows that heuristic *H1* produces a cut where $E[\mathcal{W}(S)]$ is at least a factor of $\mathcal{W}(Y^*)$.

Lemma 3.4.6 *If S is the solution produced by heuristic H1 and $\xi(X^*, s, t) = -1$, then $E[\mathcal{W}(S)] \geq \frac{\mathcal{W}(Y^*)}{2}$.*

Proof. For a vertex $u \neq s$, the probability that edge us is in $\delta(S)$ is $\frac{1-\xi(X^*,u,s)}{2} = \frac{1-\cos(v_u, v_s)}{2} = \frac{1-y^*_{us}}{2}$.

If u and v are vertices not equal to s then the probability that edge uv is in $\delta(S)$ is the probability that $(s,u) \in \delta(S)$ and $(s,v) \notin \delta(S)$ plus the probability that $(s,u) \notin \delta(S)$ and $(s,v) \in \delta(S)$. That is,

$$\begin{aligned} \Pr[(u,v) \in \delta(S)] &= \frac{1-y^*_{us}}{2}\left(1-\frac{1-y^*_{vs}}{2}\right) + \left(1-\frac{1-y^*_{us}}{2}\right)\frac{1-y^*_{vs}}{2} = \frac{1-y^*_{us}y^*_{vs}}{2} \\ &\geq \frac{1}{2}\min\{\frac{1-y^*_{us}}{2}+\frac{1-y^*_{vs}}{2}, \frac{1+y^*_{us}}{2}+\frac{1+y^*_{vs}}{2}\} \qquad (3.7) \\ &\geq \frac{1}{2}\left(\frac{1-y^*_{uv}}{2}\right). \qquad (3.8) \end{aligned}$$

The inequality (3.7) is valid from Fact 3.4.5 and inequality (3.8) is valid from the triangle inequality constraints of R. More precisely,

$$\frac{1-y^*_{uv}}{2} \leq \frac{1-y^*_{us}}{2} + \frac{1-y^*_{vs}}{2}$$

and

$$\begin{aligned}\frac{1-y^*_{uv}}{2} &\leq \frac{1-y^*_{ut}}{2}+\frac{1-y^*_{vt}}{2} \\ &= \frac{1+y^*_{us}}{2}+\frac{1+y^*_{vs}}{2}. \end{aligned} \tag{3.9}$$

In inequality (3.9) we use the fact that $\xi(X^*, s, t) = -1$. In this case, the angle between s and t is π, and we have $\cos(v_u, v_s) = -\cos(v_u, v_t)$.

The expectation of $\mathcal{W}(S)$ is

$$\begin{aligned} E[\mathcal{W}(S)] &= \sum_{u<v} w_u w_v \Pr[uv \in \delta(S)] \\ &= \sum_{u\in V\setminus\{s\}} w_u w_s \Pr[us \in \delta(S)] + \sum_{u\in V\setminus\{s\}} \sum_{v\in V\setminus\{s\} \mid v<u} w_u w_v \Pr[uv \in \delta(S)] \\ &\geq \sum_{u\in V\setminus\{s\}} w_u w_s \frac{1-y^*_{us}}{2} + \frac{1}{2}\sum_{u\in V\setminus\{s\}} \sum_{v\in V\setminus\{s\} \mid v<u} w_u w_v \frac{1-y^*_{uv}}{2} \\ &\geq \frac{1}{2}\sum_{u<v} w_u w_v \frac{1-y^*_{uv}}{2} \\ &= \frac{\mathcal{W}(Y^*)}{2}. \end{aligned}$$

□

Now, we present a solution X^* where the expectation of $\mathcal{C}(S)$ for the cut produced by heuristic *H1* is unbounded. Consider the case where $(1 - y^*_{uv})/2 = 0$, which means no contribution to $\mathcal{C}(Y^*)$ and $(1 - y^*_{us})/2 = 1/2$. In this case the probability that (u, v) is in $\delta(S)$ is $1/2$, that implies a half contribution to the expectation of $\mathcal{C}(S)$. Adding *edge by edge*, each edge could have a similar behavior, and the expectation of $\mathcal{C}(S)$ will grow unbounded in relation to the value of $\mathcal{C}(Y^*)$. See Figure 3.3.

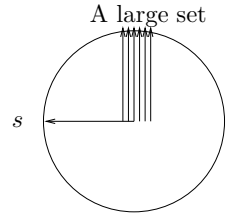


Figure 3.3: *H1* may produce a cut S with unbounded value of $\mathcal{C}(S)$.

Something symmetric happens for heuristic *H2*. In this case, we prove that $E[\mathcal{C}(S)]$ is at most $\mathcal{C}(Y^*)$, but we could not obtain a lower bound to $E[\mathcal{W}(S)]$.

Lemma 3.4.7 *If S is the solution produced by heuristic H2, then $E[\mathcal{C}(S)] \leq \mathcal{C}(Y^*)$.*

Proof. The probability that $(u, s) \in \delta(S)$ for $u \in V \setminus \{s\}$ is given by

$$\Pr[(u,s) \in \delta(S)] = \frac{1 - \xi(X^*, u, s)}{2} = \frac{1 - \cos(v_u, v_s)}{2} = \frac{1 - y^*_{us}}{2}.$$

If u and v are vertices not equal to s then

$$\Pr[(u,v) \in \delta(S)] = \left| \frac{y^*_{su} - y^*_{sv}}{2} \right| = \left| \frac{1 - y^*_{sv}}{2} - \frac{1 - y^*_{su}}{2} \right|,$$

that is exactly the probability to choose α between values $\xi(X^*, s, v)$ and $\xi(X^*, s, u)$. The triangle inequality constraint implies that

$$\frac{1 - y^*_{sv}}{2} - \frac{1 - y^*_{su}}{2} \leq \frac{1 - y^*_{uv}}{2} \quad \text{and} \quad \frac{1 - y^*_{su}}{2} - \frac{1 - y^*_{sv}}{2} \leq \frac{1 - y^*_{uv}}{2}.$$

Therefore, we have

$$\Pr[(u,v) \in \delta(S)] = \left| \frac{1 - y^*_{su}}{2} - \frac{1 - y^*_{sv}}{2} \right| \leq \frac{1 - y^*_{uv}}{2}, \quad \text{for all } \ (u,v) \in V \times V,$$

and

$$E[\mathcal{C}(S)] = \sum_{u<v} c_{uv} \Pr[(u,v) \in \delta(S)] \leq \sum_{u<v} c_{uv} \frac{1 - y^*_{uv}}{2} = \mathcal{C}(Y^*).$$

□

To obtain a solution X^* where the expectation of $\mathcal{W}(S)$ is not lower bounded, consider an edge (u, v), where $y^*_{uv} = -1$ (full contribution to $\mathcal{W}(Y^*)$), and $y^*_{us} = y^*_{vs}$. In this case, we have $\Pr[(u,v) \in \delta(S)] = 0$. Symmetrically to *H1*, adding *edge by edge*, each edge may have a similar behavior, and the expectation of $\mathcal{W}(S)$ does not increase, becoming unbounded in relation to the value of $\mathcal{W}(Y^*)$. See Figure 3.4.

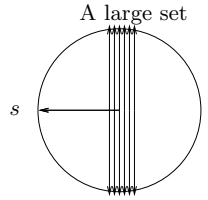


Figure 3.4: *H2* may produce a cut S with unbounded value of $\mathcal{W}(S)$.

The heuristic *H3* uses the hyperplane rounding strategy presented by Goemans and Williamson [22] for the Max-Cut problem. They proved that the probability that an edge (u, v) is cut by a random hyperplane is $\arccos(y^*_{uv})/\pi$. As can be seen in Figure 3.5, this value is always greater than $0.878(1 - y^*_{uv})/2$. Therefore, the following lemma is straightforward.

Lemma 3.4.8 *If S is the solution produced by heuristic H3, then* $E[\mathcal{W}(S)] \geq 0.878\mathcal{W}(Y^*)$.

The difficulty to obtain an upper bound to $\Pr[(u,v) \in \delta(S)] = \frac{\arccos(y^*_{uv})}{\pi}$ comes from the fact that $\frac{\arccos(y^*_{uv})}{\pi}$ cannot be bounded by $k\frac{1-y^*_{uv}}{2}$ for any constant k (see Figure 3.5).

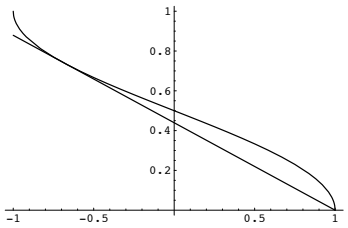


Figure 3.5: $\frac{\arccos(x)}{\pi}$ versus $0.878\left(\frac{1-x}{2}\right)$.

Analyzing heuristic *H4*, we found out some interesting properties. The probability that (u,v) belongs to (A,B) is the probability that r_1 and r_2 cut (u,v), which values $(\arccos(x_{uv})/\pi)^2$, times the probability that $u \in A \cup B$, which is $1/2$.

$$\Pr[(u,v) \in (A,B)] = \left(\frac{\arccos(x_{uv})}{\pi}\right)^2 \frac{1}{2}.$$

We can observe that this probability is upper and lower bounded by constant factors of the contribution in $\mathcal{C}(Y^*)$ and $\mathcal{W}(Y^*)$ (see Figure 3.6). More precisely, we have that $E[\mathcal{C}(A,B)] \leq \frac{1}{2}\mathcal{C}(Y^*)$ and $E[\mathcal{W}(A,B)] \geq 0.2023\mathcal{W}(Y^*)$. Therefore, the following lemma is valid.

Lemma 3.4.9 *If S is the solution produced by heuristic H4 then* $E[\mathcal{W}(S)] \geq 0.2023\mathcal{W}(Y^*)$.

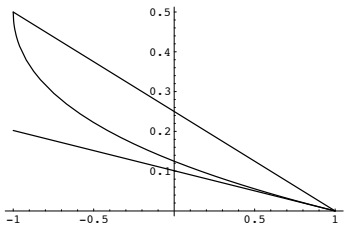


Figure 3.6: $0.2023\frac{1-x}{2}$ versus $\frac{1}{2}\frac{1-x}{2}$ versus $(\arccos(x)/\pi)^2\frac{1}{2}$.

Provided that the final cut S may have more edges than (A,B), we could not prove an upper bound for $E[\mathcal{C}(S)]$ by a factor of $\mathcal{C}(Y^*)$.

Consider now the case when $\xi(X^*, s, t) > -1$. One way to deal with this situation is to assure that one edge ij is separated. In Section 3.5 we show in details how to do this separation. By solving the relaxation with this separation for an arbitrary vertex i and all $j \in V \setminus \{i\}$ and choosing the minimum, we can obtain a lower bound that contains a pair of vertices with $\xi(X^*, s, t) = -1$. Another way to deal with this situation is to discard empty cuts in the heuristics. In this case, $\mathcal{C}(S)$ and $\mathcal{W}(S)$ will be multiplied by the same factor and therefore we can maintain the same analysis, except for heuristic *H1* that uses the fact $\xi(X^*, s, t) = -1$.

3.5 A Branch and Bound Algorithm for the Sparsest Cut Problem

The idea used in the branch and bound algorithm, named $B\&B$, for the Sparsest Cut Problem is simple. It consists of using the relaxation R as a dual bound and the heuristics as primal bounds. Although the relaxation has high time complexity, it leads to excellent lower bounds.

The first step consists of solving relaxation R, which leads to the first lower bound. The second step consists of applying the four heuristics based in this relaxation. Due to the fact the heuristics are much faster than solving the relaxation, each heuristic can be repeated several times without increasing substantially the overall time.

Initially, an arbitrary vertex is chosen, say vertex 0. While a gap exists between the lower bound and the upper bound, we branch the execution. At each iteration, we choose a node from the $B\&B$ tree and branch if there exists a gap between the lower and upper bounds. The branching consists of setting the angle between a vertex v_i and v_0 to be 0 or π. That is, we set $\xi(X^*, i, 0) = 1$ for $v_i = v_0$ and $\xi(X^*, i, 0) = -1$ for $v_i = -v_0$.

When $\xi(X^*, i, 0)$ is in $\{-1, +1\}$ for all $i \in V$, each vector v_i is in $\{-v_0, v_0\}$ and we obtain an integer solution. The constraints $|v_i| = |v_j|$ for $i, j \in V$ are necessary to conclude that the solution is integer. Since relaxation R_- does not contain the constraints $|v_i| = |v_j|$, the $B\&B$ algorithm using this relaxation may not work correctly. Even if each $\xi(X^*, i, 0)$ is in $\{-1, +1\}$, the vector solution may be spread over a straight line an will not represent a cut.

A node of the branch and bound tree is a semidefinite program Q composed of the program R and some integrality constraints. Let X_Q be a solution of the program Q. The "most" fractional variable for a given solution X_Q, say $ifrac(X_Q)$, is a vertex f such that $\xi(X^*, f, 0)$ is closed to zero. The $B\&B$ algorithm maintains a heap sorted by $ifrac(X_Q)$ for each node Q. In each step, the algorithm selects a vertex f from the heap, and, if the value of a solution of Q is smaller than the current upper bound, it produces a new

branch using the restrictions $\xi(X^*, f, 0) = 1$ and $\xi(X^*, f, 0) = -1$

If we add the constraint $x_{f0} = -x_{ff} = -x_{00}$ to R, any further solution will satisfy $\xi(X^*, 0, f) = -1$. On the other hand, if we add the constraint $x_{f0} = x_{ff} = x_{00}$, any further solution will satisfy $\xi(X^*, 0, f) = 1$. Observe that the constraint $x_{ff} = x_{00}$ already belongs to R.

The branch and bound algorithm beats the brute force algorithm (generate all possible cuts) to find an optimal solution and allows us to discover the integrality gap of R in graphs of the benchmark we have used. The branch and bound algorithm is presented in Figure 3.7.

ALGORITHM $B\&B(P)$, where $P = R(G, c, w)$
Let $\mathcal{H}$ be a heap of semidefinite programs Q, indexed by the most fractional variable (incident to vertex 0) in X_Q, say $ifrac(X_Q)$, where X_Q is a solution of Q.

1. $X_{UB} \leftarrow Heuristics(P)$
2. $\mathcal{H} \leftarrow \{P\}$.
3. While $\mathcal{H} \neq \emptyset$ do
4. $Q \leftarrow RemoveMax(\mathcal{H})$ and $f \leftarrow ifrac(X_Q)$.
5. if $R(Q) < \rho(X_{UB})$ then
6. $Q' \leftarrow Q \cup \{x_{f0} = -x_{ff} = -x_{00}\}$
7. $Q'' \leftarrow Q \cup \{x_{f0} = x_{ff} = x_{00}\}$
8. if $R(Q') < \rho(X_{UB})$ then
9. if $X_{Q'}$ is a feasible cut then $X_{UB} \leftarrow X_{Q'}$
10. else
11. $X' \leftarrow Heuristics(Q')$
12. if $\rho(X') < \rho(X_{UB})$ then $X_{UB} \leftarrow X'$
13. $\mathcal{H} \leftarrow \mathcal{H} \cup \{Q'\}$.
14. if $R(Q'') < \rho(X_{UB})$ then
15. perform the corresponding steps executed in lines 10–14 for Q''.
16. Return X_{UB}.

Figure 3.7: Semidefinite based branch and bound algorithm.

3.5.1 Considerations involving LP and SDP problems

A common strategy used in many Branch & Bound algorithms to solve NP-hard problems is to consider an ILP formulation and to use additional constraints to its relaxations. Because SDP is a generalization of LP, every formulation and relaxation valid in LP is also valid to SDP. The difference is due to the fact that LP can be solved exactly and

SDP can be solved within a small error ϵ, in polynomial time complexity in the input size and in $\log(1/\epsilon)$.

To decide between an implementation using SDP or LP, it is necessary to consider some aspects. The state of the art in LP solvers reached a point so that there are many robust, fast and stable implementations, such as CPLEX and XPRESS-MP. These solvers also allow integer programming formulations and features as pre-processing and branch-and-cut algorithms. The use of SDP solvers in combinatorial optimization problems is relatively new, compared to LP.

To compare the time to solve SDP and LP relaxations in instances of equivalent size see Table 3.1 (column *Av. CPU time*) and Table 3.2 in Section 3.6. In these experiments, for instances with $n = 40$ vertices, the time to solve the associated LP was approximately 400 times faster than SDP.

Another advantage of LP is the use of a *warm start* to solve the relaxations associated to each node. Although there are theoretical works that describe the possibility of a good initial point for a family of problems with SDP programs [41], some solvers, such as SDPA package, used in the experiments presented in this paper, have restrictions that do not allow this optimization. See additional information in Section 3.6. It is well known that ILP Solvers may use a previous basis to obtain a new solution of a branching step.

On the other hand, SDP has some advantages. The use of SDP constraints may generate better lower bounds that may reduce the number of visited nodes. Moreover, given an SDP solution X^*, we can take advantage of its geometric interpretation obtaining a set $\mathcal{V}$ of vectors. This set of vectors were used in the rounding strategies of heuristics *H3* and *H4*. Given an SDP solution X^*, we can obtain a matrix $\mathcal{V}$ such that $X^* = \mathcal{V}\mathcal{V}^T$ using the Cholesky Decomposition of X^*.

In many LP formulations that use modulus, the relaxation usually does not give useful information and the use of a branch and bound over SDP may be competitive.

3.6 Computational Results

In a first attempt, we used the SDPA package [49] as SDP solver. It is an implementation of a primal-dual interior-point method for semidefinite programming. Unfortunately, the *warm start* was not implemented in the SDPA package, and we have to recompute the semidefinite program in each node of the branch and bound tree. If (X^0, Y^0) is a pair of initial primal and dual points, the SDPA solver looks for a solution (X, Y) such that $0 \preceq X \preceq omegaStar \times X^0$ and $0 \preceq Y \preceq omegaStar \times Y^0$, where *omegaStar* is a parameter [21]. Our effort to compute a good initial point in the branch and bound procedure using the SDPA solver did not work because both the mentioned constraints limit the domain and non-optimal solutions were generated by the solver.

The experiments were performed following the same approach used by Goemans and Williamson [22] for the Max-Cut Problem. We generated four types of instances: types A, B, C and D. In type A instances, each vertex has weight 1, and each edge is selected with probability 0.5 to receive cost 1. If the edge is not selected, its cost is zero. In instances of type B, the vertex weight is a random rational number uniformly distributed in $[0, 50]$, and the edge cost is also a rational number uniformly distributed in $[0, 50]$. In instances of type C, each vertex has weight 1, and each edge is selected with probability $9/|V|$ to receive cost 1. The edge cost is zero if the edge is not selected. While type A instances correspond to dense graphs, type C instances correspond to sparse ones because the expected vertex degree is 9. In the instances of type D, we arbitrarily divided the set V in two disjoint sets, V_1 and V_2, where $|V_1| = |V_2|$. Each vertex has weight 1. The edges inside V_1 or V_2 are selected with probability 0.5 to receive cost 1. Edges linking vertices in V_1 with vertices in V_2 are selected with probability 0.25 to receive cost 1. All edges not selected have cost zero.

We produced a set of instances containing 20, 30 and 40 vertices. For each instance, we obtained an optimum relaxed solution using SDPA. The CPU times are given in seconds on a Xeon 2.4 GHz with 1024MB of RAM and Linux operational system. In Table 3.1 we have the time for each group of instances.

Typ	Siz	Number of Instances	Av. CPU time (s)
A	20	400	155
	30	95	6668
	40	5	30564
B	20	400	108
	30	146	4305
	40	26	18718
C	20	400	137
	30	95	6568
	40	2	26660
D	20	400	166
	30	93	6695
	40	3	28980

Table 3.1: Time to solve R with the SDPA solver.

In order to make further investigation in formulation R, we removed the semidefinite constraint, obtaining a new relaxation, named $R_X := R \setminus \{X \succeq 0\}$, which was solved with the XPRESS-MP LP solver. Note that program R_X is a relaxation to Sparsest Cut, but not a formulation. Thus an optimum integer solution of R_X has no upper bound meaning. See Table 3.2. The SDP gap for an instance I is $\frac{R(I)}{R_X(I)}$. Because of the time

constraint, the experiment with SDPA did not find a solution to all instances in Table 3.2. To compute the SDP gap we used the solver described in the next subsection.

Type	Size	Number of Instances	Average SDP Gap	Max. SDP Gap	Av. CPU Time (s)
A	20	400	1.003	1.06	0.55
	30	300	1.010	1.10	7.05
	40	20	1.030	1.13	71.6
B	20	400	1.000	1.00	0.41
	30	300	1.000	1.00	4.33
	40	20	1.000	1.00	28.62
C	20	400	1.001	1.05	0.50
	30	300	1.002	1.09	5.6
	40	20	1.000	1.01	39.35
D	20	400	1.003	1.06	0.52
	30	300	1.010	1.08	7.00
	40	20	1.030	1.10	63.42

Table 3.2: Results of relaxation R_X: with no semidefinite constraint.

The XPRESS-MP solver found solutions to relaxation R_X more than 200 times faster than SDPA found solutions to relaxation R. The quality of the solution obtained by XPRESS-MP was at most 13% and less than 3% worst, on average.

Thus we proposed a new and faster way to solve the SDP programming to Sparsest Cut in next subsection.

3.6.1 An SDP solver to Sparsest Cut Based in Cut Generation Approach

The poor time performance of SDPA over Sparsest Cut Formulation led us to try another way to solve an SDP formulation. We present an algorithm that solves the relaxation R_X using XPRESS-MP, adds violated SDP constraints and iterates until the problem becomes positive semidefinite, within a small error.

We implemented the algorithm proposed by Fraticelli in subsection 3.3. of [19]. This algorithm consists in: giving a matrix X, return true if $X \succeq 0$. Otherwise, obtain a vector α such that $\alpha X \alpha^T < 0$. That is, we solve the relaxation, execute Fraticelli algorithm and if it is the case, add the violated constraint $\alpha X \alpha^T \geq 0$ to the formulation and iterate.

This strategy has a considerable slow convergence, because the cut generated by Fraticelli Algorithm in general is superficial. To improve this approach, we take the result of the LP solver, say X, and add a constant factor in the diagonal. Consider an arbitrary $\lambda \in \mathbb{R}^+$ and the matrix X', where $x'_{ij} = x_{ij}$ if $i \neq j$, and $x'_{ij} = (1+\lambda)x_{ij}$ otherwise. If X' is not positive semidefinite then $\alpha X' \alpha^T < 0$ is a deeper cut to X, where $\alpha X \alpha^T < 0 - \sum_{\forall i} \lambda x_{ii} \alpha_i^2$.

We start with $\lambda = 1$. If $X' \not\succeq 0$ we add the deeper constraint $\alpha X \alpha^T < 0$ and double λ until $X' \succeq 0$, at which point we iterate calling LP solver. If $X' \succeq 0$, we divide λ by 2 while $X' \succeq 0$. If λ is close to zero, we conclude that X is positive semidefinite within a small error. Otherwise, the small value of λ made $X' \not\succeq 0$. In this case we add the SDP cut and iterate with the LP solver. In practical experiments, we consider X positive semidefinite when X' is positive semidefinite with $\lambda < 10^{-8}$.

The constraint $-x_{00} \leq x_{ij} \leq x_{00}$ is an SDP valid inequality to Sparsest Cut and we decided to add it for each pair $\{i, j\}$ in the beginning of the algorithm.

We considered two experiments: in one, the algorithm inserts all triangle inequalities in the beginning and in the other the violated triangle inequalities are iteratively added in each node of the branch and bound tree. In this case, the algorithm search for violated triangle constraints always in different directions. This approach improve the time and the use of memory, what allowed us to solve larger Sparsest Cut Instances.

Each heuristic was executed 50 times on the same instance, returning the best obtained solution. Empty cuts were discarded.

For each instance and heuristic, we also saved a number k that is the iteration in which the respective heuristic found its best solution. The heuristic *H2* does not have a k associated because there are at most $|V| - 1$ possible solutions, and we have obtained all of them. For the heuristics *H1* and *H3* we also counted the empty cuts to compute the corresponding value of k, but for heuristic *H4*, we did not. The time to solve the heuristics without the time to solve the relaxation R is negligible (less than 0.3% of the time to solve R).

In the *Visited Nodes* column in tables 3.4 and 3.6, we only considered non-leaf nodes in the branch tree. The *integrality gap* for an instance I is $\frac{B\&B(I)}{R(I)}$. The *deviation factor* of a heuristic H_i is $\frac{H_i(I)}{B\&B(I)}$.

The results obtained with all triangle inequalities inserted in the beginning are described in tables 3.3 and 3.4 and the results obtained with triangle inequalities inserted only when necessary are described in tables 3.6 and 3.5.

Adding triangle inequalities on demand gave us a possibility to solve larger instances, of size 50 and 60, because this approach is faster and has a better use of memory. The program solved the instances in half of the time, compared with adding all triangle constraints in the beginning.

The new implementation using semi-infinite cuts and linear programming is at least 80 times faster and at most 1500 times faster than using SDPA solver. Another important advantage of the linear programming approach is the reuse of the base in each node of branch and bound tree.

The relaxation R has surprisingly good integrality gap. For almost all tests the obtained relaxation value was equal to an optimum cut value. The quality of the relaxation

Typ	Siz	Number of Instances	Integrality Gap		Average Deviation Factor				Maximum Deviation Factor			
			Av.	Max	*H1*	*H2*	*H3*	*H4*	*H1*	*H2*	*H3*	*H4*
A	20	400	1.0000	1.0082	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0119
	30	300	1.0000	1.0026	1.0015	1.0000	1.0000	1.0000	1.2455	1.0115	1.0000	1.0000
	40	20	1.0002	1.0030	1.0081	1.0001	1.0001	1.0001	1.1625	1.0015	1.0015	1.0015
B	20	400	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	30	300	1.0000	1.0000	1.0000	1.0000	1.0001	1.0000	1.0000	1.0000	1.0218	1.0000
	40	20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
C	20	400	1.0000	1.0002	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	30	300	1.0001	1.0058	1.0018	1.0002	1.0003	1.0003	1.3344	1.0533	1.0685	1.0533
	40	20	1.0000	1.0001	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
D	20	400	1.0000	1.0020	1.0004	1.0000	1.0000	1.0000	1.0769	1.0035	1.0000	1.0000
	30	300	1.0000	1.0070	1.0000	1.0000	1.0000	1.0000	1.0031	1.0031	1.0031	1.0031
	40	20	1.0001	1.0003	1.0000	1.0000	1.0000	1.0000	1.0001	1.0000	1.0000	1.0000

Table 3.3: Result using XPRESS-MP and all triangle inequalities from the beginning.

leads to good heuristics and the branch and bound algorithm visited a small number of nodes in the branch and bound tree.

The performance of the branch and bound algorithm depends basically on the semidefinite programming solver, which is still a very active area. Naturally, the $B\&B$ algorithm with formulation R may also become faster using better solvers.

3.7 Conclusion

We analyzed a known relaxation for the Sparsest Cut Problem using semidefinite programming. We also presented an exact algorithm and heuristics based on this formulation. The presented heuristics obtained solutions with value very close to the optimum, and the exact algorithm could be executed on small and medium sized instances. We implemented a new solver to semidefinite programming using LP programming. This solver has a good performance and it is very promising in SDP problems that are dominated by linear programming constraints and also to general SDP problems.

The performed tests showed that the use of semidefinite programming for this problem is a very promising strategy. With the continuous advances on the theory of semidefinite programming (as occurred with the theory of linear programming), we hope the presented algorithms will obtain solutions for larger graphs in less time.

Typ	Siz	Number of Instances	Average k			Av. CPU time		Visited Nodes		SDP Cuts		Trig Cuts.	
			H1	*H3*	*H4*	Heu	*B&B*	Av.	Max	Av.	Max	Av.	Max
A	20	400	1.1	1.1	2.2	1.2	1.4	1.0	1.0	6.5	215.0	0.0	0.0
	30	300	1.5	1.2	2.3	27.2	36.8	1.0	2.0	17.5	665.0	0.0	0.0
	40	20	2.8	1.4	2.0	654.8	1070.5	1.0	1.0	85.9	976.0	0.0	0.0
B	20	400	1.0	1.1	2.2	0.4	1.1	1.0	1.0	2.7	90.0	0.0	0.0
	30	300	1.0	1.1	2.5	3.7	6.1	1.0	1.0	0.7	65.0	0.0	0.0
	40	20	1.0	1.0	2.5	19.9	22.2	1.0	1.0	0.2	4.0	0.0	0.0
C	20	400	1.0	1.1	2.0	1.0	1.2	1.0	1.0	4.0	105.0	0.0	0.0
	30	300	1.3	1.3	2.2	17.8	22.5	1.1	26.0	12.6	818.0	0.0	0.0
	40	20	1.0	1.1	2.2	113.9	120.9	1.0	1.0	5.0	84.0	0.0	0.0
D	20	400	1.3	1.2	2.3	1.3	1.5	1.0	1.0	7.4	119.0	0.0	0.0
	30	300	1.1	1.2	2.7	30.3	39.0	1.0	4.0	17.6	194.0	0.0	0.0
	40	20	3.4	2.0	6.2	576.5	826.8	1.0	1.0	75.2	514.0	0.0	0.0

Table 3.4: Result using Xpress-MP and all triangle inequalities from the beginning.

3.8 Future Works

One part of this work describe a way to solve SDP with even more than a hundred thousand constraints in less CPU time compared with a well known SDP solver to a specific problem. As the method described is general, it is promising the implementation of a interface that extends the proposed solver to be a general SDP solver and compare it to others solvers in a public benchmark.

3.9 Acknowledgements

We would like to thank Cristina G. Fernandes for her helpful suggestions to this article.

Typ	Siz	Number of Instances	Integrality Gap		Average Deviation Factor				Maximum Deviation Factor			
			Av.	Max	*H1*	*H2*	*H3*	*H4*	*H1*	*H2*	*H3*	*H4*
A	20	400	1.0000	1.0082	1.0014	1.0000	1.0000	1.0000	1.5612	1.0119	1.0000	1.0000
	30	300	1.0000	1.0029	1.0009	1.0000	1.0000	1.0000	1.2222	1.0115	1.0000	1.0000
	40	20	1.0002	1.0032	1.0113	1.0001	1.0001	1.0001	1.2259	1.0015	1.0015	1.0015
	50	20	1.0002	1.0025	1.0000	1.0000	1.0000	1.0000	1.0005	1.0005	1.0005	1.0000
	60	20	1.0004	1.0044	1.0057	1.0005	1.0005	1.0003	1.0660	1.0095	1.0095	1.0063
B	20	400	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	30	300	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	40	20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	50	20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	60	20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
C	20	400	1.0000	1.0004	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	30	300	1.0001	1.0058	1.0016	1.0000	1.0001	1.0001	1.3072	1.0142	1.0142	1.0142
	40	20	1.0000	1.0001	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	50	20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	60	20	1.0032	1.0518	1.0393	1.0000	1.0000	1.0108	1.6285	1.0000	1.0000	1.1731
D	20	400	1.0000	1.0020	1.0002	1.0000	1.0000	1.0000	1.0343	1.0081	1.0000	1.0000
	30	300	1.0000	1.0070	1.0000	1.0000	1.0000	1.0000	1.0031	1.0031	1.0031	1.0102
	40	20	1.0001	1.0007	1.0018	1.0000	1.0000	1.0000	1.0365	1.0000	1.0001	1.0001
	50	20	1.0001	1.0003	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	60	20	1.0001	1.0003	1.0000	1.0000	1.0000	1.0005	1.0000	1.0000	1.0000	1.0105

Table 3.5: Result using XPRESS-MP and triangle inequalities on demand.

Typ	Siz	Number of Instances	Average k			Av. CPU time		Visited Nodes		SDP Cuts		Trig Cuts.	
			H1	*H3*	*H4*	Heu	*B&B*	Av.	Max	Av.	Max	Av.	Max
A	20	400	1.2	1.1	2.1	0.6	0.7	1.0	1.0	5.6	240.0	2206.8	5995.0
	30	300	1.4	1.3	2.2	11.9	16.9	1.0	3.0	17.3	970.0	9412.2	19600.0
	40	20	2.5	1.4	2.4	351.6	515.7	1.0	1.0	85.2	927.0	27587.3	48560.0
	50	20	1.9	1.5	2.2	2560.4	6637.1	1.1	2.0	133.5	1076.0	61945.8	103034.0
	60	20	2.6	2.8	4.5	15812.5	29055.5	1.1	2.0	285.9	2162.0	112609.4	189998.0
B	20	400	1.0	1.0	1.9	0.3	0.9	1.0	1.0	0.0	0.0	1672.9	2448.0
	30	300	1.0	1.0	1.9	2.4	3.5	1.0	1.0	0.0	0.0	5903.4	8740.0
	40	20	1.0	1.0	1.9	11.8	14.2	1.0	1.0	0.0	0.0	14455.6	16720.0
	50	20	1.0	1.0	2.2	43.9	45.7	1.0	1.0	0.0	0.0	27988.8	34560.0
	60	20	1.0	1.0	1.9	129.2	132.2	1.0	1.0	0.0	0.0	50170.2	56728.0
C	20	400	1.1	1.1	1.9	0.5	0.7	1.0	1.0	3.6	124.0	2348.7	6018.0
	30	300	1.3	1.2	2.5	11.3	14.1	1.1	21.0	7.9	449.0	12425.9	23549.0
	40	20	1.0	1.1	1.9	75.7	79.2	1.0	1.0	2.9	57.0	34168.3	48797.0
	50	20	1.0	1.0	1.6	315.2	334.5	1.0	1.0	0.0	0.0	74637.8	92456.0
	60	20	3.7	2.4	4.1	3893.5	5913.7	1.1	3.0	26.1	417.0	122591.5	172376.0
D	20	400	1.3	1.1	2.2	0.5	0.7	1.0	17.0	8.2	777.0	1802.3	6840.0
	30	300	1.2	1.2	2.4	8.3	17.7	1.0	8.0	16.2	305.0	7278.5	22027.0
	40	20	2.3	3.0	2.9	244.1	393.1	1.1	2.0	78.5	657.0	23447.0	54382.0
	50	20	1.0	1.4	3.0	744.8	1038.7	1.0	1.0	43.7	247.0	50959.7	108692.0
	60	20	1.3	1.8	4.5	2548.1	3687.8	1.0	1.0	66.4	291.0	95160.4	195552.0

Table 3.6: Result using Xpress-MP and triangle inequalities on demand.

Chapter 4

A Continuous Facility Location Problem and its Application to a Clustering Problem

Abstract

We introduce a new problem, which we denote by Continuous Facility Location Problem (ConFL), and its application to the k-Means Problem. Problem ConFL is a natural extension of the Uncapacitated Facility Location Problem where a facility can be any point in $\mathbb{R}^q$ and distance functions are not restricted to euclidean or squared euclidean distances. The proposed algorithms are based on a primal-dual technique for spaces with constant dimension. For the ConFL Problem, we present algorithms with approximation factors $1.861 + \epsilon$ for euclidean distances and $9 + \epsilon$ for squared euclidean distance. For the k-Means Problem (that is restricted to squared euclidean distance), we present an algorithm with approximation factor $54 + \epsilon$. All algorithms have good practical behaviour in small dimension. Comparisons with known algorithms show that the proposed algorithms also have good practical behaviour.

4.1 Introduction

We consider a continuous version of the Uncapacitated Facility Location Problem and its application to the k-Means Problem. In both problems we have a set of clients $C \subset \mathbb{R}^q$ and we have to find a non-empty set $F \subset \mathbb{R}^q$ and connect each client to the cheapest facility connection in F. In the Continuous Facility Location problem, the value of a solution is the total cost to connect the clients to the respective facility plus the cost to open facilities, which is $f \cdot |F|$, where f is the cost to open one facility. We used the term

continuous because the facility can be any point in the continuous space, in opposition with traditional facility locations where the set of possible facilities if pre-defined. In the k-Means Problem the set F must have exactly k facilities and the cost of a solution is the total cost to connect the clients to the respective facility. In both cases, we are looking for solutions with minimum cost.

The (Uncapacitated) Facility Location and Clustering Problems have many practical applications [17] and have been widely investigated in the literature. Applications appear in many areas of operations research, data mining, information retrieval, image processing, data compression, web applications, etc.

Our main interest is in the development of approximation algorithms. An α-approximation algorithm for a minimization problem is an efficient (polynomial-time) algorithm that produces, for any instance, a solution with value that is at most a factor of α larger than the value of an optimum solution.

The first approximation algorithm for the Facility Location Problem is a greedy algorithm with approximation factor $O(\log n)$, where n is the number of facilities and clients in the input instance, due to Hochbaum [26] in 1982. This result is the best possible, assuming $P \neq NP$. For the metric case, the first constant approximation factor is 3.17, due to Shmoys, Tardos, and Aardal [44]. There are many other techniques and bounds in the literature. One of the main techniques is the use of linear programming and the primal-dual approach. Jain and Vazirani [30] presented a primal-dual 3-approximation algorithm for the Facility Location Problem. The best approximation factor is 1.52, by an algorithm due to Mahdian, Ye, and Zhang [39], and uses the primal-dual strategy with factor revealing and cost scaling techniques. Regarding hardness results, Guha and Khuller [23] showed that any approximation algorithm must have an approximation factor of at least 1.463, assuming $NP \subsetneq DTIME[n^{O(\log \log n)}]$. The Uncapacitated Facility Location Problem is also a good example of the practical impact of better approximation factors. For example, the 1.61-approximation algorithm presented by Jain, Mahdian, Markakis, Saberi and Vazirani [28] obtained solutions with almost no error in most of the cases, indicating that approximation factors are also a good measure for the practical behavior of algorithms.

The most famous clustering problem is the l_2^2-Clustering, also known as the k-Means problem. This problem is NP-hard even for $k = 2$ [16]. One of the most popular algorithms for the k-Means problem is Lloyd's Algorithm [38], for which there exists fast implementations [31]. Although this algorithm has a good practical behavior, it has no approximation factor or a proved polynomial-time complexity. In fact, there are examples for which the algorithm obtain arbitrarily bad clusterings. Considering approximation algorithms, Matousek [40] presented an approximation scheme to k-Means when q, ϵ and k are constants, where the approximation factor is ϵ dependent. There are many other

approximation schemes under these conditions. More recently, Kumar, Sabharwal, and Sen [34] presented a linear time approximation scheme, for k and ϵ constant, independent of q. When ϵ and q are constants, Kanungo, Mount, Netanyahu, Piatko, Silverman, and Wu [32] present a $(9+\epsilon)$-approximation algorithm. Unfortunately, most of the above mentioned algorithms are very complicated or have high time complexity for small values of ϵ. In [30], Jain and Vazirani described a primal-dual approximation algorithm with factor 108 to k-Means problem (independent of k and q). Although this algorithm has a large factor, it leads to a practical algorithm (it is fast and does not require to solve any linear program).

The $(9+\epsilon)$- and the 108-approximation algorithms for the k-Means can be easily adapted to ConFL with the same approximation factors, since k is not restricted to a constant. To implement this adaptation, it is sufficient to execute k-Means for all values of k in $\{1,\ldots,|C|\}$ and return the solution with minimum cost with respect to ConFL cost function. We note that all mentioned approximation schemes for k-Means have exponential time in k.

Our results: We present approximation results for the Continuous Facility Location Problem. Adapting approximation algorithms presented in [30, 28], we obtain a $(1.861+\epsilon)$-approximation algorithm in metric l_2 and a $(9+\epsilon)$-approximation algorithm with l_2^2 distance. For the l_2-Clustering we obtain a $(6+\epsilon)$-approximation algorithm. For the k-Means Problem (l_2^2-Clustering problem), we propose an algorithm similar to the one presented in [30] with approximation factor $54+\epsilon$. All results are valid for constant q. Our main contribution is a way to deal with an exponential number of centers in polynomial-time. We report computational results for the k-Means Problem, showing that for small values of q the algorithm produces solutions close or better than known algorithms. For some instances, it gives better solutions than the ones obtained by a combination of Lloyd's and local search algorithms [31].

We implement some variants of the algorithms and make an empirical analysis of them.

4.1.1 Definitions and Notation

We denote by $\mathcal{I}$ the set of tuples (instances) (C,q,d), where $C \subset \mathbb{R}^q$ is a set of q-dimensional points and d is a reflexive distance function over $\mathbb{R}^q$. We denote by $\mathcal{I}_f$, for a real value $f>0$, the set of tuples (C,q,d,f), where $(C,q,d)\in\mathcal{I}$, and by $\mathcal{I}^k$ for a positive integer k, the set (C,q,d,k) where $(C,q,d)\in\mathcal{I}$. Given a set $P\subset\mathbb{R}^q$ and a point $p\in\mathbb{R}^q$, we denote the distance from p to P as $d(p,P)=\min\{d(p,p_x):\ p_x\in P\}$. The set of solutions of an instance $I\in\mathcal{I}_f$ (resp. $I\in\mathcal{I}^k$), in a q-dimensional space, is a set $\mathcal{S}_I=\{P\subset\mathbb{R}^q:\ P\neq\emptyset\}$ (resp. $\mathcal{S}_I^k=\{P\subset\mathbb{R}^q:\ |P|=k\}$). We use two functions to define the value of a solution, according to the problem: Given a set $P\in\mathcal{S}_I$ and a real value f,

we define $v_I(P) = \sum_{j \in C} d(j, P)$ and $v_I^f(P) = v_I(P) + f|P|$. When the instance I is clear from the context, we use $\mathcal{S}$, $v(P)$ and $v^f(P)$ to denote $\mathcal{S}_I$, $v_I(P)$ and $v_I^f(P)$, respectively.

Now, we define the d-Clustering and the Continuous Facility Location (ConFL) Problems. In the first we look for exactly k points (facilities) where a set of clients must connect. The cost of a solution is the total cost to connect all clients.

Problem 1 (d-Clustering) *Given $I \in \mathcal{I}^k$, find a solution $P \in \mathcal{S}_I^k$ such that $v_I(P)$ is minimum.*

In the Continuous Facility Location Problem (ConFL) we can open any number of facilities, but we pay a fixed value in the objective function for each facility that appears in the solution. This problem can be formally defined as:

Problem 2 (ConFL) *Given an instance $I \in \mathcal{I}_f$, find a solution $P \in \mathcal{S}_I$ such that $v_I^f(P)$ is minimum.*

Note that, although we can open any facility $i \in \mathbb{R}^q$, the cost to open i is equal to the cost to open any other facility.

4.2 From the Discrete to the Continuous Version

The algorithms we present are based in algorithms for two related problems, defined as follows. Denote by $\mathcal{I}'$ the set of tuples (F, C_t, c), where F is a finite set of facilities, C_t is a finite set of cities and $c : F \times C_t \to \mathbb{R}^+$ is such that c_{ij} is the cost to connect city $j \in C_t$ to facility $i \in F$. We denote by $\mathcal{I}_{\text{KM}}$, for a positive integer k, the set of tuples (F, C_t, c, k), where $(F, C_t, c) \in \mathcal{I}'$ and by $\mathcal{I}_{\text{FL}}$ the set of tuples (F, C_t, c, f), where $(F, C_t, c) \in \mathcal{I}'$, and f is a function $f : F \to \mathbb{R}^+$ associating a positive real cost $f(i)$ for each facility $i \in F$.

The set of solutions for an instance $I \in \mathcal{I}_{\text{FL}}$ (resp. $I \in \mathcal{I}_{\text{KM}}$) is the set $\mathcal{G}_I$ (resp. $\mathcal{G}_I^k$), where $\mathcal{G}_I = \{S \subseteq F | S \neq \emptyset\}$ and $\mathcal{G}_I^k = \{S \subseteq F : |S| = k\}$. We also remove the index I from the notation whenever the instance is clear from the context. We define a cost function between a city j and a set of facilities F' as $c(j, F') = \min\{c_{ij} : i \in F'\}$. Given a solution $S \in \mathcal{G}$ for an instance I, we define $w_{\text{FL}}(S) = \sum_{i \in S} f(i) + \sum_{j \in C_t} c(j, S)$ and given a solution $S \in \mathcal{G}^k$ we define $w_{\text{KM}}(S) = \sum_{j \in C_t} c(j, S)$.

The k-Median problem (KM) consists of opening exactly k facilities in a way to minimize the total cost to connect each city to its closest facility.

Problem 3 (KM) *Given an instance $I \in \mathcal{I}_{\text{KM}}$, find a solution $S \in \mathcal{G}_I^k$ such that $w_{\text{KM}}(S)$ is minimum.*

The Facility Location problem (FL) consists of opening a set of facilities and connecting each city to a closest open facility in a way to minimize the total cost to open facilities and to connect all cities to the open facilities. It can be formally defined as:

Problem 4 (FL) *Given an instance $I \in \mathcal{I}_{\mathrm{FL}}$, find a solution $S \in \mathcal{G}_I$ such that $w_{\mathrm{FL}}(S)$ is minimum.*

We also need the definition of center, as follows:

Definition 2 (Center) *Given an instance $(C, q, d) \in \mathcal{I}$, a point $x \in \mathbb{R}^q$ is a center of S, denoted by center(C), if $\sum_{y \in C} d(x, y)$ is minimum.*

Lemma 4.2.1 *Given an instance $(C, q, d) \in \mathcal{I}$, if d is the squared Euclidean distance, then the center of S is unique. Moreover, we have center$(C) = \left\{\frac{\sum_{y \in C} y}{|C|}\right\}$.*

Proof. When d is a the squared Euclidean distance function,

$$d(x, C) = \sum_{y \in C} d(x, y) = \sum_{y \in C} \sum_{i=1}^{q} (x_i - y_i)^2.$$

As $\sum_{y \in C} d(x, y)$ is a paraboloid, the point of minimum have the property that the derivative $\frac{\partial(d(x,C))}{\partial(x_i)} = 0$, for all $i \in \{1, \ldots, q\}$. So, for any $i \in \{1, \ldots, q\}$ we have:

$$\begin{aligned} \frac{\partial(d(x, C))}{\partial(x_i)} &= \frac{\partial(\sum_{y \in C} d(x, y))}{\partial x_i} \\ &= \frac{\partial(\sum_{y \in C} (x_i - y_i)^2)}{\partial x_i} \\ &= \sum_{y \in C} (2x_i - 2y_i). \end{aligned}$$

Therefore, the minimum is attained when

$$x_i = \frac{\sum_{y \in C} y_i}{|C|}.$$

□

For some distance functions, the center may not be unique. For example, in the Euclidean space it may occur when the points are collinear.

We first describe algorithms using the squared Euclidean distance function, where the center is unique and easily computable. In Section 4.4.1 we pointed how the algorithms and approximations factors can be extended to the Euclidean metric.

Denote by l_2^2-ConFL the problem ConFL with distance function l_2^2. Let $S^* \in \mathcal{S}$ be an optimal solution to an instance $I \in \mathcal{I}_f$ for l_2^2-ConFL, where $I = (C, q, d, f)$. Consider a set of centers of all nonempty subsets of C, $\Phi(C) = \{center(C') : \emptyset \neq C' \subseteq C\}$. The solution S^* is made just by centers. If not, we can find a better solution than S^*. As one can see, $S^* \subseteq \Phi(C)$.

A simple way to adapt an algorithm from FL to the l_2^2-ConFL problem is to define F as the set $\Phi(C)$. Let $I_f = (C, q, l_2^2, f) \in \mathcal{I}_f$ be an instance to l_2^2-ConFL problem. Define $I_f^R = (F', C_t, c, f') \in \mathcal{I}_{\text{FL}}$ as an instance to FL where $C_t = C$, $F' = \Phi(C)$, $f'(i) = f$, $\forall i \in F'$ and $c_{ij} = l_2^2(i, j), \forall i \in F', j \in C_t$, where $l_2^2(i, j)$ is the distance between point $i \in \mathbb{R}^q$ and $j \in \mathbb{R}^q$ in l_2^2 distance function.

Proposition 4.2.2 *If S_1^* and S_2^* are optimum solutions for instances I_f and I_f^R, respectively for* l_2^2-ConFL *and* FL*, then $v^f(S_1^*) = w_{\text{FL}}(S_2^*)$. Furthermore, any solution S_2 for instance I_f^R is directly converted to a solution S_1 for I_f, with $v^f(S_1) = w_{\text{FL}}(S_2)$.*

Proof. The proof is straightforward from the fact that in any optimum solution S_1^* of I_f an open facility i must be a center of the points connected to it in S_1^*. Otherwise a solution of smaller cost can be obtained. □

A way to adapt an algorithm for the KM problem to the l_2^2-Clustering problem is similar. Let $I = (C, q, l_2^2, k) \in \mathcal{I}^k$ be an instance of the l_2^2-Clustering problem. Define $I_k^R = (F', C_t, c, k)$ as an instance to KM where $C_t = C$, $F' = \Phi(C)$ and $c_{ij} = l_2^2(i, j), \forall i \in F', j \in C$.

Proposition 4.2.3 *If S_1^* and S_2^* are optimum solutions for instances I_k and I_k^R, respectively, for* l_2^2-Clustering *and k-Median, then $v(S_1^*) = w_{\text{KM}}(S_2^*)$. Furthermore, any solution S_2 for instance I_k^R is directly converted to a solution S_1 for I^k keeping $v(S_1) = w_{\text{KM}}(S_2)$.*

The proof of Proposition 4.2.3 is similar to the proof of Proposition 4.2.2.

Corollary 4.2.4 *Given a ρ-approximation algorithm $\mathcal{A}$ for the* FL *or* KM *problems, denote by $\hat{\mathcal{A}}$ the corresponding algorithm to* l_2^2-ConFL *or to* l_2^2-Clustering *problem, applying algorithm $\mathcal{A}$ with set of facilities $F = \Phi(C)$. We have that $\hat{\mathcal{A}}$ is a ρ-approximation algorithm to the respective problem.*

Although very simple, the straightforward implementation may lead to very large instances, as $\Phi(C)$ is exponential in the size of I_f and I_k. In the next section, we show how to adapt one algorithm to FL and one algorithm for k-Median to l_2^2-ConFL and l_2^2-Clustering in polynomial-time, when the dimension q if fixed.

The following fact is straightforward:

Fact 4.2.5 *Any polynomial time algorithm for the k-Means Problem, when k does not have any restriction (e.g. being constant, or having a time complexity $O(n^k)$), leads to a polynomial time algorithm for the l_2^2-*ConFL *Problem with the same factor.*

This algorithm can be achieved executing the k-Median algorithm for all values of k between 1 and $|C|$ and selecting the minimum with respect to the l_2^2-ConFL cost function.

The mentioned results of Kanungo, Netanyahu, Piatko, Silverman and Wu [32] and Jain and Vazirani [30] lead to the following theorems:

Theorem 4.2.6 *[32] There exists a $(9+\epsilon)$-approximation algorithm for l_2^2-*ConFL *with l_2^2 distance when ϵ and the space dimension q are constants.*

Theorem 4.2.7 *[30] There exists a (108)-approximation algorithm for l_2^2-*ConFL *with l_2^2 distance.*

4.3 Adapting an A_{JV} to ConFL

In [30], Jain and Vazirani presented a 3-approximation algorithm, which we denote by A_{JV}, to the FL problem with time complexity $O(m \log m)$ and a 6-approximation algorithm to the KM problem with time complexity $O(m \log m(L + \log n))$, where $n = |C| + |F|$, $m = |C| \cdot |F|$ and $L = \log \frac{c_{\max}}{c_{\min}}$, where $c_{\max}$ is the maximum cost and $c_{\min}$ is the minimum cost in the instance. Both approximation factors are proved for the metric case.

The A_{JV} algorithm consists of two phases: In *Phase 1*, it obtains a dual feasible solution and determines a set of temporarily open facilities. In *Phase 2*, the algorithm selects the facilities to open, from the temporarily open facilities.

Phase 1: Algorithm A_{JV} starts with all cities disconnected and all facilities closed. It uses the notion of time and each event occurs at some time. For each $j \in C_t$, the algorithm maintains a variable α_j that represents the contribution offered by city j at that time. Variable α_j can be seen as the radius of a ball centered at city j. Initially, the time and the values of α_j start with 0. At each iteration, the algorithm grows uniformly the radius α_j, for each unconnected city $j \in C$ together with the time variable. When $\alpha_j = c_{ij}$ for some edge (i, j), the algorithm assign edge (i, j) as tight and start to offer a contribution $\beta_{ij} = \alpha_j - c_{ij}$ to open facility i. When the total contribution offered to a facility i is equal to f_i, the algorithm temporarily open facility i and temporarily connects all cities j with a positive contribution β_{ij} to i. Furthermore, if j is unconnected and has a tight edge with i, it is temporarily connected to i. The values of α_j, for all unconnected cities j, are increased during *Phase 1* until there are no more unconnected cities. When a city j becomes temporarily connected, its contribution radius is fixed.

Phase 2: Let F_t be the set of temporarily open facilities obtained in *Phase 1*. The final set of open facilities is a maximal set $F_o \subseteq F_t$ such that, two facilities $i', i'' \in F_o$ do not have a city j with positive contribution to i' and i'', that is, with $\beta_{ij} > 0$ and $\beta_{i'j} > 0$.

For more details about algorithm A_{JV} and the proof of its approximation factor, see [30]. Our interest in this section is to describe how to implement $\hat{A}_{\text{JV}}$ in polynomial time in the size of the original instance. The approximation factor will simply be inherited from A_{JV} as stated in Corollary 4.2.4.

The straightforward implementation of algorithm $\hat{A}_{\text{JV}}$ is a non-polynomial-time algorithm to l_2^2-ConFL as the set $\Phi(C)$ may have exponential size and it produces solutions with the same factor that A_{JV} to the corresponding instance of FL. Jain and Vazirani have shown that A_{JV} is a 3-approximation to FL if the distance function is metric and a 9-approximation in the l_2^2 space. In Figure 4.1 we provide an example where l_2^2 does not respect triangle inequality property. This example shows that l_2^2 is not a metric space.

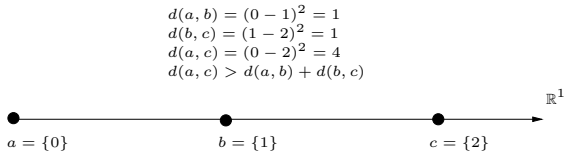


Figure 4.1: Example showing that l_2^2 is not metric.

Now, we describe how to implement $\hat{A}_{\text{JV}}$ in polynomial-time with a small error ratio ϵ when the dimension q is bounded by a constant. The main idea is the fact that under certain conditions, we have to consider only a polynomial number of facilities.

Since the algorithm works in $\mathbb{R}^q$, the contribution of each city j defines a q-dimensional sphere of radius α_j. When these spheres increase, as α_j increase, they start to intersect each other partitioning the space in regions. See Figure 4.2.

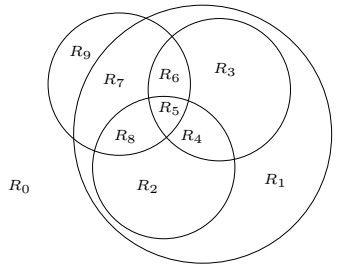


Figure 4.2: Examples of a space partitioned in intersection regions.

In some distance functions, a sphere defines a set of points with distance r to the center and may have different shapes, as a square or a rhombus. In Euclidean and

Squared Euclidean, the shape is the same and it is easily handled, for example, to calculate intersections among spheres or among a sphere and planes or lines. Taking a sphere as an example, if we are handling spheres in the space with Squared Euclidean distance function, a sphere of radius 4 and center at the origin pass through $(2, 0)$. This means that an implementation must care and also work with the square root of the radius in order to discover the limits of the sphere.

During the execution of algorithm, an unconnected city j increase its radius α_j and any point p receives a contribution $\max\{\alpha_j - d(j,p), 0\}$ from j. An *intersection region* R, generated by a set of cities S at a given time, is the set of points in $\mathbb{R}^q$ that receives positive contributions from the cities in S and no contribution of cities in $C \setminus S$. We denote the set of cities contributing to R by $cit(R)$. As cities contribution boundaries change during the execution of the algorithm, the intersection regions are not static. The region can appears, it changes as the boundaries increases and it can even disappear, when it is completely covered by other boundaries. The following facts are valid.

Fact 4.3.1 *If R is an intersection region then the point $center(cit(R))$ receives the maximum contribution from $cit(R)$ compared to other points in R.*

Proof. The total contribution received by a point $x \in R$ from the points of $cit(R)$ is given by $\sum_{j \in cit(R)}(\alpha_j - d(j,x))$. Therefore, the contribution is maximized when $\sum_{j \in cit(R)} d(j,x)$ is minimum. □

Assuming different values for radius, and a given intersection region R, it is possible that $center(cit(R)) \notin R$. An example is showed in Figure 4.3. Note that the center of two points in l_2^2-distance function is the mid point in the line connecting them.

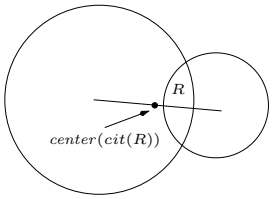


Figure 4.3: Examples of $center(cit(R)) \notin R$.

At each moment, we denote by $contrib(p)$, the total contribution received by p from all cities in C, for any point $p \in \mathbb{R}^q$. Note that A_{JV} guarantees that $contrib(p) \leq f$, and when $contrib(p)$ reaches f it is (temporarily) opened and the radius of all cities contributing to p stop increasing. If the contribution radius of an unconnected city j touches a point p having $contrib(p) = f$, then j is connected to p and its radius stop increasing.

The next lemma shows that Algorithm $\hat{A}_{\text{JV}}$ must only consider centers of regions.

Lemma 4.3.2 *If a facility is (temporarily) opened by algorithm $\hat{A}_{\mathrm{JV}}$ then it is a center of an intersection region at some time.*

Proof. Clearly, any temporarily opened facility i must have non-null contribution and must belong to some intersection region R. The total contribution received by i is exactly f. This is also the maximum contribution obtained by a point in R. From Fact 4.3.1 i must be a center of R. □

In the algorithm A_{JV}, the contribution of each facility starts with zero and cities are connected to facilities during the execution of the algorithm. We propose an implementation of $\hat{A}_{\mathrm{JV}}$ that considers one facility per intersection region. This facility starts to be considered when the intersection region is created by a set of cities.

The following lemma says that the search for a candidate facility in a region can be restricted to the maximal set of cities that contribute to that region.

Lemma 4.3.3 *Consider the execution of the algorithm $\hat{A}_{\mathrm{JV}}$ at time T, where there exists an intersection region R. If, in this moment, a city $j \in C \backslash cit(R)$ contributes to $center(R)$, then $center(R)$ will not be eligible to be a facility, unless it becomes a center of a different region S with $center(S) = center(R)$.*

Proof. Given a moment T, if $p_R = center(R)$ receives a non-null contribution of a city $j \in C \backslash cit(R)$ then p_R must also belong to a region R' defined by the contribution radius of the cities in $cit(R) \cup j$. If we denote by $p'_R = center(R')$ we have that $contrib(center(p_R)) < contrib(center(p_R))$ and therefore, $center(p_R)$ cannot be an open facility. □

The following lemma give us an intuition for the number of regions. Although this result is already known, we present a proof here for completeness.

Lemma 4.3.4 *The number of regions generated by the intersection among n circles in $\mathbb{R}^2$ is bounded by $n(n-1)+2$.*

Proof. The proof is by induction. The case $n = 1$ is straightforward. Suppose that the number of regions for $n-1$ circles, is at most $(n-1)(n-2)+2$. If we add the last circle, it touches each other in, at most two points. The maximum number of intersections is $2(n-1)$. If we consider an arc between each two intersections in the last circle, it is divided in at most $2(n-1)$ arcs. Each arc generates exactly one new region, so the final number of regions is at most $(n-1)(n-2)+2+2(n-1) = n(n-1)+2$. □

Consider algorithm $\hat{A}_{\mathrm{JV}}$ applied in $\mathbb{R}^2$ and the regions defined by the $|C|$ circles with radius α_j, for each $j \in C$. The number of regions at each moment is $O(|C|^2)$ and we will show, in Corollary 4.3.11 for $q = 2$, that the total number of regions for all values of α during the execution of the algorithm is $O(|C|^3)$.

4.3.1 Implementation for l_2^2

We denote by n-sphere, a hypersphere of dimension n. Consider the following lemma:

Lemma 4.3.5 *The intersection between an n-sphere with a non-tangent hyperplane of dimension $n-1$ result in a $(n-1)$-sphere or a empty set.*

Proof. Consider an n-sphere of radius $A^2 \in \mathbb{R}^+$. Without loss of generality, the n-sphere can be centered at the origin and it can be represented by the following equation:

$$x_1^2 + \sum_{i=2}^{n} x_i^2 = A^2.$$

The hyperplane can be represented, without loss of generality, by

$$x_1 = K, \text{ for some } K \in \mathbb{R}^+, K \neq A.$$

The intersection between the n-sphere and the hyperplane is given by the following:

$$\{\mathbf{x} \mid x_1 = K, \sum_{i=2}^{n} x_i^2 = A^2 - K^2\},$$

that is a $(n-1)$-sphere or a empty set. □

We also need the following lemma:

Lemma 4.3.6 *The intersection between two non-concentric and non-tangent n-spheres can be represented as the intersection of an n-sphere with a non-tangent hyperplane of dimension $n-1$.*

Proof. Consider, without loss of generality, an n-sphere $\phi(a)$ of radius $A^2 \in \mathbb{R}^+$ centered at the origin and an n-sphere $\phi(b)$ of radius $B^2 \in \mathbb{R}^+$ centered at $(C, 0, \ldots, 0)$, such as $C \neq A + B$ and $C \neq A - B$.

The sphere $\phi(a)$ can be represented as

$$x_1^2 + \sum_{i=2}^{n} x_i^2 = A^2.$$

and the sphere $\phi(b)$ can be represented as

$$(x_1 - C)^2 + \sum_{i=2}^{n} x_i^2 = B^2.$$

The set of points in the intersection will be

$$\{\mathbf{x} \mid \mathbf{x} \in \phi(a), \text{ and } x_1 = (A^2 - B^2 + C^2)/(2C).\}$$

where the hyperplane cannot be tangent to $\phi(a)$ because $C \neq A + B$ and $C \neq A - B$ □

The following fact is known:

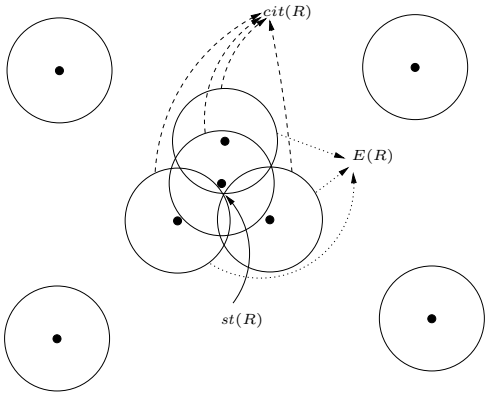


Figure 4.4: Example of $E(R)$ and $cit(R)$.

Fact 4.3.7 *In a $\mathbb{R}^q$ space, the intersection among m linear independent half-space hyperplanes has dimension $q - m$.*

Consider the space $\mathbb{R}^q$ and an intersection region $R \subset \mathbb{R}^q$. We denote by $st(R)$ the first point of R that is covered by all spheres of $cit(R)$ and denote by active sphere associated to an unconnected city. An active sphere at moment T has radius $\alpha_j = T$, while a non-active sphere has radius $\alpha_j < T$. We consider that the points in C suffered a very small perturbation so that no point $p \in \mathbb{R}^q$ belongs to more than $q + 1$ spheres in a given moment unless all spheres that contains p are not actives.

Given an intersection region R at a certain time T, let $st(R)$ and $t(R)$ be the initial time of the intersection region. Some cities $j \in C$ may have contribution radius greater than the distance $d(j, st(R))$ at time $t(R)$. So, let $E(R)$ be the set of cities such that $E(R) = \{j \in C \mid d(j, st(R)) = \alpha_j \text{ at time } t(R)\}$.

We denote by *tight cities* the set of cities in $E(R)$. This set is divided in two groups: the unconnected cities at time $t(R)$, denoted by $S(R)$, and the temporarily connected cities at time $t(R)$, denoted by $G(R)$. The starting point $st(R)$ belongs to the contribution boundary of cities in $E(R)$. It is the intersection of the spheres defined by cities in $G(R)$, which have fixed contribution, and the intersection of spheres in $S(R)$, which have radius $t(R)$. See Figure 4.4.

Denote by $init(R)$ the tuple $(st(R), t(R), S(R), G(R))$. For a given $E(R) = G(R) \cup S(R)$, it is possible to discover both $t(R)$ and $st(R)$ using elementary geometrical operations.

The point $st(R)$ belongs to the intersection of the spheres in $G(R)$ and is equidistant from all points of $S(R)$. For any two points $s_1, s_2 \in S(R)$, the set of points equidistant from s_1 and s_2 is the half-space hyperplane orthogonal to s_1s_2-line that contains the point $(s_1 + s_2)/2$. Thus, the point $st(R)$ belongs to the intersection of half-spaces

hyperplanes defined for all pair of distinct $s_1, s_2 \in S$.

Denote by $h(r, s)$ the hyperplane $\{x : d(x, r) = d(x, s)\}$ for $r, s \in \mathbb{R}^q$. Let $h(S(R)) = \bigcap_{\forall r,s \in S(R)} h(r, s)$. That is, if $x \in h(S(R))$ then $d(x, j) = d(x, S(R))$ for any $j \in S(R)$. Note that $h(\{s\}) = \mathbb{R}^q$. Let $\phi(j) = \{x \mid d(x, j) = \alpha_j\}$ and $\phi(G(R)) = \bigcap_{j \in G(R)} \phi(j)$. Observe that $\phi(\emptyset) = \mathbb{R}^q$.

The following lemma shows that all regions can be associated with a set of at most $q+1$ cities.

Lemma 4.3.8 *With a small perturbation in the points of C, the starting point $st(R)$ of any region R is defined by at most $q + 1$ cities, or simply $|E(R)| \leq q + 1$.*

Proof.

Consider the case when $G(R) = \emptyset$. The point $st(R)$ is equidistant to each city in $S(R)$. Thus, for an arbitrary city $s_1 \in S(R)$ and all cities $s' \in S(R) \setminus \{s_1\}$, the starting point belongs to each linearly independent half-space hyperplane $h(s_1, s')$. As stated in Fact 4.3.7, the dimension of the intersection of m linearly independent half-space hyperplanes in a $\mathbb{R}^q$ space is $q - m$. If $|S(R)|$ is $q + 1$, the $h(S(R))$ has dimension zero and it is single point. If $S(R)$ has more than $q + 1$ elements in a $\mathbb{R}^q$ domain, no point in $\mathbb{R}^q$ can be equidistant for all elements in $S(R)$.

Now, consider the case when $G(R) \neq \emptyset$. We assume $S(R) \geq 1$ because if all cities in $E(R)$ are fixed, the starting point does not exist.

Consider an arbitrary sphere $\phi(s_1) \in G(R)$. The starting point belongs to the intersection of $\phi(s_1)$ and $\phi(s')$ for all $\phi(s')$ in $G(R) \setminus \{s_1\}$. Using Lemma 4.3.6, $\phi(G)$ can be calculated as the intersection of $\phi(s_1)$ with $|G(R)| - 1$ half-space linearly independent hyperplanes. The starting point need also to be equidistant to all points in $S(R)$, that is the domain $\mathbb{R}^q$ restricted the intersection of $|S(R)| - 1$ linearly independent hyperplanes. Thus, the starting point belongs to the intersection of an n-sphere with $|E(R)| - 2$ linearly independent half-space hyperplanes. In this case it is possible to recursively apply Lemma 4.3.5. If $|E(R)| = q + 1$, the starting of a given intersection region belongs to a 1-sphere or is an empty set. A 1-sphere consists of two points. If $E(R) > q + 1$ there is no point in $\mathbb{R}^q$ that satisfies both the $\phi(G(R))$ and $h(S(R))$ constraints.

□

Given a region R, if $E(R)$ has $q + 1$ points and $G(R) = \emptyset$, the starting point is well defined. If $G(R) \neq \emptyset$, there are at most two candidates for the starting point $st(R)$ and both will be considered by the algorithm. Remember that our issue is to reduce the exponential size set of candidates $\Phi(C)$ removing ineligible facilities. The presence or absence of any point in $\mathbb{R}^q$ is irrelevant unless for the eligible subset, that must be considered.

If $E(R)$ has less than $q+1$ elements, the intersection $h(S(R)) \cap \phi(G(R))$ does not define zero, one or two single points because there is a degree of freedom. In this case we consider an additional constraint.

Given a set $P \subset \mathbb{R}^q$, $g(P)$ is defined as the affine combination of points in P. That is, $g(\{p\})$ is the single point p, $g(\{p_1, p_2\})$ is the infinite line that contains points p_1 and p_2, $g(\{p_1, p_2, p_3\})$ is the infinite plane that contains points p_1, p_2 and p_3, and so on.

Lemma 4.3.9 *The starting point $st(R)$ belongs to the affine combination of the cities in $E(R)$, denoted by $g(E(R))$.*

Proof. Consider, for the sake of contradiction, that $st(R) \notin g(E(R))$. For the orthogonal projection of $st(R)$ in $g(E(R))$, the distance of each city to the orthogonal projection will be smaller than the distance from $st(R)$, so the orthogonal projection of $st(R)$ will belong to R before $st(R)$, which is a contradiction. □

Consider a region R such that $|E(R)| = q'$ and $q' < q+1$. The domain can be restricted from the q-dimensional space to $g(E(R))$ that has dimension $q' - 1$. In this domain, the intersection $h(S(R)) \cap \phi(G(R))$ has zero degree of freedom, and the starting point will be restricted to zero, one or two possible points.

When the algorithm starts, all cities are unconnected. This means that $G(R)$ is empty for all regions. It is possible to define all *init*'s as a set of tuples $\{(st(R), t(R), S(R), G(R))$ for each region $R\}$. Note that $cit(R)$ is unknown in this moment. It will be known *on the fly*, or during the execution of the algorithm. So we prefer to name as E, S and G the sets $E(R)$, $S(R)$ and $G(R)$, respectively, when the region is not yet defined.

We start the algorithm with a set of tuples for all possible starting points at moment $T = 0$, which is

$$\mathcal{T}(C) \leftarrow \{(p, t, S, G) \mid G = \emptyset, \forall S \subseteq C \text{ and } |S| \leq q+1, \\ p = h(S) \cap g(S), t = d(p, S)\}. \tag{4.1}$$

The construction of $\mathcal{T}(C)$ consists to define a new starting point candidate for each subset with $q+1$ or less cities. The size of $\mathcal{T}(C)$ is exponential in the dimension, but, for a fixed dimension q, it has size $|C|^{q+1}$, which is the value to combine $|C|$ cities in sets of size 1, 2, up to $q+1$.

In this moment, the algorithm starts to increase the radius around each city in C. The implementation consists in an event based algorithm. For each tuple in $\mathcal{T}(C)$, there is an associated event at time t, that is the time a new region is created.

When the algorithm starts at time $T = 0$, a set of events in $\mathcal{T}(C)$ are ready to be processed. The events consist in a new region R for each S with just one city, which happens at time $t = 0$. In order to process such event, the algorithm discovers which

cities belong to $cit(R)$, and then create a new event, which is a new facility positioned in $center(cit(R))$. Lemma 4.2.1 shows how to calculate the center of a set of points. In the general case, the cities in $cit(R)$ are able to contain other cities than $E(R)$ and may not contain some cities in $E(R)$. Remember that $E(R)$ are tight cities, with zero contribution to $st(R)$ at the moment t. The set $cit(R)$ contains also all cities with a strict positive contribution to $st(R)$ in moment t. Figure 4.4 shows an example of $E(R) \subsetneq cit(R)$. It is possible to have cities in $E(R)$ that do not belong to $cit(R)$. See Figure 4.6 for examples. In the end of this section we will show an algorithm to find which cities of $E(R)$ belong to $cit(R)$.

The event of a new facility being considered is associated with the time when this facility is totally paid. There is a third set of events that consists in an unconnected city that reaches a temporarily open facility. The time associated with this event, for a city j and a temporarily open facility i is $T = d(i, j)$.

The algorithm contains three types of events, represented by the sets CtF, OpenF and NewF. The set CtF has the events of connecting a city to a temporarily open facility. The set OpenF has the events of temporarily opening a facility and connecting its contributor cities to it; and the set NewF has the events of creating a new region and adding a new facility to be considered. The algorithm is described in Figure 4.5. The event NewF is designed to keep the *init* tuples (p, t, S, G).

When G is an empty set, there is just one candidate to be the initial points, that is p, calculated as $g(S) \cap h(S)$. When G is not empty, there are zero or two candidates to be the initial point, because $g(E) \cap h(S) \cap \phi(G)$ is a 1-sphere or a empty set.

When a city j is connected, the tuple (p, t, S, G) where $j \in S$ is changed. The new S does not contains j, which now belongs to G. The initial point must be recalculated, using elementary geometric operation of intersection of hyperplanes and intersection of spheres in order to result in a new empty or two starting point candidates. For example, if G contains two fixed q-spheres $\phi(a)$ and $\phi(b)$, which have distance of their centers greater than $\alpha_a + \alpha_b$, they have no intersection. A region associated with this tuple will never happen. There is no candidates to be starting points and the time associated with this event algorithm is assigned as infinity. The other case, in which the intersection will result in two points, we will update the tuple (p, t, S, G) where $j \in S$ to a new tuple, where j is moved from S to G. The initial point will be changed to a set of size 2, $\{p_1, p_2\}$, and the time associated with the event will be changed to a set of size 2, $\{t_1, t_2\}$. The implementation maintains this single tuple working as two possible events, one at time t_1 and other at time t_2. We define an operation on set NewF that is $\min\{\text{NewF}, T\}$, which returns a not used event with minimum time greater than or equal to T. The set U is the set of unconnected cities and T is a notion of time associated with the algorithm.

PHASE 1 OF $\hat{A}_{\mathrm{JV}}$

Input: An instance (C, q, l_2^2, f) to l_2^2-ConFL

Output: A set of temporarily open facilities.

1) $U \leftarrow C$, CTF $\leftarrow \emptyset$, OPENF $\leftarrow \emptyset$, (i) NEWF $\leftarrow \mathcal{T}(C)$, $T \leftarrow 0$.

2) Set all events in NEWF as not used.

3) While $U \neq \emptyset$:

   3.1) (ii) $evt \leftarrow \min\{\min\{\text{CTF}\}, \min\{\text{OPENF}\}, \min\{\text{NEWF}, T\}\}$.

   3.2) $T \leftarrow t$, for $t \in evt$.

   3.3) If $evt = (t, i, j) \in$ CTF
   (iii) delete evt from CTF,
   if j is unconnected,
   temporarily connect j to i, $U \leftarrow U \setminus \{j\}$,
   fix the contribution radius of j as t.
   (iv) Update OPENF and (v) NEWF.

   3.4) If $evt = (t, i) \in$ OPENF,
   (iii) delete evt from OPENF,
   temporarily open facility i,
   for each unconnected city j with a positive contribution to i
   temporarily connect j to i, $U \leftarrow U \setminus \{j\}$ and
   fix the contribution radius of j as t.
   (iv) update OPENF and (v) update NEWF.
   Add $(d(i, j), i, j)$ to CTF for each $j \in U$.

   3.5) If $evt = (p, t, S, G) \in$ NEWF,
   (iii) set evt as used,
   (vi) a new facility i is created and added to OPENF.

4) Return the set of temporarily open facilities.

Figure 4.5: Implementation details of Phase 1 of $\hat{A}_{\mathrm{JV}}$.

We have some details about the algorithm described in Figure 4.5. When an event is marked as used, the algorithm assign it as used and does not delete from NewF. If necessary, when one or more cities in S became connected, the time associated with the event changes. This time can increase or decrease with respect to the time of the original event. If the time increases and the final time is greater than or equal to T, this new event will be able to generate a new region. So, the tuple is updated and marked as not used in NewF. An initial tuple with S and G can be updated just a few times, because S starts with at most $q+1$ cities, and, at each update, the size of S decreases by 1. The algorithm does not start with all starting points, because it is not known the sequence the cities are connected to a temporarily open facility. So, the algorithm starts with a set of possible events, and, every time one or more cities are connected, the algorithm moves the connected cities from S to G. The starting point needs to be recalculated, using elementary operations of intersection of hyperplanes and intersection of q-spheres with hyperplanes. One can consider it as a hard operation to recalculate NewF every time a city is connected, but this operation will be used at most once by city in C. When a city j is connected, the algorithm must update all tuples in NewF where $j \in S$, moving j from S to G, recalculate the starting point and the event time, and assign these events as not used.

When a city is connected, the algorithm updates the event set OpenF, which is the event associated with facilities. Each facility has its position and the time it will be totally paid. If the algorithm reaches the time when the facility is totally paid, the algorithm temporarily open this facility an connects the facility contributors to it. The cost to open a facility i is a constant value f. The time this facility will be opened can be calculated as $\sum_{j \in C} \max\{\alpha_j - d(i,j), 0\} = f$. If the facility i is associated with a region R, this value is $\sum_{j \in cit(R)} \alpha_j - d(i,j)$. If a new city $j' \notin cit(R)$ starts to contribute to $center(cit(R))$ this facility is ineligible, as stated in Lemma 4.3.3.

Suppose an event in OpenF of opening a facility i where some cities contribute to it. The cities are $cit(R)$ where i is created as a center region R. We have a set S' of unconnected contributors to i and a set G' of connected contributors to i. The time in which this facility is totally paid is the value T such that $\sum_{j \in S'}(T - d(i,j)) + \sum_{j \in G'}(\alpha_j - d(i,j)) = f$. When a city j is connected to some facility, all other facilities in OpenF where $j \in S'$ must be updated. This consists to move j from S' to G' and recalculate its event time. In the events associated with temporarily open a facility, the event time never decrease when updated, because the city, when it is connected, stops increasing its contribution boundary.

One important assumption of the algorithm described in Figure 4.5 is that one event in NewF is associated with at most one region. This is proved in Lemma 4.3.10.

Lemma 4.3.10 *One event (p, t, S, G) is associated with at most one region, under the small perturbation assumption.*

Proof.

Consider an event defined by a time t and with starting point p. The problem happens if two regions, named as R_1 and R_2, or more than two, start at a same time t and at a same place p. This situation does not occur under the small perturbation assumption. Consider, for the sake of contradiction, two distinct regions R_1 and R_2 starting at a same point p. The point p is $h(S) \cap \phi(G)$. There are one contribution boundary associated with one specific city, say $j' \in C$, that divides R_1 and R_2. If we remove j', the region defined by $cit(R_1) \cup cit(R_2)$ will start in point p in time t and is calculated without j' influence. Thus, as j' has a perturbation, its contribution boundary in moment t does not contain p, that is a contradiction.

A similar argument can be applied to three or more regions, showing that any region has a different starting point. One event cannot be associated with two regions because it has only one starting point p.

□

Considering that any region can be associated to its initial point, we have:

Corollary 4.3.11 *Under the small perturbation assumption, the number of regions considered by algorithm $\hat{A}_{\text{JV}}$ is bounded by $O(q|C|^{q+1})$.*

Finally, we have to show how the algorithm discovers the set $cit(R)$ in a considered event (p, t, S, G). The cities $j \in C$ such that $\alpha_j > d(p, j)$ belong to $cit(R)$. The problem happens with cities in $S(R)$ and $G(R)$ for which $\alpha_j = d(p, j)$. They may or may not belong to $cit(R)$. In Figure 4.6 there is a series of events where this issue appears.

Giving an event (p, t, S, G), the algorithm has to discover $cit(R)$. Let Q be the set of cities with strictly positive contribution to p and $E = S \cup G$. As our concern is to limit the number of facilities to be considered as a polynomial number in the size of the instance, the algorithm avoids a large number of facilities that are ineligible. On the other hand, if we add some ineligible facilities, the algorithm will work correctly, but in some sense slower. One possible approach is to consider a new facility in position $p = center(Q \cup A)$ for all $A \subseteq E$. Certainly, one of these sets of cities will be the set which corresponds to the created region. The others sets will generate facilities that do not interfere in the result. The number of facilities considered in this approach is exponential in the size of E. As size of E is limited in $q + 1$, the number of facilities considered will increase by a factor 2^{q+1}. The algorithm, as stated before, has a polynomial time complexity for a constant value of q, so these extra facilities do not make the algorithm an exponential one.

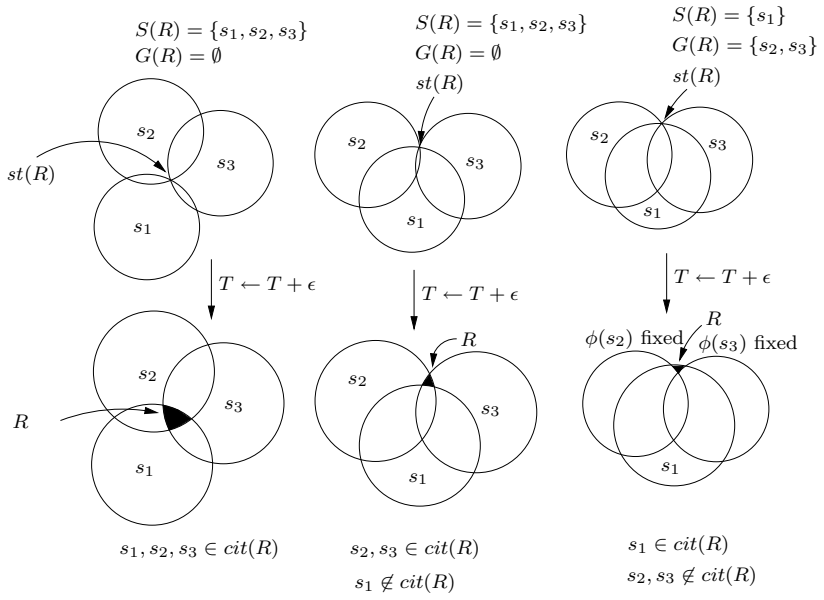


Figure 4.6: How to discover which cities belong to $cit(R)$.

We believe that some techniques of continuous calculus can be used to discover the correct subset of E that belong to $cit(R)$. In the experiments of Section 4.5, we use the following approach. The experiments works in space $\mathbb{R}^q$ for $q = 2$. So, the set of tight cities E can have size one, two or three. If the set has size one or two, all cities of E belong to $cit(R)$. If the size of E is three, we use a numerical approach that consists to increase the time variable T with a relative very small value and check which contribution boundaries involves the region.

The Phase 2 of algorithm A_{JV} consists to find a maximal independent set of temporarily open facilities and return it as the final result. In the considered graph, the vertex set is the set of temporarily open facilities and there is an edge between two facilities, say i and i', if and only if there is at least one city with strictly positive contribution to i and i' for any pair of (i, i') of temporarily open facilities.

Given $S(R)$ and $G(R)$, let β_q be the complexity time to discover $st(R)$, that is the complexity time to calculate intersection $h(S(R))$ with $\phi(G(R))$ and $g(E(R))$, and let $\theta_{C,q} = q|C|^{q+1}$. The final time complexity of the implementation of $\hat{A}_{\mathrm{JV}}$ proposed in Figure 4.5 is $O(q^2|C|^{q+2}\beta_q)$, assuming the numerical approach to discover cities in $cit(R)$.

In order to achieve this time complexity, it is necessary to summing up various branches of the algorithm. The time to create the initial set of events $\mathcal{T}(C)$ is the time to create all possible combinations of $S \subseteq C$ with size less than or equal to $q + 1$. For each of these $O(|C|^{q+1})$ subsets, it is necessary to calculate the starting point p, which takes $O(\beta_q)$

and then calculate the distance $d(p, s_x)$ for some $s_x \in S$ that can be done in $O(q)$. The final complexity $O(|C|^{q+1}(\beta_q + q))$ that is the same that (i) $O(|C|^{q+1}\beta_q)$.

The maximum number of temporarily open facilities in the algorithm is $|C|$, because each temporarily open facility is responsible for removing at least one city from U. Thus, the maximum size of set CtF is $|C|^2$, which is an edge between each unconnected city with a temporarily open facility.

The maximum size of OpenF corresponds to the maximum number of regions considered. As each region can be associated with a distinct initial point, and each initial point is one of the $|C|^{q+1}$ tuples of $\mathcal{T}(C)$ in its original state, of after 1 to q updates, the size of OpenF has order $O(\theta_{C,q})$.

Let CtF, OpenF be minimum heaps and NewF a special structure that save not used events with time greater than the current T in a minimum heap. In this way, it is possible to discover the minimum of these three sets in constant time. After discovering the minimum, the algorithm delete one element of CtF $\cup$ OpenF or set a non-used event as a used event in NewF (iii). The maximum number of times that the algorithm executes $evt \leftarrow \min\{\min\{\text{CtF}\}, \min\{\text{OpenF}\}\min\{\text{NewF}, T\}\}$ is $O(\theta_{C,q} + |C|^2 + \theta_{C,q})$, which is (ii) $O(\theta_{C,q})$.

Deleting an element of a heap with size n can be performed in time $O(\log(n))$, so the execution of (iii) is bounded by $O(\theta_{C,q} \log \theta_{C,q})$.

Every time one or more cities become connected, it is necessary to update OpenF and to update NewF.

The algorithm visits all facilities in OpenF and recalculate its event time. Suppose we can calculate the new event time in $O(q)$. The branch that update a set OpenF takes $O(\theta_{C,q}q)$ to update and $O(\theta_{C,q})$ to reconstruct the heap property.

Now it will be showed how to update the event time in $O(q)$ when a city become connected. Consider a facility that has many cities contributing to it, divided in unconnected, named as S', and temporarily connected, named as G'. A naive algorithm to update the time in which this facility is totally paid consists in isolating T in equation $\sum_{j \in S'}(T - d(i,j)) + \sum_{j \in G'}(\alpha_j - d(i,j)) = f$, which has complexity $O(|S' \cup G'|q)$. However, there is a more efficient approach. Suppose we have already calculated the $p = \mathit{center}(\mathit{cit}(R))$. The point p does not change during the execution of the algorithm, because Lemma 4.2.1 do not care if cities are fixed or not fixed. Also, suppose that we have a previous time t when the facility is totally paid just before a city j is connected. All contributions will be the same, except j contribution, which changes from t to α_j at the event moment. So, the difference $t - \alpha_j$ must be supplied by all other unconnected cities. The new event time can be calculated in $O(q)$, as the previous event time, plus $(t - \alpha_j)$ divided by the number of remaining unconnected cities $|S' \setminus \{j\}|$.

With an $O(q)$ algorithm to update the event time in OpenF, each update of all events takes the mentioned $O(\theta_{C,q}q)$. The update is executed every time a city becomes temporarily connected, so it can be called at most $|C|$ times. Thus, the final update OpenF complexity during the algorithm is (iv) $O(|C|\theta_{C,q}q)$.

The update of the NewF set is similar. Every time a city j becomes connected, the update procedure iterate by all $O(|C|^{q+1})$ elements of NewF and recalculate its initial point, which takes $O(\beta_q|C|^{q+1})$. If some event had changed it S or R sets, it is assigned as not used. Then the algorithm recreate the internal heap property to the not used elements with time t greater than or equal the current algorithm time T. The complexity of create a heap property will be $O(|C|^{q+1})$. Remembering that the update is done at most once by new temporarily connected city, the overall time complexity of update NewF during the algorithm is (v) $O(|C||C|^{q+1}\beta_q)$.

The final branch of the algorithm is Line (v). In this line the algorithm create $O(\theta_{C,q})$ new facilities. To create one facility event, the algorithm discovers which cities belong to $cit(R)$, in $O(\gamma_C)$ time complexity , then it calculates the $center(R)$, in $O(q|C|)$, and it adds the event to OpenF in $O(\log\theta_{C,q})$. This branch takes (vi) $O(\theta_{C,q}(\gamma_C+q|C|+\log\theta_{C,q}))$

The final algorithm complexity time for the presented implementation is the sum of (i), (ii), (iii), (iv), (v), (vi) complexities, which is

$$O(|C|^{q+1}(|C|\beta_q + |C|q^2 + q\gamma_C + q^2\log q|C|)).$$

Where β_q is the time to calculate a starting point of a region and γ_C is the time to discover which cities belong to a region R.

4.3.2 Approximation Factors

We can adapt algorithm $\hat{A}_{\text{JV}}$ to the KM Problem. If we set the cost of opening a facility to a small value, the number of facilities in the solution will be large. Otherwise, a large value of f will force $\hat{A}_{\text{JV}}$ to open only one facility. With this idea, it is possible to use (in practice) a binary search over $\hat{A}_{\text{JV}}$ to open k facilities, although it may not converge to k. Denote this algorithm by A^*_{JV}. So, Jain and Vazirani [30] described a randomized rounding between the solution with $k_1 > k$ and the solution with $k_2 < k$ to obtain a solution that opens exactly k facilities.

Considering l_2^2 distance function, A_{JV} is a 9-approximation to FL Problem [30]. Thus, by Proposition 4.2.2, $\hat{A}_{\text{JV}}$ is a $(9+\epsilon)$-approximation algorithm to ConFL.

When the binary search performed in algorithm A^*_{JV} converges to k centers, the obtained solution has a deviation factor of at most $(9+\epsilon)$. When the number of centers does not converge to k, the randomized rounding step increases the approximation bound by a factor of 6, as shown in Section 6 of [30]. Thus $\hat{A}_{\text{JV}}$ is a $(54+\epsilon)$-approximation algorithm when the the dimension is constant.

4.4 Adapting A_{JMMSV} to ConFL

Jain, Mahdian and Saberi [29] and Jain, Mahdian, Markakis, Saberi and Vazirani [28] presented a 1.861-approximation algorithm for the FL problem with time complexity $O(m \log m)$, where $m = |C| \cdot |F|$. This algorithm, named here A_{JMMSV} is similar to A_{JV} and the intersection regions technique described in Section 4.3 can also be applied to describe a polynomial-time algorithm $\hat{A}_{\text{JMMSV}}$ to l_2^2-ConFL.

Algorithm A_{JMMSV} has only one phase, similar to Phase 1 of A_{JV}, and once a facility is open, it remains open until the end of the execution. This algorithm considers only two events: *(a)* if some unconnected city j has $\alpha_j = c_{ij}$ for an open facility i, then city j is connected to i; *(b)* if the contributions of unconnected cities U to a closed facility i reaches f (i.e., $\sum_{j \in U} \max(0, \alpha_j - c_{ij}) = f$) then facility i is opened and every city $j \in U$ with $\alpha_j \geq c_{ij}$ is connected to i and city j is removed from U.

When a city j is connected to an open facility, its contribution to other points becomes 0 and the set of events NewF is fixed in the beginning of the algorithm. Whenever an event $(p, t, S, G) \in$ NewF has a city already connected, it must be discarded and $\hat{A}_{\text{JMMSV}}$ does not need to maintain used events in NewF in opposition to the algorithm in Figure 4.5. When a city j becomes connected to a facility i, its contribution radius to other facilities is updated to zero. The set of facilities in OpenF need to be recalculated, removing the j-contribution.

For the distance function l_2^2 and q constant, this operation can be performed efficiently with time complexity $O(q)$. One event in OpenF is a tuple (t, i, J) where $i \in \mathbb{R}^q$ is the facility, t is the time when i is totally paid and J is the contributors to i. When a city j becames connected, the new updated event will have $J' = J - \{j\}$ contributors cities. The new facility position is given by $i' = (center(J)|J| - j)/(|J| - 1)$. The naive algorithm to calculate the time when i' is totally paid consists in iterate all cities $j' \in J'$ and update the time the event time with time t' such as $\sum_{j' \in J'}(t' - d(j', i')) = f$. If city j contributes for a facility x in time T, its contribution is given by a paraboloid $P_j(x, T) = T - d(j, x)$. The contribution of the cities in a set J is the sum of paraboloids, and can be represented by the polynomial

$$P_J(x, T) = T|J| - \sum_{i=1}^{q}(a_i x_i^2 + b_i x) + c$$

for a some constants a, b and c. The polynomial $P_{J'}(x, T)$ can be computed from $P_J(x, T)$ in $O(q)$ removing the contribution of the connected city. The new opening time t' is such that $P(i', t') = f$ where t' can be efficiently calculated in $O(q)$.

4.4.1 Extension of A_{JMMSV} to Euclidean Metric

For the l_2 metric, most of the analysis is still valid. We have the same description for a region, a starting point, etc. The starting points still have a set $E(R) = S(R) \cup G(G)$ of tight cities, where $E(R)$ have size less than or equal to $q + 1$. It will also leads to a polynomial number of regions, and each point in $center(cit(R))$ is a candidate to be a facility. The first difference to deal is that $center(cit(R))$ is not necessarily only one point. For example, in a region with $cit(R) = \{a, b\}$, any point in the straight segment between a an b will receive the maximum contribution of a and b.

The Fact 4.3.1 says the point $center(cit(R))$ receives the maximum contribution of the $cit(R)$. It needs to be restated to any point in $center(cit(R))$ receives the maximum contribution of cities in $cit(R)$.

The Lemma 4.3.2 shows that the only points eligible to be an open facilitiy by the algorithm is the center of a set of cities that define a region at some time. In Euclidean case, only points that belong to a center of a set of cities that define a region at some time is eligible to be an open facility.

Lemma 4.3.3 shows that a center becomes ineligible when a new contribution boundary start to contribute to it. This lemma needs to be generalized. When a point in $center(cit(R))$ receives a contribution of a new city $j \notin cit(R)$, all points in $center(cit(R))$ become ineligible. The proof is analogous to the one presented in Lemma 4.3.3, and therefore it is omitted.

The set $center(cit(R))$ may have more than one element. Each of them receives the maximum contribution from the cities, with respect to R. All points in $center(cit(R))$ will became totally paid and will be able to be an open facility at a same time. The algorithm A_{JMMSV}, choose one of them arbitrarily. Thus, it is sufficient to select any arbitrary point in $center(cit(R))$, so that the algorithm works in Euclidean Metric.

However, there are an additional difficult in calculate a center in Euclidean Space. In the Euclidean plane, the computation of a center of a set of points S is an expensive operation. This problem is known as Fermat-Weber Problem or Geometric Median Problem.

Consider the case when S has exactly 3 points. If one of the interior angles of the triangle defined by S has more than 120° the center is the point of S with that angle. If all angles are smaller than 120°, the geometric median is the point inside the triangle which leads to an angle of 120° to any pair of points in S. This is also known as the Fermat point of S.

If S has more than 3 points, the local search algorithm given by Weiszfeld [47] can be used to compute a center. This algorithm has efficient implementations [37] and has been largely investigated in the literature. There is also a $(1+\epsilon)$-approximation algorithm due to Xue and Ye [48] to compute the center of a set, with time complexity $O(\sqrt{nq}\log(c_{\max}/\epsilon)+$

$\log(nq)$) where n is the number of points, c_{max} the maximum cost and q is the space dimension.

Using Proposition 4.2.2, the proposed implementation of algorithm $\hat{A}_{\text{JMMSV}}$ is a $(1.861 + \epsilon)$-approximation algorithm to ConFL.

4.5 Computational Results

We implemented $\hat{A}_{\text{JMMSV}}$ and $\hat{A}_{\text{JV}}$ in the Java programming language. We compared the presented algorithms with an efficient program to the k-Means problem, developed by Kanungo, Netanyahu, Piatko, Silverman and Wu [31] and available in the Internet. This program, which we denote by A_{KMNPSW}, is also used to derive an algorithm to the l_2^2-ConFL problem. Program A_{KMNPSW} is implemented in *C++* and uses four algorithms to find a good clustering: The algorithm of Lloyd, a local search algorithm (based on swapping centers), a hybrid heuristic that iterates between Lloyd and local search and a more complex hybrid heuristic.

The adaptation of A_{KMNPSW} to the l_2^2-ConFL problem, which we denote by A^+_{KMNPSW}, consists of executing the algorithm A_{KMNPSW} for each possible number of centers and returning a solution with the minimum value. Which means, if S_k is the solution returned by A_{KMNPSW} using k centers, A^+_{KMNPSW} returns a solution with value $\min_{1\leq k\leq |C|}\{v(S_k) + k \cdot f\}$.

The algorithms were compared with two randomly generated sets of points, one with 50 and the other with 100 points, where each point is uniformly distributed in the square $[0, 1] \times [0, 1]$. For each set, we made a set of instances I_f for various values of f.

Although all tests were performed on the same computer, comparisons of the algorithms using execution time are not conclusive, because they differ in the programming language ($\hat{A}_{\text{JMMSV}}$ and $\hat{A}_{\text{JV}}$ are implemented in Java and A^+_{KMNPSW} is implemented in *C++*) and A_{KMNPSW} is used as a black box (it reads the instance each time it is executed). Alternatively, small execution times indicate the applicability of the presented algorithms.

The results are presented in tables 4.1 and 4.2. For each algorithm A_x in the set $\mathbb{A} = \{\hat{A}_{\text{JMMSV}}, \hat{A}_{\text{JV}}, A^+_{\text{KMNPSW}}\}$, we present its execution time in milliseconds, *t(ms)*, the ratio with the best obtained solution, *dev* (given by $A_x(I)/\min_{y\in\mathbb{A}} A_y(I)$), and the number of open facilities *#fac*. We report the time taken by algorithm A^+_{KMNPSW} separatedly, since it spends basically the same time for each instance of the same set.

The time to solve instances of size 50 with A^+_{KMNPSW} was 2.4 seconds, and it is independent of f. Algorithm A^+_{KMNPSW} was executed with the standard number of stages: 100. If we increase the number of stages, it produces better solutions, but its running time increases basically by the same factor. If we set the number of stages to 1,000, the

	$\hat{A}_{\text{JMMSV}}$			$\hat{A}_{\text{JV}}$			A^{+}_{KMNPSW}	
f	*t(ms)*	*dev*	*#fac*	*t(ms)*	*dev*	*#fac*	*dev*	*#fac*
2^{3}	2,654	1.00	1	2,076	1.02	1	1.00	1
2^{2}	1,259	1.12	1	1,784	1.08	1	1.00	2
2^{1}	641	1.04	2	974	1.31	1	1.00	2
2^{0}	392	1.08	3	661	1.32	2	1.00	3
2^{-1}	263	1.19	3	610	1.18	3	1.00	4
2^{-2}	201	1.19	6	502	1.46	3	1.00	6
2^{-4}	85	1.05	11	458	1.12	9	1.00	11
2^{-6}	68	1.00	22	414	1.10	15	1.03	21
2^{-8}	58	1.00	32	458	1.01	29	1.07	39
2^{-10}	79	1.00	41	595	1.00	41	1.06	47
2^{-11}	75	1.00	46	642	1.00	45	1.02	48
2^{-12}	77	1.00	48	741	1.00	48	1.00	48
2^{-13}	75	1.00	50	746	1.00	50	1.00	50

Table 4.1: Performance of $\hat{A}_{\text{JMMSV}}$, $\hat{A}_{\text{JV}}$ and A^{+}_{KMNPSW} for instances with 50 random points for l_2^2-ConFL.

worst result of A^{+}_{KMNPSW} has *dev* equals to 1.007 in 13.9 seconds. If we set the number of stages to 10,000, A_{KMNPSW} finds always the smallest, but in 126 seconds of execution.

	$\hat{A}_{\text{JMMSV}}$			$\hat{A}_{\text{JV}}$			A^{+}_{KMNPSW}	
f	*t(ms)*	*dev*	*#fac*	*t(ms)*	*dev*	*#fac*	*dev*	*#fac*
2^{3}	22,887	1.02	1	34,230	1.04	1	1.00	1
2^{0}	3,425	1.16	3	12,848	1.81	2	1.00	4
2^{-3}	1,036	1.05	12	6,774	1.71	6	1.00	13
2^{-6}	527	1.00	34	6,786	1.32	20	1.04	26
2^{-9}	478	1.00	66	8,359	1.02	64	1.11	68
2^{-12}	489	1.00	92	10,559	1.00	92	1.02	96
2^{-15}	482	1.00	98	12,440	1.00	98	1.00	99
2^{-18}	467	1.00	100	13,080	1.00	100	1.00	100

Table 4.2: Performance of $\hat{A}_{\text{JMMSV}}$, $\hat{A}_{\text{JV}}$ and for instances with 100 random points for l_2^2-ConFL.

For the set with 100 points, algorithm A^{+}_{KMNPSW} was also executed with 100 stages (default), presenting a maximum *dev* of 1.11. If we set the number of stages to 1,000, the worst result have *dev* 1.02 in 73 seconds. If we set the number of stages to 10,000, A_{KMNPSW} always return the minimum solution, but in 727 seconds of execution.

We also present computational experiments for the k-Means problem. We consider the same two randomly generated set of points (with 50 and 100 points) with different

values of k (number of centers). The algorithms $\hat{A}_{\text{JMMSV}}$ and $\hat{A}_{\text{JV}}$ were adapted to the k-Means problem applying a binary search in the value of f to obtain the correct number of centers. We denote the adapted algorithms as A^*_{JMMSV} and A^*_{JV}. The obtained results are presented in Table 4.3. For the algorithm A_{KMNPSW}, we also present the *dev* column for different number of stages: 10^2, 10^3 and 10^4.

The CPU time to solve A_{KMNPSW} with 100 stages was 40 milliseconds on average with a maximum of 66 milliseconds. With 1,000 stages, the average time was 266 milliseconds with a maximum of 469 milliseconds. With 10,000 stages, the average CPU time was 2.5 seconds with a maximum CPU time of 4.6 seconds.

We can also observe that A^*_{JMMSV} and A^*_{JV} found better solutions for large values of k. For instance of size 44, we can observe that A_{KMNPSW} is 2.9 times worse than A^*_{JMMSV} with 100 stages. For instance of size 47, A_{KMNPSW} is 1.53 times worse than A^*_{JMMSV} with 1,000 stages. With 10,000 stages, A_{KMNPSW} always return the smallest solution.

	A^*_{JMMSV}		A^*_{JV}		A_{KMNPSW}		
k	*t(ms)*	*dev*	*t(ms)*	*dev*	*dev* 10^2	*dev* 10^3	*dev* 10^4
1	5,152	1.15	3,879	1.57	1.00	1.00	1.00
10	409	1.14	4,319	1.29	1.00	1.00	1.00
20	438	1.11	3,136	1.61	1.24	1.01	1.00
30	473	1.06	6,762	1.31	1.66	1.07	1.00
40	463	1.00	3,141	1.00	2.78	1.12	1.00
44	541	1.11	2,131	1.00	3.27	1.48	1.00
47	306	1.00	6,711	1.00	1.80	1.53	1.00
50	59	1.00	695	1.00	1.00	1.00	1.00

Table 4.3: Performance of A^*_{JMMSV}, A^*_{JV} and A_{KMNPSW} for instances with 50 random points for k-Means.

We made the same experiments with instances of size 100. Table 4.4 shows that the quality of A_{KMNPSW} can be far from the optimum using 100 stages. We have that A^*_{JMMSV} and A^*_{JV} are stable in solution quality, with maximum *dev* equals to 1.29 and 1.48.

	A^*_{JMMSV}	A^*_{JV}	A_{KMNPSW}		
# stages			10^2	10^3	10^4
max(dev)	1.30	1.48	10.24	4.01	1.02
avg(time) s	5.7	56	0.11	0.7	7

Table 4.4: Performance of A^*_{JMMSV}, A^*_{JV} and A_{KMNPSW} for instances with 100 random points for k-Means.

4.6 Future Works

As future works we pointed that the proposed technique could be proved to be more general. If yes, it would become a theoretical tool to convert graph to space problems keeping polynomiality.

4.7 Conclusion

In this paper, we consider a new problem, denoted by ConFL, and we propose good approximation algorithms for Euclidean and squared Euclidean distance functions. We also propose to solve the d-Clustering problem via the ConFL problem. Implementations of the proposed algorithms for l_2^2-ConFL and d-Clustering with distance function l_2^2 shows that they have good practical performance. We also compared these algorithms for the l_2^2-Clustering problem with a hybrid algorithm (that takes the best solution among 4 algorithms) and two presented algorithms. In many cases, the presented algorithms obtained better solutions than the hybrid algorithm, suggesting that they are good candidates to be incorporated in the hybrid strategy.

Capítulo 5

Conclusões

No estudo desenvolvido neste doutorado houve um equilíbrio entre teoria e aplicabilidade. Em todos os trabalhos houve uma seção de resultados experimentais, onde os algoritmos foram analisados empiricamente. Baseados nos estudos realizados, podemos abordar uma questão relevante, a métrica de qualidade para algoritmos.

Um fator de aproximação próximo de 1 é comumente visto como o mais desejável. Críticos costumam questionar se um algoritmo 10-aproximado teria alguma utilidade. Quem se interessaria por uma solução dez vezes pior que a ótima? Qual a relação entre fator de aproximação e desempenho prático? Fatores referentes a complexidade de tempo podem pesar mais na hora se optar por um algoritmo?

Um fator de aproximação próximo a 1 garante a qualidade da solução, porém um fator de aproximação mais alto não implica diretamente em soluções de baixa qualidade. Ao se restringir o domínio a uma classe de instâncias de interesse prático, um algoritmo $(\log n)$-aproximado pode ser mais rápido, mais simples de ser implementado e devolver soluções de melhor qualidade que um algoritmo 4-aproximado. O estudo desenvolvido no Capítulo 2 aproximou-se disso, ao conseguir soluções com desvio máximo de 1,27% em uma quantidade de tempo 3 ordens de grandeza menor que os desenvolvidos sobre um resolvedor profissional de programação linear para o primeiro grupo de instâncias estudadas.

Outro fator de qualidade de um algoritmo que busca um espaço em aplicações é sua desvinculação da programação linear. Um programa combinatório é muito mais facilmente utilizável do que outro que necessita de um pacote de programação linear para funcionar. Algoritmos baseados no método primal dual, como os concebidos neste trabalho para o problema da Classificação Métrica Uniforme e para o de clusterização, podem ser facilmente distribuídos e rodar em clientes que possuam um sistema computacional mesmo que de baixo poder de processamento, como celulares ou outros aparelhos eletrônicos. Algoritmos que necessitam de um resolvedor de programação linear ou linear inteira tem

requisitos de hardware que inviabilizam seu uso em sistemas menores.

Confrontando algoritmos aproximados com algoritmos de programação inteira, vemos que cada abordagem tem um nicho de aplicabilidade diferente. Quão confortável ficamos ao responder a um cliente que o caminho hamiltoniano oferecido para a viatura da polícia é o menor possível dentro de uma precisão de 10^{-4}. Porém, um cálculo que demore uma hora para ser realizado pode inviabilizar uma aplicação. Um algoritmo aproximado pode servir como um bom limitante inicial, diminuindo o processamento do algoritmo inteiro, ou mesmo fornecendo uma solução que satisfaça as necessidades do cliente.

Uma métrica que pode ser levada em conta é a relação custo-benefício entre a qualidade da solução e seu tempo computacional. Heurísticas, algoritmos aproximados e programação inteira ocupam espaços distintos em relação a esta métrica. Espera-se que uma heurística seja rápida e possua uma qualidade média aceitável. Para um algoritmo aproximado, exige-se um comportamento de pior caso dentro de certos limites. A programação inteira está interessada em soluções ótimas, mesmo que estas sejam custosas.

Os resultados obtidos no Capítulo 4 mostram que um algoritmo $(54 + \epsilon)$-aproximado pode encontrar soluções melhores que as melhores heurísticas conhecidas para um problema clássico em tempo competitivo. Isto nos diz que um fator de aproximação mais elevado não representa diretamente o desvio prático do algoritmo. Quanto às heurísticas, mesmo as melhores conhecidas, podem desviar-se mais que algoritmos aproximados. Isto vem a fortalecer o fato de que algoritmos aproximados são uma engenhosa maneira de construir a solução a partir de técnicas eficientes com respaldo teórico, enquanto heurísticas são intuitivamente interessantes e recebem validação experimental. Uma análise formal da qualidade das soluções produzidas, como feito para algoritmos aproximados, enriquece algoritmos e heurísticas.

Ao tratarmos complexidade de tempo, a inclusão de uma restrição de caráter cúbico como a restrição de desigualdade triangular, ou mesmo restrições mais densas, traz ganhos para a análise teórica, gerando relaxações mais justas e novas restrições que podem ser utilizadas na análise do algoritmo. Entretanto, tais restrições podem ser inviáveis para a maioria das aplicações. Sistemas lineares ou semidefinidos com este tipo de restrição estão fadados a instâncias pequenas e moderadas.

Quanto à experiência com programação semidefinida, ela é uma poderosa ferramenta para construção de boas relaxações, dado que é mais forte que relaxações lineares. Os aspectos práticos envolvidos mostram que é uma área carente. Os experimentos mostraram que os resolvedores podem ser ainda muito melhorados a fim de se tornarem mais competitivos.

Outro aspecto relevante dos algoritmos aproximados é que eles estão inseridos dentro da teoria da computação e seus aspectos teóricos são suficientes para justificá-los na maioria dos casos. A busca por melhores fatores de aproximação em problemas clássicos

levam a comunidade a buscar novas técnicas de algoritmos e de provas. Dois exemplos de destaque são o problema de Localização de Facilidades, onde Jain *et al.* propuseram um método de LP revelador de fator [28] e o problema do corte Máximo, onde foi introduzido o uso da programação semidefinida em algoritmos de aproximação por Goemans e Williamson [22], dentre outros trabalhos inovadores e premiados internacionalmente.

Quebras de fatores de aproximação, demonstração da existência de esquemas de aproximação, demonstração de resultados de inaproximabilidade são exemplos da busca por novos paradigmas, mesmo que tais resultados estejam desvinculados de uma aplicação direta.

Com esta tese fizemos uma sistemática análise de problemas interessantes para a Ciência da Computação. Propusemos novos algoritmos, obtivemos resultados experimentais e propusemos técnicas, como a técnica das esferas no Capítulo 4. Com isso, oferecemos uma tese que cumpre o seu papel de dar um pequeno passo na longa caminhada da Ciência.

Referências Bibliográficas

[1] Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 553–562, New York, NY, USA, 2005. ACM.

[2] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 222–231, New York, NY, USA, 2004. ACM.

[3] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B*, 36(2):192–236, 1974.

[4] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B*, 48(3):259–302, 1986.

[5] Evandro C. Bracht, Luis, A. A. Meira, and F. K. Miyazawa. A greedy approximation algorithm for the uniform metric labeling problem analyzed by a primal-dual technique. *ACM Journal of Experimental Algorithmics*, 10:2.11, 2005.

[6] Evandro C. Bracht, Luis A. A. Meira, and Flavio Keidi Miyazawa. A greedy approximation algorithm for the uniform labeling problem analyzed by a primal-dual technique. In *WEA*, volume 3059 of *Lecture Notes in Computer Science*, pages 145–158. WEA, Springer, 2004.

[7] Evandro C. Bracht, Luis A.A. Meira, and F.K. Miyazawa. Instances and programs to *Uniform Metric Labeling Problem*. `http://www.ic.unicamp.br/~aprox/labeling`.

[8] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 307–318, 1998.

[9] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *20th Annual IEEE Conference on Computational Complexity*, pages 144–153, June 2005.

[10] Shuchi Chawla, Anupam Gupta, and Harald Räcke. Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 102–111, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[11] Chandra Chekuri, Sanjeev Khanna, Joseph Naor, and Leonid Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 109–118, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[12] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research, Issue 3*, 4:233–235, August 1979.

[13] Fernand S Cohen. *Markov random fields for image modelling and analysis*, pages 243–272. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 1986. isbn:0-89838-177-0.

[14] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[15] Nikhil R. Devanur, Subhash A. Khot, Rishi Saket, and Nisheeth K. Vishnoi. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 537–546, New York, NY, USA, 2006. ACM Press.

[16] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56:9–33, 2004.

[17] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[18] R.C. Dubes and A.K. Jain. Random field models in image analysis. *Journal of Applied Statistics*, 16(2):131–164, 1989.

[19] Barbara M. P. Fraticelli. *Semidefinite Cuts and Partial Convexification Techniques with Applications to Continuous Nonconvex Optimization, Stochastic Integer Programming, and Facility Layout Problems*. PhD thesis, Virginia Polytechnic Institute, 2001.

[20] K. Freivalds. A nondifferentiable optimization approach to ratio-cut partitioning. In *Proc. 2nd Workshop on Efficient and Experimental Algorithms*, Lectures Notes on Computer Science, LNCS 2647, pages 134–147. Springer-Verlag, 2003.

[21] Katsuki Fujisawa, Masakasu Kojima, Kasuhide Nakata, and Makoto Yamashita. *SDPA Users's Manual, Version 6.00* page 39, 2002.

[22] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42:1115–1145, 1995.

[23] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31:228–248, 1999.

[24] A. Gupta and E. Tardos. A constant factor approximation algorithm for a class of classification problems. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 652–658, 2000.

[25] C. Helmberg. Semidefinite programming for combinatorial optimization. *Habilitationsschrift*, ZIB-Report ZR-00-34, Konrad-Zuse-Zentrum, October 2000.

[26] D. S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(2):148–162, 1982.

[27] Thomas Hofmeister and Martin Hühne. Semidefinite programming and its applications to approximation algorithms. In *Lectures on Proof Verification and Approximation Algorithms. (the book grow out of a Dagstuhl Seminar, April 21-25, 1997)*, pages 263–298, London, UK, 1998. Springer-Verlag.

[28] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

[29] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proc. of the thiry-fourth annual ACM symposium on Theory of computing*, pages 731–740. ACM Press, 2002.

[30] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[31] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.

[32] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry: Theory and Applications*, 28:89–112, 2004.

[33] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *Proceedings of the 40th Annuall IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1999.

[34] Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A simple linear time $(1 + \epsilon)$-approximation algorithm for k-means clustering in any dimensions. In *Proc. of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 454–462, 2004.

[35] M. Laurent and R. Rendl. Semidefinite programming and integer programming. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Handbook on Discrete Optimization*, pages 393–514. Elsevier B.V., December 2005.

[36] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. *Proc. 29th Ann. IEEE Symp. on Foundations of Comput. Sci., IEEE Computer Society, 422-431.*, 1988.

[37] Yuying Li. A Newton acceleration of the Weiszfeld algorithm forminimizing the sum of euclidean distances. *Computational Optimization and Applications*, 10(3):219–242, 1998.

[38] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[39] M. Mahdian, Y. Ye, and J. Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing*, 36(2):411–432, 2006.

[40] J. Matousek. On approximate geometric k-clustering. *Discrete Computational Geometry*, 24(1):61–84, 2000.

[41] J. E. Mitchell. Restarting after branching in the SDP approach to MAX-CUT and similar combinatorial optimization problems. *Journal of Combinatorial Optimization*, 5(2):151–166, 2001.

[42] Dash Optimization. *Xpress-MP Manual. Release 13*. 2002.

[43] David B. Shmoys. Cut problems and their application to divide-and-conquer, 1996. In D. Hochbaum, editor, Approximation Algorithms for NP-Hard Problems, pages 192–235. PWS Publishing.

[44] D.B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proc. of the 29th ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[45] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[46] S. Wang and J. Siskind. Image segmentation with ratio cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):675–690, 2003.

[47] E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnes est minimum. *Tohoku Math. Journal*, 43:355–386, 1937.

[48] Guoliang Xue and Yinyu Ye. An efficient algorithm for minimizing a sum of p-norms. *SIAM J. on Optimization*, 10(2):551–579, 1999.

[49] Makoto Yamashita. SDPA - Semidefinite programming solver, 2002. available at: http://grid.r.dendai.ac.jp/sdpa/.

[50] Heng Yang, Yinyu Ye, and Jiawei Zhang. An approximation algorithm for scheduling two parallel machines with capacity constraints. *Discrete Appl. Math.*, 130(3):449–467, 2003.