

Instituto de Matemática e Estatística
Universidade de São Paulo

Algoritmos de Aproximação e Problemas de *Clustering*

Samuel Praça de Paula

Supervisora: Prof^a Cristina Gomes Fernandes

São Paulo, 2012

Sumário

I	Parte Objetiva	1
1	Introdução	2
2	Conceitos	2
2.1	Problemas de decisão e classes de complexidade	2
2.1.1	NP-completude	2
2.2	Problemas de otimização	2
2.2.1	Problemas NP-difíceis	2
2.3	Grafos	2
2.4	Programação Linear	2
3	Uma Introdução aos Algoritmos de Aproximação	3
3.1	Por que Algoritmos de Aproximação?	3
3.2	Uma Introdução às Técnicas	4
3.3	Arredondamento Determinístico	5
3.4	Arredondando uma Solução Dual	7
3.5	O Método Primal-Dual	10
3.6	Um Algoritmo Guloso	11
3.7	Um Algoritmo de Arredondamento Aleatorizado	16
4	Problemas de <i>clustering</i>	18
4.1	Minimização da máxima distância entre elementos de um mesmo <i>cluster</i>	19
4.1.1	Um algoritmo de aproximação para <i>k-tMM</i>	20
4.1.2	Complexidade dos problemas de <i>clustering</i>	22
4.2	Problemas de <i>k</i> -centros com tolerância a falhas	24
5	Conclusões	24

II	Parte Subjetiva	26
6	Desafios e frustrações	27
7	Relação entre o curso e o trabalho de formatura	27
8	Trabalhos futuros	27

Parte I

Parte Objetiva

1 Introdução

Encontrar um conjunto mínimo de pedaços suspeitos de código de forma que, com os pedaços tomados, se possam determinar “assinaturas” de todo um conjunto dos vírus de computador conhecidos. Determinar onde devem ser abertos centros de distribuição de um certo produto de modo que nenhuma loja que o vende esteja muito distante do centro de distribuição mais próximo. Descobrir como dividir elementos em categorias de modo que as categorias não sejam muito heterogêneas.

Há uma quantidade imensa de problemas, tais como esses, que podem ser modelados como problemas de *otimização combinatória*. Podemos resolvê-los com o auxílio de um computador. No entanto, dentre estes, muitos dos mais interessantes são computacionalmente difíceis, no sentido de que não podem ser resolvidos por algoritmos eficientes (isto é, de tempo polinomial no tamanho da descrição do problema) a menos que $P = NP$. Como em geral precisamos obter respostas em tempo razoável, uma maneira de enfrentar esse problema são os Algoritmos de Aproximação.

Algoritmos de aproximação são algoritmos que, para um problema de otimização, calculam eficientemente soluções aproximadamente ótimas. Ou seja, terminam em tempo polinomial e devolvem uma solução cujo valor dista do valor ótimo no máximo em um determinado fator chamado de garantia de desempenho.

Neste trabalho, descrevemos algumas técnicas de desenvolvimento de algoritmos de aproximação através de exemplos de algoritmos para variados problemas, com foco nos problemas de *clustering*, como o *k-center* e algumas de suas numerosas variantes. Estão presentes algoritmos e resultados clássicos encontrados na literatura, bem como resultados presentes em artigos científicos recentes e/ou cujos resultados são os melhores conhecidos – ou mesmo os melhores possíveis – para os problemas estudados.

2 Conceitos

Esta seção comportará um breve texto com o objetivo de introduzir as noções de complexidade computacional, problemas de otimização, grafos, programação linear e possivelmente outros aspectos técnicos necessários para o entendimento do trabalho.

2.1 Problemas de decisão e classes de complexidade

2.1.1 NP-completude

2.2 Problemas de otimização

2.2.1 Problemas NP-difíceis

2.3 Grafos

2.4 Programação Linear

A seguir, apresentamos uma adaptação do primeiro capítulo do livro de Williamson e Shmoys [6] como uma introdução geral às técnicas de algoritmos de aproximação. Seguimos o conteúdo e ordem

de apresentação originais, por vezes fazendo algumas modificações e complementos, em especial nas demonstrações fornecidas.

3 Uma Introdução aos Algoritmos de Aproximação

3.1 Por que Algoritmos de Aproximação?

A otimização combinatória trata de como tomar diversas decisões do mundo real de modo a obter o melhor resultado possível dado um objetivo (ex.: maximizar lucro, minimizar gasto de material, etc.)

No entanto, os problemas de otimização combinatória mais interessantes são, em sua maioria, NP-difíceis. Isso significa que não conhecemos algoritmos eficientes que resolvam tais problemas e, de fato, a menos que $P = NP$, algoritmos eficientes para esses problemas *não existem*. O que, então, podemos fazer em tais casos?

Desejamos conhecer algoritmos de otimização que:

1. encontrem soluções ótimas
2. em tempo polinomial
3. para qualquer instância.

No entanto, para problemas NP difíceis, no mínimo um desses requisitos precisa ser relaxado, a menos que $P = NP$.

Há abordagens que relaxam o requisito “para qualquer instância”, buscando algoritmos que resolvam, em tempo polinomial, um subconjunto das instâncias de um dado problema.

Uma forma mais comum de atacar o problema é desistir da exigência de que o consumo de tempo seja polinomial. Buscam-se formas espertas de explorar todo o espaço de soluções de forma a encontrar soluções ótimas, podendo-se ter de esperar por minutos ou horas, ou mesmo correndo o risco de que a instância usada nem possa ser resolvida em tempo razoável. Isso é feito por quem resolve formulações de problemas de otimização como problemas de programação inteira, ou pesquisadores de Inteligência Artificial que usam técnicas como a busca A^* , por exemplo.

No entanto, a abordagem mais comum é relaxar a restrição de otimalidade e, em vez disso, buscar uma solução “suficientemente boa”, principalmente se tal solução puder ser obtida em questão de segundos — ou menos. Há uma extensa variedade de técnicas nessa categoria, e que muitas vezes resultam em soluções que são boas na prática.

Essa terceira forma de encarar os problemas NP-difíceis de otimização combinatória é a que levaremos em consideração: iremos relaxar a exigência de otimalidade, mas relaxaremos o mínimo que pudermos. Tentaremos encontrar soluções que tenham valor próximo ao da solução ótima (em termos da função objetivo para cada problema).

Definição 3.1. *Uma α -aproximação para um problema de otimização é um algoritmo polinomial que, para toda instância desse problema, produz uma solução cujo valor está a um fator α do valor de uma solução ótima.*

Nesse caso, dizemos que α é a *garantia de desempenho* do algoritmo (também chamada de *razão de aproximação* ou *fator de aproximação* do algoritmo). Adotaremos o padrão de que $\alpha > 1$ para problemas de minimização e $0 < \alpha < 1$ para problemas de maximização.

Por que, afinal, estudar algoritmos de aproximação?

- Porque precisamos solucionar problemas de otimização que aparecem com frequência no mundo real e são, muitas vezes, NP-difíceis.
- Porque bons algoritmos de aproximação para versões mais simples de um problema podem nos dar boas heurísticas para o problema original.
- Porque isso nos fornece uma base matemática rigorosa com a qual estudar heurísticas.
- Porque nos dá uma maneira de medir quão difíceis são vários problemas de otimização discreta (em termos de quão bem eles podem ser aproximados).
- Por prazer!

Existem problemas que admitem algoritmos de aproximação realmente muito bons, a ponto de terem esquemas de aproximação em tempo polinomial.

Definição 3.2. Um esquema de aproximação em tempo polinomial (*PTAS*, do inglês polynomial-time approximation scheme) é uma família de algoritmos $\{A_\varepsilon\}$ em que, para cada $\varepsilon > 0$, A_ε é uma $(1 + \varepsilon)$ -aproximação, para problemas de minimização, ou uma $(1 - \varepsilon)$ -aproximação, para problemas de maximização.

3.2 Uma Introdução às Técnicas

Consideremos o seguinte problema: dados um conjunto de elementos $E = \{e_1, \dots, e_n\}$, conjuntos S_1, S_2, \dots, S_m tais que cada $S_j \subseteq E$ e um peso não negativo w_j para cada S_j , encontrar uma coleção de peso mínimo dos conjuntos que cubra E . Isto é, desejamos encontrar $I \subseteq \{1, \dots, m\}$ que minimize $\sum_{j \in I} w_j$ sujeito a $\bigcup_{j \in I} S_j = E$. Se, para todo conjunto S_j , o peso é $w_j = 1$, o problema é chamado de cobertura por conjuntos *sem peso*.

O problema da cobertura por conjuntos é uma abstração de diversos tipos de problemas, como a *cobertura por vértices*. Em tal problema, dados um grafo (não dirigido) $G = (V, E)$ e um peso não negativo w_i para cada vértice $i \in V$, desejamos encontrar um conjunto de vértices de peso mínimo $C \subseteq V$ tal que para cada aresta $(i, j) \in E$, ou $i \in C$ ou $j \in C$. É fácil ver por que a cobertura por vértices é um caso particular da cobertura por conjuntos; o conjunto E da cobertura por conjuntos é exatamente o conjunto de arestas, e para cada $i \in V$, cria-se um conjunto $S_i = \{e \in E : e \text{ incide em } i\}$ de peso w_i (peso do vértice).

Vamos representar o problema da cobertura por conjuntos como um problema de programação linear. Teremos, para cada conjunto S_j , uma variável de decisão x_j , que indicará se S_j é incluído ou não na solução. Queremos que a variável seja booleana, e portanto formularemos o problema como um PLI (problema de programação linear inteira). Para garantir que todo elemento aparece na união dos conjuntos que compõem a solução, devemos ter a restrição

$$\sum_{j: e_i \in S_j} x_j \geq 1, \text{ para cada } e_i, i = 1, \dots, n.$$

Além disso, queremos que o peso dos conjuntos que aparecem na solução seja mínimo. Ou seja, nossa função objetivo, a ser minimizada, é $\sum_{j=1}^m w_j x_j$. Daí, o PLI equivalente ao nosso problema é:

$$\begin{aligned}
& \text{minimizar } \sum_{j=1}^m w_j x_j \\
& \text{sujeito a } \sum_{j: e_i \in S_j} x_j \geq 1, \quad i = 1, \dots, n, \\
& x_j \in \{0, 1\}, \quad j = 1, \dots, m.
\end{aligned}$$

Seja Z_{PLI}^* o valor ótimo desse PLI para uma dada instância do problema. Como o PLI é um modelo exato, temos $Z_{PLI}^* = \text{OPT}$, onde OPT é o valor de uma solução ótima do problema de cobertura por conjuntos para tal instância.

No entanto, em geral, um PLI não pode ser resolvido em tempo polinomial. De fato, se pudéssemos resolver esse PLI em tempo polinomial, teríamos resolvido também a cobertura por conjuntos, que é NP-difícil, e então teríamos $P = NP$.

Trabalharemos, então, com um PL (problema de programação linear), que pode ser resolvido em tempo polinomial:

$$\begin{aligned}
& \text{minimizar } \sum_{j=1}^m w_j x_j \\
& \text{s.a. } \sum_{j: e_i \in S_j} x_j \geq 1, \quad i = 1, \dots, n, \\
& x_j \geq 0, \quad j = 1, \dots, m.
\end{aligned}$$

Note-se que é desnecessário exigir que $x_j \leq 1$ devido à nossa função objetivo (dada uma solução, um $x_j > 1$ pode ser trocado por $x_j = 1$ sem perda de viabilidade ou aumento do custo da solução).

Esse PL é uma *relaxação* do PLI, o que significa que toda solução viável do PLI é viável também no PL, e o valor (custo) de uma solução viável do PLI é o mesmo no PL. Como toda solução ótima do PLI é viável no PL e tem valor Z_{PLI}^* , uma solução ótima do PL terá valor $Z_{PL}^* \leq Z_{PLI}^* = \text{OPT}$. Isso significa que o PL é uma maneira de obter, em tempo polinomial, um limite inferior para o valor ótimo (OPT) do problema de cobertura por vértices.

Veremos como usar o PL para obter algoritmos de aproximação para o problema de cobertura por conjuntos.

3.3 Arredondamento Determinístico

Tentemos recuperar uma solução para a cobertura de conjuntos a partir de uma solução ótima do PL, digamos x^* . Incluiremos S_j na solução se, e somente se, $x_j^* \geq 1/f$, onde f é o número máximo de conjuntos em que um elemento aparece, i.e.,

$$f = \max_{e_i \in E} |\{j : e_i \in S_j\}|. \quad (3.1)$$

Seja I o conjunto de índices j dos conjuntos incluídos nessa solução. O conjunto I indexa uma cobertura por conjuntos. Uma solução inteira para o problema \hat{x} é obtida definindo-se $\hat{x}_j = 1$ se $x_j^* \geq 1/f$, e $\hat{x}_j = 0$, caso contrário.

Lema 3.3. *A coleção de conjuntos S_j , $j \in I$, é uma cobertura por conjuntos.*

Demonstração. Diremos que um elemento e_i é coberto se a solução I dada pelo lema contém algum conjunto do qual e_i seja membro. Tomemos um elemento e_i qualquer e vejamos que ele está coberto. Como x^* é solução ótima, e portanto viável, do PL, sabemos que $\sum_{j:e_i \in S_j} x_j^* \geq 1$.

Pela definição de f , sabemos que e_i está em no máximo f conjuntos S_j diferentes. Isto é, há no máximo f termos nessa soma, e portanto pelo menos um desses termos tem que ser no mínimo igual a $1/f$. Então, para algum j tal que $e_i \in S_j$, vale que $x_j \geq 1/f$. Logo, $j \in I$, e o elemento e_i está coberto. \square

Teorema 3.4. *O algoritmo de arredondamento é uma f -aproximação para o problema de cobertura por conjuntos.*

Demonstração. Como a resolução do PL e a obtenção de uma cobertura a partir da solução do PL podem ser feitos em tempo polinomial, segue que o algoritmo termina em tempo polinomial.

Por nossa construção, $j \in I$ implica que $x_j^* \geq 1/f$. Então, para cada $j \in I$, vale que $1 \leq f \cdot x_j^*$.

Além disso, como $x^* \geq 0$ (pois é solução viável do PL), $f \geq 0$ e $w \geq 0$ (os pesos são não-negativos), segue que o termo $f w_j x_j$ é não negativo para cada $j = 1, \dots, m$.

Assim, o valor (custo) da cobertura dada pelo algoritmo é

$$\begin{aligned}
 \sum_{j \in I} w_j &\leq \sum_{j \in I} w_j \cdot (f \cdot x_j^*) \\
 &\leq \sum_{j \in I} w_j \cdot (f \cdot x_j^*) + \sum_{j \notin I} w_j \cdot (f \cdot x_j^*) \\
 &= \sum_{j=1}^m w_j \cdot (f \cdot x_j^*) \\
 &= f \sum_{j=1}^m w_j x_j^* \\
 &= f \cdot Z_{PL}^* \\
 &\leq f \cdot OPT
 \end{aligned}$$

onde a desigualdade final vem da discussão de que $Z_{PL}^* \leq OPT$. \square

Para o caso especial da cobertura por vértices, $f = 2$ pois cada aresta incide em exatamente dois vértices. Portanto, para esse caso, o algoritmo acima é uma 2-aproximação.

3.4 Arredondando uma Solução Dual

Para começar, suponha que cada elemento e_i é taxado em um preço não negativo y_i por sua cobertura numa solução do problema de cobertura por conjuntos. É natural imaginar que alguns elementos podem ser cobertos por conjuntos de baixo peso, enquanto outros acabem exigindo a inclusão de conjuntos de grande peso na solução. Gostaríamos de capturar essa distinção cobrando pequenas taxas dos elementos no primeiro caso, enquanto elementos como no segundo caso tenham que pagar preços maiores. Para que os preços sejam razoáveis, não pode ocorrer de a soma dos preços dos elementos em um conjunto S_j seja maior que o peso do próprio conjunto, uma vez que podemos cobrir todos esses elementos pagando o peso w_j . Assim, para cada conjunto S_j , temos o seguinte limitante superior para os preços:

$$\sum_{i:e_i \in S_j} y_i \leq w_j.$$

Podemos encontrar o máximo preço total que podem ser cobrados dos elementos resolvendo o seguinte PL:

$$\begin{aligned} & \text{maximizar } \sum_{i=1}^n y_i \\ & \text{sujeito a } \sum_{i:e_i \in S_j} y_i \leq w_j, \quad j = 1, \dots, m, \\ & y_i \geq 0, \quad i = 1, \dots, n. \end{aligned}$$

Esse PL é o *dual* da relaxação do problema da cobertura por conjuntos (e a este chamaremos de *primal*). Isso nos garante uma série de relações importantes, dentre as quais a propriedade da *dualidade fraca*, que diz que para quaisquer x e y viáveis no PL primal e no dual, respectivamente, vale que

$$\sum_{i=1}^n y_i \leq \sum_{j=1}^m w_j x_j,$$

isto é, o valor da solução y no dual não é maior que o valor da solução x . Em particular, nenhuma solução viável do PL dual tem valor maior que o valor de uma solução ótima do primal. Então, para qualquer y viável no dual, vale que $\sum_{i=1}^n y_i \leq \text{OPT}$ (pois já vimos que $Z_{PL}^* \leq \text{OPT}$). Esse fato será importante no desenvolvimento de algoritmos de aproximação.

Temos, ainda, outra propriedade importante: a chamada *dualidade forte*. É a propriedade de que uma solução ótima do PL dual e uma solução ótima do PL primal têm valores *iguais*. No nosso caso, isso significa que, se x^* é uma solução ótima do primal e y^* uma solução ótima do dual, então vale a seguinte igualdade:

$$\sum_{j=1}^m w_j x_j^* = \sum_{i=1}^n y_i^*.$$

Agora, usemos uma solução do nosso PL dual para obter uma solução da cobertura por conjuntos.

Seendo y^* uma solução ótima dual, tomemos como solução a coleção dos S_j para os quais a desigualdade dual é satisfeita *sem folga*. Isto é, tomamos os S_j para os quais $\sum_{i:e_i \in S_j} y_i^* = w_j$. Seja I' o conjunto dos índices dos conjuntos em tal coleção.

Lema 3.5. *A coleção de conjuntos $\{S_j : j \in I'\}$ é uma cobertura por conjuntos.*

Demonstração. Suponha que exista algum elemento e_k não coberto. Então, para todo conjunto S_j contendo e_k , deve valer que

$$\sum_{i:e_i \in S_j} y_i^* < w_j. \quad (3.2)$$

Seja ε a menor diferença entre o lado direito e o esquerdo de todas as desigualdades para os conjuntos em que e_k ocorre. Isto é,

$$\varepsilon = \min_{j:e_k \in S_j} (w_j - \sum_{i:e_i \in S_j} y_i^*).$$

Pela desigualdade (3.2), sabemos que $\varepsilon > 0$.

Considere agora uma nova solução dual y' em que $y'_k = y_k^* + \varepsilon$ e todos os outros componentes de y' são iguais aos de y^* .

Para cada j tal que $e_k \in S_j$, vale que

$$\sum_{i:e_i \in S_j} y'_i = \sum_{i:e_i \in S_j} y_i^* + \varepsilon \leq w_j,$$

pela definição de ε . E, para cada j tal que $e_k \notin S_j$, temos

$$\sum_{i:e_i \in S_j} y'_i = \sum_{i:e_i \in S_j} y_i^* \leq w_j.$$

Segue, então, que y' é uma solução viável para o PL dual e, além disso,

$$\sum_{i=1}^n y'_i > \sum_{i=1}^n y_i^*,$$

pela maneira como construímos y' .

No entanto, isso contradiz a otimalidade de y^* . Logo, todo elemento necessariamente está coberto e I' é uma cobertura por conjuntos. \square

Teorema 3.6. *O algoritmo de arredondamento dual descrito acima é uma f -aproximação para o problema da cobertura por conjuntos.*

Demonstração. Assim como todo PL, nosso programa dual pode ser resolvido em tempo polinomial. É evidente que nossa construção de uma cobertura por conjuntos a partir da solução do PL também pode ser feita em tempo polinomial. Assim, o algoritmo sugerido é polinomial, bastando provar sua garantia de desempenho.

A ideia central para isso é o seguinte argumento de “cobrança”: quando escolhemos um conjunto S_j para estar na cobertura, nós “pagamos” por ele cobrando y_i^* de cada um de seus elementos e_i .

Cada elemento e_i paga sua taxa de cobertura (y_i^*) no máximo uma vez para cada conjunto que o contém — e, portanto, no máximo f vezes. Então, o custo total é no máximo $f \sum_{i=1}^m y_i^*$, ou seja, f vezes o valor de uma solução ótima.

Isto é, como $j \in I'$ apenas se $w_j = \sum_{i:e_i \in S_j} y_i^*$, temos que o custo da cobertura I' é

$$\begin{aligned} \sum_{j \in I'} w_j &= \sum_{j \in I'} \sum_{i:e_i \in S_j} y_i^* \\ &= \sum_{i=1}^n |\{j \in I' : e_i \in S_j\}| \cdot y_i^* \\ &\leq \sum_{i=1}^n f \cdot y_i^* \\ &= f \sum_{i=1}^n y_i^* \\ &\leq f \cdot \text{OPT}, \end{aligned}$$

onde a última desigualdade vem da dualidade fraca. □

É possível mostrar que esse algoritmo não pode se sair melhor do que o da Seção 3.3. Mais precisamente, podemos mostrar que se I indexa a solução devolvida pelo algoritmo de arredondamento daquela seção, então $I \subseteq I'$. Como veremos a seguir, isso vem da propriedade de *folgas complementares* em soluções ótimas de um par primal-dual.

A dualidade fraca para o par de programas lineares que estamos considerando pode ser obtida pela seguinte cadeia de igualdades e desigualdades:

$$\sum_{i=1}^n y_i \leq \sum_{i=1}^n y_i \sum_{j:e_i \in S_j} x_j \tag{3.3}$$

$$\begin{aligned} &= \sum_{j=1}^m x_j \sum_{i:e_i \in S_j} y_i \\ &\leq \sum_{j=1}^m x_j w_j. \end{aligned} \tag{3.4}$$

A desigualdade (3.3) se deve ao fato de que $\sum_{j:e_i \in S_j} x_j \geq 1$ para cada i , para x viável. A desigualdade (3.4) se deve ao fato de que $\sum_{i:e_i \in S_j} y_i \leq w_j$ para cada j , para y viável.

Para soluções ótimas x^* e y^* , a dualidade forte implica que seus valores são iguais, isto é, $\sum_{i=1}^n y_i^* = \sum_{j=1}^m w_j x_j^*$. Então, as desigualdades (3.3) e (3.4) devem ser obedecidas com igualdade.

A única maneira de isso ser satisfeito é que, sempre que $y_i^* > 0$, valha que $\sum_{j:e_j \in S_j} x_j^* = 1$, e sempre que $x_j^* > 0$, então $\sum_{i:e_i \in S_j} y_i^* = w_j$. Isto é, sempre que uma variável do PL (primal ou dual) é não nula, a restrição correspondente no dual ou primal deve ser satisfeita sem folgas (i.e., com igualdade).

Tal relação é conhecida como a *condição de folgas complementares*. Assim, se x^* e y^* são soluções ótimas, as folgas complementares devem ser satisfeitas. E a recíproca é verdadeira: se \hat{x} e \hat{y} são soluções viáveis primal e dual, respectivamente, e as folgas complementares são satisfeitas, segue que \hat{x} e \hat{y} têm valores iguais e, portanto, são soluções ótimas.

No algoritmo de arredondamento primal, incluímos j em I se $x_j^* \geq 1/f$. Assim, $j \in I$ implica $x_j^* > 0$, que por sua vez implica $\sum_{i: e_i \in S_j} y_i^* = w_j$, satisfazendo nosso critério para incluir j em I' .

Logo, $j \in I$ implica $j \in I'$. Ou seja, $I \subseteq I'$.

3.5 O Método Primal-Dual

Uma das desvantagens dos algoritmos apresentados nas duas seções anteriores é que ambas envolvem a resolução de um PL. Embora programas lineares possam ser resolvidos em tempo polinomial e haja algoritmos que são muito rápidos na prática (Simplex), algoritmos que resolvem um problema específico são, muitas vezes, muito mais rápidos.

A ideia básica do algoritmo dessa seção é que o algoritmo de arredondamento dual da seção anterior usa poucas propriedades da *otimalidade* da solução dual y . Em vez de realmente resolver o PL dual, podemos simplesmente construir uma solução viável com tais propriedades. Construir tal solução dual pode ser muito mais rápido do que realmente resolver o PL dual e portanto levará a um algoritmo mais rápido.

O algoritmo da seção anterior usa as seguintes propriedades:

1. $\sum_{i=1}^n y_i \leq \text{OPT}$, o que é verdade para qualquer y dual viável.
2. Incluímos j em I' exatamente quando $\sum_{i: e_i \in S_j} y_i = w_j$, e então I' indexa uma cobertura por conjuntos.

Esses dois fatos juntos foram usados para provar que o custo de I' não é maior do que f vezes o valor ótimo.

De fato, é a *demonstração* do Lema 3.5 (que diz que construímos uma cobertura por conjuntos válida) que mostra como obter um algoritmo que constrói uma solução dual.

Considere uma solução viável dual qualquer y , e seja T o conjunto de índices de todas as restrições do dual satisfeitas com igualdade, isto é, $T = \{j : \sum_{i: e_i \in S_j} y_i = w_j\}$.

Se T indexa uma cobertura por conjuntos, nada temos a fazer. Senão, então algum elemento e_i não está coberto, e, como mostrado na prova do Lema 3.5, é possível obter uma solução de maior valor aumentando y_i de um $\varepsilon > 0$. Mais especificamente, podemos aumentar y_i de $\min_{j: e_i \in S_j} (w_j - \sum_{k: e_k \in S_j} y_k)$, que é um valor positivo já que e_i não está coberto. Para um j que minimiza a expressão vale que a restrição referente ao S_j passa a ser satisfeita com igualdade. Além disso, a solução dual modificada continua viável. Assim, podemos adicionar um tal j a T , e assim o elemento e_i agora está coberto pelos conjuntos indexados por T . Repetimos o processo até que T seja uma cobertura por conjuntos.

Como um e_i adicional é coberto a cada passo, o processo é repetido no máximo n vezes. Para completar a descrição do algoritmo, só precisamos dar uma solução dual viável inicial. É fácil encontrar uma tal solução, visto que o vetor nulo satisfaz as restrições do PL dual.

Esse tipo de algoritmo é chamado de *primal-dual* por analogia ao método primal-dual usado em algoritmos para outros problemas de otimização, como programação linear, fluxo máximo em redes,

caminhos mais curtos, etc. Um algoritmo primal-dual começa com viabilidade dual (uma solução dual viável), e usa a informação dual para inferir uma solução primal, não necessariamente viável. Se a solução primal realmente não é viável, a solução dual é modificada de modo a incrementar o seu valor, e esse processo é repetido. No nosso caso, temos:

Algoritmo 3.1: Algoritmo primal-dual para a cobertura por conjuntos.

Entrada: Elementos e_i , conjuntos S_j e seus pesos w_j

Saída: Indexação I de uma cobertura por conjuntos

```

1  $y \leftarrow 0$ ;
2  $I \leftarrow \emptyset$ ;
3 enquanto  $I$  não é uma cobertura faça
4   | Seja  $i$  tal que  $e_i \notin \bigcup_{j \in I} S_j$ ;
5   |  $\varepsilon \leftarrow \min_{j: e_i \in S_j} (w_j - \sum_{k: e_k \in S_j} y_k)$ ;
6   |  $y_i \leftarrow y_i + \varepsilon$ ;
7   | Seja  $l$  com  $e_i \in S_l$  tal que  $\sum_{j: e_j \in S_l} y_j = w_l$ ;
8   |  $I \leftarrow I \cup \{l\}$ ;
9 fim

```

Teorema 3.7. O algoritmo 3.1 é uma f -aproximação para o problema da cobertura por conjuntos, onde f é o máximo número de ocorrências de um e_i nos conjuntos S_j .

Demonstração. O algoritmo demora no máximo n passos para acabar, uma vez que a cada passo pelo menos um elemento passa a ser coberto por I . Como cada passo leva tempo polinomial, o algoritmo é polinomial.

Como sempre alteramos y de modo que não perdemos sua viabilidade dual, segue que, ao fim do algoritmo, y é viável.

Observe que, ao final do algoritmo, I é o conjunto $\{S_j : \sum_{i: e_i \in S_j} y_i = w_j\}$, que é uma cobertura por conjuntos, pela condição de parada do algoritmo.

É fácil, então, ver que a demonstração do Teorema 3.6 vale com y no lugar do y^* . Portanto, a solução produzida pelo algoritmo primal-dual tem valor no máximo $f \cdot \text{OPT}$. \square

3.6 Um Algoritmo Guloso

Até aqui, as técnicas que vimos levaram, todas, a algoritmos de aproximação com garantia de desempenho igual a f (conforme definido em 3.1). Estudaremos, agora, outro tipo de técnica. Algoritmos *gulosos* são algoritmos que, a cada passo, tomam uma decisão que seja *localmente* ótima (“gulosa”), fazendo uma escolha que otimiza essa decisão em particular, sem preocupar-se com o fato de isso levar ou não a um ótimo global. Há problemas que podem ser resolvidos por algoritmos gulosos, isto é, problemas para os quais existem algoritmos gulosos que encontram necessariamente uma solução ótima. Este não será o caso dos nossos problemas, que são difíceis. No entanto, algoritmos gulosos são, muitas vezes, heurísticas boas e bastante usadas. No caso do problema da cobertura por conjuntos, veremos que um guloso nos dará uma aproximação com garantia de desempenho que pode ser significativamente melhor que f .

Este é um algoritmo guloso bastante natural para o problema da cobertura por conjuntos. A cada passo, escolhemos um conjunto que nos dê o melhor “custo-benefício”, isto é, um conjunto que minimiza a razão entre seu peso e a quantidade de elementos ainda não cobertos que ele contém. No caso de empate, pegamos arbitrariamente um dos conjuntos que atinja esse mínimo. Continuamos escolhendo novos conjuntos até que todos os elementos estejam cobertos.

Algoritmo 3.2: Um algoritmo guloso para o problema da cobertura por conjuntos.

```

1  $I \leftarrow \emptyset;$ 
2  $\hat{S}_j \leftarrow S_j, \quad \forall j;$ 
3 enquanto  $I$  não é uma cobertura faça
4    $l \leftarrow \arg \min_{j: \hat{S}_j \neq \emptyset} \frac{w_j}{|\hat{S}_j|};$ 
5    $I \leftarrow I \cup \{l\};$ 
6    $\hat{S}_j \leftarrow \hat{S}_j \setminus S_l, \quad \forall j;$ 
7 fim

```

É claro que isso nos dá um algoritmo polinomial: a cada passo, um novo S_j é adicionado à cobertura. Então, não podemos levar mais do que m passos. Além disso, no mínimo um elemento é coberto, então também certamente não precisamos de mais do que n passos. A cada passo, computamos $O(m)$ razões, cada uma em tempo constante. Portanto, nosso algoritmo tem complexidade $O(m \cdot \min\{m, n\})$.

Para prosseguir, precisaremos de uma nova notação e um fato.

Seja H_k o k -ésimo número harmônico, isto é,

$$H_k = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k}.$$

Note que H_k é aproximadamente $\int_1^k \frac{1}{x} = \ln k - \ln 1 = \ln k$, e portanto $H_k \approx \ln k$. Mais especificamente, $1 + \int_1^k \frac{1}{x} \leq H_k \leq \int_1^{k+1} \frac{1}{x}$, donde pode-se concluir que $1 + \ln k \leq H_n \leq \ln(k+1)$.

Fato 3.8. Dados a_1, \dots, a_k e b_1, \dots, b_k positivos, vale que

$$\min_{i=1, \dots, k} \frac{a_i}{b_i} \leq \frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i} \leq \max_{i=1, \dots, k} \frac{a_i}{b_i}.$$

Demonstração. Mostraremos apenas a primeira das duas desigualdades. A outra pode ser provada por uma argumentação similar.

Sejam p, q, r, s reais positivos. Se $\frac{p}{q} \leq \frac{r}{s}$, então $\frac{p}{q} \leq \frac{p+r}{q+s}$, pois

$$\begin{aligned} \frac{p}{q} \leq \frac{r}{s} &\Leftrightarrow ps \leq qr \text{ (pois } s, q \geq 0) \\ &\Leftrightarrow ps + pq \leq qr + pq \\ &\Leftrightarrow p(q+s) \leq q(p+r) \\ &\Leftrightarrow \frac{p}{q} \leq \frac{p+r}{q+s} \text{ (pois } q > 0 \text{ e } q+s > 0). \end{aligned}$$

Isso nos permite realizar uma prova por indução em k .

Para $k=1$, temos que $\frac{a_1}{b_1} = \min_{i=1, \dots, k} \frac{a_i}{b_i} = \frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i}$.

Seja, agora, $k > 1$ e suponha que a desigualdade valha para $k-1$. Seja l um índice tal que $\frac{a_l}{b_l} = \min_{i=1, \dots, k} \frac{a_i}{b_i}$. Então, $\frac{a_l}{b_l} \leq \min_{i \neq l} \frac{a_i}{b_i}$ e, por hipótese de indução, $\min_{i \neq l} \frac{a_i}{b_i} \leq \frac{\sum_{i \neq l} a_i}{\sum_{i \neq l} b_i}$.

Então, $\frac{a_l}{b_l} \leq \frac{\sum_{i \neq l} a_i}{\sum_{i \neq l} b_i}$ e, pela propriedade que provamos no início da demonstração, segue que

$$\frac{a_l}{b_l} \leq \frac{a_l + \sum_{i \neq l} a_i}{b_l + \sum_{i \neq l} b_i} = \frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i}.$$

□

Teorema 3.9. *O algoritmo 3.2 é uma H_n -aproximação para o problema da cobertura por conjuntos.*

Demonstração. A intuição básica para a análise do algoritmo é a seguinte. Seja OPT o valor de uma solução ótima S^* para o problema da cobertura por conjuntos. Sabemos que uma solução ótima cobre todos os n elementos com uma solução de peso OPT. Assim, pelo menos um dos conjuntos S_j de uma solução ótima cobre seus elementos com peso médio de no máximo OPT/ n . Isso porque se todos os elementos tivessem peso médio superior a esse valor, a soma destes pesos seria superior a OPT, o que é um absurdo, visto que tal soma é o peso de S^* .

Da mesma forma, após k elementos serem cobertos, a solução ótima consegue cobrir os $n - k$ restantes com uma solução de peso OPT, o que implica que algum conjunto cobre seus elementos ainda não cobertos com peso médio de no máximo OPT/ $(n - k)$. Então, em geral, o guloso paga por volta de OPT/ $(n - k + 1)$ para cobrir o k -ésimo elemento, dando uma garantia de desempenho igual a $\sum_{k=1}^n \frac{1}{n - k + 1} = H_n$.

Vamos formalizar essa intuição. Seja n_k o número de elementos que ainda não foram cobertos ao começo da k -ésima iteração. Se o algoritmo acaba após l iterações, então $n_1 = n$ e $n_{l+1} = 0$. Tome uma iteração k qualquer, e seja I_k o conjunto de índices dos conjuntos escolhidos nas iterações 1 a $k - 1$. Para cada $j = 1, \dots, m$, seja \hat{S}_j o conjunto de elementos não cobertos em S_j no começo dessa iteração, isto é, $\hat{S}_j = S_j - \bigcup_{p \in I_k} S_p$. Afirmamos que, para o conjunto de índice j escolhido na iteração k , vale que

$$w_j \leq \frac{n_k - n_{k+1}}{n_k} \text{OPT}. \quad (3.5)$$

Dada a desigualdade 3.5, podemos provar o teorema. Seja I o conjunto de índices dos conjuntos na solução final, ou seja, $I = I_{l+1}$. Então,

$$\begin{aligned} \sum_{j \in I} w_j &\leq \sum_{k=1}^l \frac{n_k - n_{k+1}}{n_k} \text{OPT} \\ &\leq \text{OPT} \cdot \sum_{k=1}^l \left(\frac{1}{n_k} + \frac{1}{n_k - 1} + \dots + \frac{1}{n_{k+1} + 1} \right) \\ &= \text{OPT} \cdot \sum_{i=1}^n \frac{1}{i} \\ &= H_n \cdot \text{OPT}, \end{aligned} \quad (3.6)$$

(3.7)

onde a desigualdade 3.6 vem do fato de que $\frac{1}{n_k} \leq \frac{1}{n_k - i}$ para $0 \leq i \leq n_k$.

Para provar a desigualdade 3.5, vamos primeiro argumentar que, na k -ésima iteração,

$$\min_{j: \hat{S}_j \neq \emptyset} \frac{w_j}{|\hat{S}_j|} \leq \frac{\text{OPT}}{n_k}. \quad (3.8)$$

Sendo O o conjunto de índices dos conjuntos em uma solução ótima, então a desigualdade 3.8 segue do Fato 3.8, observando-se que

$$\begin{aligned} \min_{j:\hat{S}_j \neq \emptyset} \frac{w_j}{|\hat{S}_j|} &\leq \min_{j \in O} \frac{w_j}{|\hat{S}_j|} \\ &\leq \frac{\sum_{j \in O} w_j}{\sum_{j \in O} |\hat{S}_j|} \\ &= \frac{\text{OPT}}{\sum_{j \in O} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_k}, \end{aligned}$$

onde a última desigualdade segue do fato de que como O é uma cobertura por conjuntos, $n_k \leq n$ certamente é no máximo igual ao número de elementos cobertos por O .

Seja j o índice de um conjunto que minimiza essa razão, de modo que $\frac{w_j}{|\hat{S}_j|} \leq \frac{\text{OPT}}{n_k}$. Se adicionarmos o conjunto S_j à nossa solução, então haverá $|\hat{S}_j|$ elementos não cobertos a menos, de forma que $n_{k+1} = n_k - |\hat{S}_j|$. Logo,

$$w_j \leq \frac{|\hat{S}_j| \text{OPT}}{n_k} = \frac{n_k - n_{k+1}}{n_k} \text{OPT}.$$

□

Podemos melhorar um pouco a garantia de desempenho do algoritmo usando o PL dual das seções anteriores na nossa análise. Seja g o tamanho máximo de um conjunto S_j , isto é, $g = \max_j |S_j|$. Lembremos que Z_{PL}^* é o valor ótimo do nosso PL. O teorema a seguir implica diretamente que o algoritmo guloso é uma H_g -aproximação, uma vez que $Z_{PL}^* \leq \text{OPT}$.

Teorema 3.10. *O algoritmo guloso desta seção devolve uma solução indexada por I tal que $\sum_{j \in I} w_j \leq H_g \cdot Z_{PL}^*$, onde g é o tamanho máximo de um conjunto S_j .*

Demonstração. Para provar o teorema, construiremos uma solução dual y inviável tal que $\sum_{j \in I} w_j = \sum_{i=1}^n y_i$. Mostraremos, então, que $y' = \frac{1}{H_g} y$ é uma solução dual viável. Pelo teorema de dualidade fraca, $\sum_{i=1}^n y'_i \leq Z_{PL}^*$, de modo que $\sum_{j \in I} w_j = \sum_{i=1}^n y_i = H_g \sum_{i=1}^n y'_i \leq H_g \cdot \text{OPT}$.

Essa técnica de construir uma solução dual inviável com o mesmo custo da solução primal viável construída e que, multiplicada por um único escalar, torna-se dual viável, chama-se *dual-fitting*.

Para construir a solução inviável y , suponha que escolhemos adicionar S_j à solução na iteração k . Então, para cada $e_i \in \hat{S}_j$, estabelecemos $y_i = \frac{w_j}{|\hat{S}_j|}$. Como cada $e_i \in \hat{S}_j$ está descoberto na iteração k e então continua coberto durante as próximas iterações do algoritmo (pois adicionamos S_j à solução), a variável y_i tem seu valor estabelecido exatamente uma vez; em particular, seu valor é definido na iteração em que o elemento e_i é coberto. Além disso, $w_j = \sum_{i: e_i \in \hat{S}_j} y_i$; isto é, o peso do conjunto S_j escolhido na k -ésima iteração é igual à soma dos y_i dos elementos de S_j que passam a ser cobertos nessa iteração. Isso implica que $\sum_{j \in I} w_j = \sum_{i=1}^n y_i$.

Falta provar que a solução dual $y' = \frac{1}{H_g} y$ é viável. Precisamos mostrar que, para cada S_j , $\sum_{i: e_i \in S_j} y'_i \leq w_j$. Tome um conjunto arbitrário S_j . Seja a_k o número de elementos nesse conjunto que estão descobertos no início da k -ésima iteração, de modo que $a_1 = |S_j|$, e $a_{l+1} = 0$. Seja A_k o conjunto dos elementos de S_j que passam a ser cobertos na k -ésima iteração, de modo que $|A_k| = a_k - a_{k+1}$. Se o conjunto S_p é escolhido na k -ésima iteração, então, para cada elemento $e_i \in A_k$, vale que

$$y'_i = \frac{w_p}{H_g |\hat{S}_p|} \leq \frac{w_j}{H_g a_k},$$

onde \hat{S}_p é o conjunto dos elementos de S_p descobertos no início da k -ésima iteração. A desigualdade segue do fato de que, se S_p é escolhido na k -ésima iteração, ele deve minimizar a razão entre seu peso e o número de elementos descobertos que ele contém. Assim,

$$\begin{aligned}
\sum_{i:e_i \in S_j} y'_i &= \sum_{k=1}^l \sum_{i:e_i \in A_k} y'_i \\
&\leq \sum_{k=1}^l (a_k - a_{k+1}) \frac{w_j}{H_g a_k} \\
&\leq \frac{w_j}{H_g} \sum_{k=1}^l \frac{a_k - a_{k+1}}{a_k} \\
&\leq \frac{w_j}{H_g} \sum_{k=1}^l \left(\frac{1}{a_k} + \frac{1}{a_k - 1} + \dots + \frac{1}{a_{k+1} + 1} \right) \\
&\leq \frac{w_j}{H_g} \sum_{i=1}^{|S_j|} \frac{1}{i} \\
&= \frac{w_j}{H_g} H_{|S_j|} \\
&\leq w_j,
\end{aligned}$$

onde a desigualdade final segue de que $|S_j| \leq g$. Aqui vemos o motivo de dividir a solução dual por H_g : sabemos que $H_{|S_j|} \leq H_g$ para todo j . \square

De fato, nenhuma aproximação para o problema da cobertura por conjuntos pode ter garantia de desempenho melhor do que H_n , sob uma hipótese um pouco mais forte do que $P \neq NP$.

Teorema 3.11. *Se existir uma $c \ln n$ -aproximação para o problema da cobertura por conjuntos sem peso para alguma constante $c < 1$, então existe um algoritmo determinístico de tempo $O(n^{O(\log \log n)})$ para todo problema NP-completo.*

Teorema 3.12. *Existe uma constante $c > 0$ tal que a existência de uma $c \ln n$ -aproximação para a cobertura por conjuntos implica que $P = NP$.*

Teoremas como esses são por vezes chamados de *resultados de inaproximabilidade* ou teoremas de *dificuldade*, uma vez que mostram que é NP-difícil fornecer soluções de valor próximo ao ótimo com certas garantias de desempenho.

As f -aproximações para a cobertura por conjuntos nos dão, para o caso especial da cobertura por vértices, uma 2-aproximação. Nenhum algoritmo com garantia melhor que essa é conhecido, e dois teoremas sobre a dificuldade da cobertura por vértices foram demonstrados.

Teorema 3.13. *Se existir uma α -aproximação para o problema da cobertura por vértices com $\alpha < 10\sqrt{5} - 21 \approx 1,36$, então $P = NP$.*

O teorema a seguir menciona uma conjectura chamada de *conjectura dos jogos únicos*, que diz, grosso modo, que um determinado problema (chamado de problema dos jogos únicos) é NP-difícil.

Teorema 3.14. *Admitindo que a conjectura dos jogos únicos seja correta, se existir uma α -aproximação para a cobertura por vértices com constante $\alpha < 2$, então $P = NP$.*

Então, admitindo-se que $P \neq NP$ e o problema dos jogos únicos é NP-difícil, já encontramos essencialmente a melhor aproximação possível para o problema da cobertura por vértices.

3.7 Um Algoritmo de Arredondamento Aleatorizado

Nesta seção trataremos de uma última técnica para a obtenção de um algoritmo de aproximação para o problema da cobertura por conjuntos. Embora o algoritmo não tenha uma garantia de desempenho melhor que a do guloso que estudamos (algoritmo 3.2), além de ser mais lento do que este, nós o estudaremos por ser um exemplo introdutório de algoritmo de aproximação aleatorizado.

Novamente, o algoritmo irá resolver um PL obtido da relaxação do PLI que modela o problema, e então arredondar a solução fracionária para uma inteira. Mas, em vez de fazê-lo de maneira determinística, o algoritmo irá fazê-lo de maneira aleatória, usando a técnica do *arredondamento aleatorizado*. Seja x^* uma solução ótima para o PL. Gostaríamos de arredondar os valores fracionários de x^* ou para 0 ou para 1 de modo a obter uma solução \hat{x} para o PLI sem que o custo desta seja muito mais alto. A ideia central aqui é que iremos interpretar o valor fracionário x_j^* como a probabilidade de que \hat{x}_j deveria ser 1. Assim, cada S_j é incluído na nossa solução com probabilidade x_j^* , e esses m eventos (a inclusão de S_j na solução) são eventos aleatórios independentes.

Seja X_j uma variável aleatória que vale 1, se o conjunto S_j é incluído na solução, e 0, caso contrário. Então o valor esperado da solução é

$$E \left[\sum_{j=1}^m w_j X_j \right] = \sum_{j=1}^m w_j \Pr[X_j = 1] = \sum_{j=1}^m w_j x_j^* = Z_{PL}^*,$$

ou seja, o valor do PL, que não é maior que OPT! No entanto, veremos que é bem provável que a solução não seja uma cobertura por conjuntos. Mesmo assim, isso ilustra bem o motivo de aproximações aleatorizadas serem capazes de fornecer aproximações bastante boas em alguns casos.

Calculemos agora a probabilidade de que um dado elemento e_i não esteja coberto por tal procedimento. É a probabilidade de que nenhum dos conjuntos contendo e_i estejam na solução, isto, é,

$$\prod_{j: e_i \in S_j} (1 - x_j^*).$$

Podemos obter um limitante superior para essa probabilidade usando o fato de que $1 - x \leq e^{-x}$ para qualquer $x \geq 0$, onde e é a base do logaritmo natural. Então,

$$\begin{aligned} \Pr[e_i \text{ não estar coberto}] &= \prod_{j: e_i \in S_j} (1 - x_j^*) \\ &\leq \prod_{j: e_i \in S_j} e^{-x_j^*} \\ &= e^{-\sum_{j: e_i \in S_j} x_j^*} \\ &\leq e^{-1}, \end{aligned}$$

onde a última desigualdade vem da restrição do PL de que $\sum_{j: e_i \in S_j} x_j^* \geq 1$. Embora e^{-1} seja um limitante superior para a probabilidade de que um dado elemento não esteja coberto, é possível chegar arbitrariamente perto desse limitante, de modo que no pior caso é bem provável que esse procedimento de aproximação não produza uma cobertura por conjuntos.

Quão pequena essa probabilidade deve ser para que seja bastante provável que se produza uma cobertura? É mais importante, qual é a noção “correta” de “bastante provável”? Uma das possíveis maneiras de pensar nisso é impor uma garantia conforme nosso foco em algoritmos de tempo polinomial. Suponha que, para qualquer constante c , fôssemos capazes de elaborar um algoritmo polinomial cuja probabilidade de falha fosse no máximo uma polinomial inversa n^{-c} ; então, dizemos ter um algoritmo que funciona *com alta probabilidade*.

Mais precisamente, teríamos uma família de algoritmos, uma vez que pode ser necessário dar algoritmos progressivamente mais lentos ou com piores garantias de desempenho para obter resultados analogamente mais seguros. Se pudéssemos elaborar um procedimento aleatorizado tal que

$$\Pr[e_i \text{ não está coberto}] \leq \frac{1}{n^c}$$

para alguma constante $c \geq 2$, então

$$\begin{aligned} \Pr[\text{existe um elemento descoberto}] &\leq \sum_{i=1}^n \Pr[e_i \text{ descoberto}] \\ &\leq n \cdot \frac{1}{n^c} \\ &= \frac{1}{n^{c-1}}, \end{aligned} \tag{3.9}$$

e teríamos uma cobertura com alta probabilidade. De fato, conseguimos atingir tal limitante da seguinte forma: para cada S_j , imaginamos uma moeda que dá cara com probabilidade x_j^* , e nós lançamos a moeda $c \ln n$ vezes. Incluímos S_j na solução se, e somente se, obtivermos cara em um dos lançamentos da moeda. Então, a probabilidade de S_j não ser incluído é $(1 - x_j^*)^{c \ln n}$, e

$$\begin{aligned} \Pr[e_i \text{ descoberto}] &= \prod_{j: e_i \in S_j} (1 - x_j^*)^{c \ln n} \\ &\leq \prod_{j: e_i \in S_j} e^{-x_j^* (c \ln n)} \\ &= e^{(-c \ln n) \sum_{j: e_i \in S_j} x_j^*} \\ &\leq \frac{1}{n^c}, \end{aligned} \tag{3.10}$$

como desejávamos.

Provaremos que o algoritmo tem um bom valor esperado dado que produza uma cobertura por conjuntos.

Teorema 3.15. *O algoritmo é uma $O(\ln n)$ -aproximação aleatorizada que produz uma cobertura por conjuntos com alta probabilidade.*

Demonstração. Vejamos que o algoritmo é polinomial. Primeiro, resolve um PL, o que pode ser feito em tempo polinomial. Depois, para cada S_j , simula o lançamento de $c \ln n$ moedas, onde c é constante. Admitindo que cada lançamento seja feito em tempo constante, esse passo do algoritmo consome tempo $O(m \ln n)$.

Seja $p_j(x_j^*)$ a probabilidade de um dado S_j estar na solução em função de x_j^* . Pela construção do algoritmo, sabemos que

$$p_j(x_j^*) = 1 - (1 - x_j^*)^{c \ln n}.$$

Observe que, se $x_j^* \in [0, 1]$ e $c \ln n \geq 1$, então a derivada p_j' em x_j^* é limitada por

$$p_j'(x_j^*) = (c \ln n)(1 - x_j^*)^{(c \ln n) - 1} \leq (c \ln n).$$

Então, como $p_j(0) = 0$ e o crescimento da função p_j é limitado superiormente por $c \ln n$ no intervalo $[0, 1]$, segue que $p_j(x_j^*) \leq (c \ln n)x_j^*$ no intervalo $[0, 1]$. Se X_j é uma variável aleatória que vale 1, se o

conjunto S_j está na solução, e 0, caso contrário, então o valor esperado do procedimento aleatório é

$$\begin{aligned} E \left[\sum_{j=1}^m w_j X_j \right] &= \sum_{j=1}^m w_j \Pr[X_j = 1] \\ &\leq \sum_{j=1}^m w_j (c \ln n) x_j^* \\ &= (c \ln n) \sum_{j=1}^m w_j x_j^* = (c \ln n) Z_{PL}^*. \end{aligned}$$

No entanto, gostaríamos de limitar o valor esperado da solução dado que uma cobertura foi produzida. Seja F o evento de a solução obtida pelo procedimento ser uma cobertura, e seja \bar{F} seu complemento. Sabemos pela discussão anterior (em 3.9 e 3.10) que $\Pr[F] \geq 1 - \frac{1}{n^{c-1}}$, e também sabemos que

$$E \left[\sum_{j=1}^m w_j X_j \right] = E \left[\sum_{j=1}^m w_j X_j \middle| F \right] \Pr[F] + E \left[\sum_{j=1}^m w_j X_j \middle| \bar{F} \right] \Pr[\bar{F}].$$

Como $w_j \leq 0$ para todo j ,

$$E \left[\sum_{j=1}^m w_j X_j \middle| \bar{F} \right] \geq 0.$$

Assim,

$$\begin{aligned} E \left[\sum_{j=1}^m w_j X_j \middle| F \right] &= \frac{1}{\Pr[F]} \left(E \left[\sum_{j=1}^m w_j X_j \right] - E \left[\sum_{j=1}^m w_j X_j \middle| \bar{F} \right] \Pr[\bar{F}] \right) \\ &\leq \frac{1}{\Pr[F]} \cdot E \left[\sum_{j=1}^m w_j X_j \right] \\ &\leq \frac{1}{1 - \frac{1}{n^{c-1}}} \cdot E \left[\sum_{j=1}^m w_j X_j \right] \\ &= \frac{(c \ln n) Z_{PL}^*}{1 - \frac{1}{n^{c-1}}} \\ &\leq 2c(\ln n) Z_{PL}^* \end{aligned}$$

para $n \geq 2$ e $c \geq 2$, que satisfazem $(1 - \frac{1}{n^{c-1}}) \geq \frac{1}{2}$. □

Embora nesse caso haja uma aproximação mais simples e rápida com uma melhor garantia de desempenho, algoritmos aleatorizados são, por vezes, mais fáceis de descrever e analisar do que algoritmos determinísticos. Muitos algoritmos aleatorizados admitem uma variante determinística que atinge a garantia de desempenho esperada do algoritmo original. No entanto, em alguns casos a versão determinística é difícil de descrever. Também pode ocorrer de a única maneira conhecida de analisar um algoritmo é analisar sua versão aleatorizada.

Com isto, encerramos nossa introdução a algoritmos de aproximação.

4 Problemas de *clustering*

Nessa seção, trataremos do problema de particionar elementos em conjuntos (*clusters*) de acordo com suas semelhanças. Trata-se de um problema bastante genérico que aparece em diversas aplicações,

e por isso existem muitas variantes desse problema. A principal diferença entre elas é a função objetivo. Uma das formas mais simples desse problema é o chamado *k-center*, (ou problema dos *k-centros*) em que, dada uma coleção S de pontos com distâncias definidas para cada par de pontos, desejamos encontrar um conjunto $C = \{c_1, \dots, c_k\} \subset S$, cujos elementos chamamos de *centros*, de forma que a máxima distância de um ponto $p \in S$ ao centro mais próximo de p seja mínima. Note que nessa formulação temos implicitamente uma partição de S em k conjuntos (B_1, \dots, B_k) dada por $B_k = \{p \in S: \text{o centro mais próximo de } p \text{ é } c_k\}$.

Uma possível interpretação desse problema seria, por exemplo, a automatização de um processo de divisão em categorias. Pode-se imaginar, por exemplo, o uso de dados de compras de consumidores. Conforme os tipos e quantidades de produtos comprados, pode-se elaborar uma maneira de medir a “distância” entre dois consumidores. Nesse contexto, o problema dos k centros pode ser visto como o problema de estabelecer k “perfis de consumidor” de maneira que os consumidores incluídos em cada perfil sejam um grupo o menos heterogêneo possível. Poderíamos usar a mesma idéia para tentar encontrar formas de categorizar alunos de um curso, deputados de uma casa legislativa, etc., conforme os critérios de comparação e representações matemáticas que pareçam convenientes ou interessantes.

Outra interpretação para esse problema (e similares) é como um problema de *localização de instalações* (ou *facility location*). Os pontos representam um conjunto de locais que precisam ser supridos por certo tipo de instalações. Deseja-se abrir tais instalações em k dos pontos de maneira que nenhum ponto esteja muito longe da instalação mais próxima. Por exemplo, poderíamos ter um conjunto de hospitais públicos, e dispomos de recursos para abrir centros de tratamento de câncer em k deles. Desejamos saber em quais hospitais instalar tais centros de modo a minimizar a máxima distância que um paciente tenha que percorrer ao ser transferido de seu hospital para um dos que dispõem de centros de tratamento, caso seja necessário. A interpretação do *clustering* como um problema de localização de instalações leva a variantes do problema que levam em conta possibilidade de falha de algumas instalações, por exemplo.

A seguir, estudaremos algumas variações do problema e algoritmos de aproximação para resolvê-las.

4.1 Minimização da máxima distância entre elementos de um mesmo *cluster*

Veremos aqui alguns resultados para problemas de *clustering* estudados por Gonzalez em [3], adaptando e por vezes complementando seu conteúdo. Trataremos do problema de minimizar a máxima distância entre qualquer par de pontos que esteja dentro de um mesmo *cluster*; veremos a complexidade de algumas de suas variações e um algoritmo de aproximação.

Seja $G = (V, E, W)$ um grafo não direcionado com pesos não-negativos nas arestas dadas pela função $W: E \rightarrow \mathbb{R}_0^+$. Um *k-split* de G é uma partição de V em k conjuntos (B_1, B_2, \dots, B_k) . Cada B_i é um **cluster**. Cada *k-split* tem um valor $\text{val}(B_1, \dots, B_k)$ dado pela função objetivo “MM” (max-max), definida por:

$$\text{val}(B_1, \dots, B_k) = \max\{M_1, \dots, M_k\},$$

onde

$$M_i = \max\{W(e): e \in E \text{ tem as pontas em } B_i\}.$$

Em geral, pensaremos no problema de otimização $\min_{(B_1, \dots, B_k)} \text{val}(B_1, \dots, B_k)$.

Diremos que estamos tratando do *caso métrico* caso o grafo seja completo e suas arestas satisfaçam a desigualdade triangular. Isto é,

$$W(ij) \leq W(ik) + W(kj), \quad \forall i \neq j \neq k \in V.$$

Note que, na verdade, não precisamos exigir que o grafo seja completo – caso não seja, para cada $ij \notin E$ podemos incluir ij em E estabelecendo $W(ij) = d(i, j)$, onde $d(i, j)$ é o custo do menor caminho de i a j no grafo. Claramente, o grafo continua respeitando a desigualdade triangular.

Adotaremos a notação de [3] para as variações desse problema. Iremos nos referir ao problema de *clustering* α - β MM, onde α é o número de *clusters* que desejamos formar e β representa uma descrição do espaço em que estão os pontos. Se o número de *clusters* for um parâmetro de entrada para o problema, então escrevemos $\alpha = k$. Se os pontos a serem divididos em *clusters* estão em um espaço euclidiano de dimensão m , com a distância entre pontos definida pela distância euclidiana, então $\beta = m$. Se o problema está definido no caso métrico, escrevemos $\beta = t$. Se não há nenhuma suposição sobre o grafo em questão, ou seja, no caso mais geral possível, escrevemos $\beta = g$. O “MM”, finalmente, refere-se à nossa função objetivo, conforme já mencionamos.

4.1.1 Um algoritmo de aproximação para k - t MM

Seja S nosso conjunto de pontos e $d(i, j)$ a distância entre os pontos $i, j \in S$. Suponha que $|S| > k$, caso contrário o problema torna-se trivial — basta tomar $\{i\}$ como conjunto da partição para cada $i \in S$. Seja $\text{OPT}(S, k)$ o valor de um k -*split* ótimo para S .

Definição 4.1. *Um conjunto $T \subseteq S$ é uma clique se para todo par (i, j) de pontos distintos de T , existe uma aresta com extremos i e j .*

Note que, no caso métrico, todo subconjunto de S é uma clique.

Definição 4.2. *Uma clique $T \subseteq S$ é uma a -clique de peso h se $|T| = a$ e $\min_{i \neq j \in T} d(i, j) = h$.*

Lema 4.3. *Se S tem uma $(k + 1)$ -clique de peso h , então $\text{OPT}(S, k) \geq h$.*

Demonstração. Tome um k -*split* qualquer, digamos (B_1, \dots, B_k) .

Seja i , $1 \leq i \leq k$, tal que $|B_i \cap T| \geq 2$. Tal i existe pois $|T| = k + 1 > k$.

Sejam p, q elementos distintos em $B_i \cap T$. Segue da nossa hipótese que $d(i, j) \geq h$. Logo,

$$\text{val}(B_1, \dots, B_k) \geq h,$$

isto é, qualquer k -*split* tem valor no mínimo h . □

Considere o problema k - t MM. Isto é, o problema de particionar pontos em k conjuntos (onde k é um parâmetro do problema) de forma a minimizar o valor da partição conforme a função objetivo MM, no caso métrico. Propomos o seguinte algoritmo, bastante simples, para resolvê-lo.

Observe que o algoritmo começa colocando todos os pontos em B_1 e selecionando um elemento arbitrário como o centro c_1 desse conjunto. A cada passo ℓ , o algoritmo já tem os *clusters* B_1, \dots, B_ℓ , e cria um novo *cluster* com o seguinte critério: tome um ponto, digamos v , que esteja a distância máxima do centro c_j de seu conjunto B_j atual. Mova v para $B_{\ell+1}$, e tome-o como o centro $c_{\ell+1}$ desse *cluster*. Então, mova para $B_{\ell+1}$ todos os pontos que estão mais próximos de $v = c_{\ell+1}$ do que de seus respectivos centros atuais.

Teorema 4.4. *O algoritmo 4.1 é uma 2-aproximação para o k - t MM.*

Demonstração. Em primeiro lugar, o algoritmo é polinomial. A cada *cluster* criado, a lista dos pontos é percorrida para obter aquele que está a máxima distância de seu centro atual. Esse passo pode ser feito

Algoritmo 4.1: Algoritmo para o k -tMM.

Entrada: Conjunto S , distâncias d sobre S , natural k

Saída: k -split (B_1, \dots, B_k) de S

```
1  $B_1 \leftarrow S$ ;  
2  $c_1 \leftarrow$  elemento arbitrário em  $B_1$ ;  
3 para  $\ell \leftarrow 1$  até  $k - 1$  faça  
4    $h \leftarrow \max\{d(c_j, v_i) : v_i \in B_j, 1 \leq j \leq \ell\}$ ;  
5   Seja  $v_i$  tal que  $v_i \in B_j$  e  $d(d_i, c_j) = h$ ;  
6   Mova  $v_i$  para  $B_{\ell+1}$ ;  
7    $c_{\ell+1} \leftarrow v_i$ ;  
8   para cada  $v \in B_1 \cup \dots \cup B_\ell$  faça  
9     Seja  $j$  tal que  $v \in B_j$ ;  
10    se  $d(v, c_j) \geq d(v, c_{\ell+1})$  então  
11      | Mova  $v$  de  $B_j$  para  $B_{\ell+1}$ ;  
12    fim  
13  fim  
14 fim
```

em tempo $O(n)$, onde $n = |S|$. Após escolhido o novo centro, é necessário novamente percorrer os pontos para verificar quais deverão ser movidos para o novo conjunto. Esse passo também é $O(n)$. Portanto, a criação de um novo *cluster* custa tempo $O(n)$. O consumo de tempo do algoritmo, portanto, é $O(kn)$, que é $O(n^2)$, visto que $k \leq n$.

Além disso, é claro que o algoritmo gera um k -split. Resta verificar que o valor da solução gerada não é superior a $2 \cdot \text{OPT}(S)$.

Seja (B_1, \dots, B_k) a solução gerada pelo algoritmo. Seja $h = \max d(v, c_j) : v \in B_j$.

Se p e q são vértices no mesmo *cluster*, digamos B_j , e são tais que $\text{val}(B_1, \dots, B_k) = d(p, q)$, note que, como estamos no caso métrico, vale que

$$\text{val}(B_1, \dots, B_k) = d(p, q) \leq d(p, c_j) + d(c_j, q) \leq h + h = 2 \cdot h \quad (4.1)$$

Seja, agora v^* um vértice tal que $d(v^*, c_{j^*}) = h$, onde j^* é tal que $v^* \in B_{j^*}$, isto é, v^* está a distância máxima de seu centro c_{j^*} . Observe que, como v^* não é o centro de seu conjunto, nenhum passo do algoritmo escolheu v^* como novo centro. Ou seja, todo centro, ao ser escolhido, estava a distância *no mínimo* h de seu centro. Além disso, v^* está a distância pelo menos h de cada centro, visto que está a essa distância do centro mais próximo.

Portanto, $\{v^*, c_1, \dots, c_k\}$ é uma $(k+1)$ -clique de peso h . Segue de 4.3, portanto, que $\text{OPT}(S, k) \geq h$. Disso e 4.1, temos que

$$\text{val}(B_1, \dots, B_k) \leq 2 \cdot h \leq 2 \cdot \text{OPT}(S, k).$$

□

Antes de prosseguir, vejamos que a análise é justa. Mostraremos que é possível gerar instâncias para as quais a razão entre o valor ótimo e o valor da solução dada por 4.1 seja tão próxima de 2 quanto se queira.

Considere a seguinte classe de instâncias do k -1MM (*clustering* de pontos na reta real), que é um caso particular do k -tMM:

- $S = \{0, 1, 2 - \varepsilon, 3\}$, onde $0 < \varepsilon < 1$,
- $d(i, j)$ é a distância euclidiana entre i e j (no caso 1-dimensional, $d(i, j) = |i - j|$), e
- $k = 2$

Suponha que no primeiro passo o algoritmo tome o ponto 1 como c_1 . No passo seguinte, o ponto mais distante de c_1 é o ponto 3, que é então tomado como o centro c_2 do conjunto B_2 . O algoritmo termina com:

- $B_1 = 0, 1, 2 - \varepsilon, B_2 = 3$,
- $\text{val}(B_1, B_2) = d(2 - \varepsilon, 0) = 2 - \varepsilon$.

Note que o valor ótimo para essa instância é $\text{val}(\{0, 1\}, \{2 - \varepsilon, 3\}) = 1 + \varepsilon$. Portanto, a razão entre o valor da solução do algoritmo e o valor ótimo é $\frac{2 - \varepsilon}{1 + \varepsilon}$.

4.1.2 Complexidade dos problemas de *clustering*

Demos um algoritmo de aproximação para o k -tMM, mas até agora não nos perguntamos se esse problema, ou pelo menos o k -gMM, que é mais geral, são NP-difíceis. Pelo que discutimos até agora, poderia ser que uma solução exata para o problema pudesse ser atingível em tempo polinomial mesmo que $P \neq NP$, e nesse caso nossa aproximação não teria muito valor.

Mas, como veremos agora, é verdade que esses problemas são NP-difíceis. Daremos uma prova de que a versão de decisão do k -tMM é um problema NP-completo, conforme a que pode ser vista em [3]. Disso segue, como veremos, que a versão de otimização de k -tMM é NP-difícil. O mesmo vale para k -gMM, claro, por ser uma generalização daquele.

Mais que isso, veremos, ainda, que o problema da $(2 - \varepsilon)$ -aproximação para o k -tMM é NP-difícil para qualquer $\varepsilon > 0$. Ou seja, o algoritmo 4.1 é o melhor possível para esse problema, a menos que $P = NP$.

Considere o problema da cobertura exata por conjuntos de tamanho 3, que chamaremos de XC3:

Dados um conjunto finito de elementos $X = \{x_1, x_2, \dots, x_{3q}\}$ e uma coleção de subconjuntos de X de tamanho 3, $C = Y \subseteq X$: $|Y| = 3$, com $|C| = m$, e nenhum elemento de X aparece em mais do que 3 elementos de C . O problema XC3 é decidir se C contém uma cobertura exata de X , isto é, um $C' \subseteq C$ tal que cada elemento de X ocorre em exatamente um elemento de C' .

Sabe-se que esse problema é NP-completo [2]. Considere agora a seguinte variação de XC3. Chamaremos esse novo problema de RXC3: a versão *restrita* de XC3.

No problema RXC3, é dada uma instância (X, C) de XC3 tal que cada elemento de X ocorre em *exatamente* 3 dos conjuntos na coleção C . O problema é, como em XC3, decidir se C contém uma cobertura exata de X .

O problema RXC3 também é NP-completo: em [3] é mostrada uma redução polinomial de XC3 a RXC3. Mostraremos que RXC3 tem uma redução polinomial à versão de decisão de k -tMM.

O problema de decisão k -tMM consiste em, dados um conjunto de pontos (vértices) S com distância $d(p, q)$ definida para cada par de vértices e obedecendo a desigualdade triangular, um inteiro positivo k e um real não negativo w , decidir se S admite um k -split de peso não superior a w .

Teorema 4.5. *O problema de decisão k -tMM é NP-completo.*

Demonstração. É fácil ver que o problema de decisão k -tMM está em NP. Dada uma instância (S, d, k, w) para o problema, um certificado para “sim” é um k -split (B_1, \dots, B_k) de valor $\text{val}(B_1, \dots, B_k) \leq w$. Tal certificado tem tamanho polinomial no tamanho da entrada, e claramente pode ser verificado em tempo polinomial.

Veremos agora que RXC3 tem uma redução polinomial ao k -tMM.

Seja (X, C) uma instância do RXC3, com $|X| = 3q$, com q inteiro, e $|C| = m$. Considere a seguinte instância do k -tMM:

- O conjunto de vértices, S , contém um vértice v_i para cada $x_i \in X$, e cada $Y \in C$ introduz 9 novos vértices. Se $Y = \{v_i, v_j, v_k\}$, a seguinte estrutura representa Y :
- As distâncias d são dadas da seguinte maneira: as arestas da representação de um $Y \in C$, conforme acima, têm peso 1. As demais arestas têm peso 2;
- $k = 3m + q$; e
- $w = 1$.

Tal construção pode ser feita em tempo polinomial no tamanho da instância (X, C) .

Para completar nossa prova, basta mostrar que S admite um k -split de valor ≤ 1 se, e somente se, C contém uma cobertura exata para X .

(\Leftarrow) Seja $C' \subseteq C$ cobertura exata de X . Para cada $Y \in C'$, o subgrafo que representa Y é agrupado em 4 *clusters*, conforme na figura

Para cada $Y \in C \setminus C'$, o subgrafo correspondente é agrupado em 3 *clusters*, conforme na figura

Note que esses *clusters* dão uma partição de S , isto é, cada vértice é incluído em exatamente um *cluster* por essa regra.

Cada elemento em C contribui com 3 *clusters*, exceto aqueles em C' , que adicionam mais 1 *cluster* cada.

Como C' é uma cobertura exata de X e todos os seus elementos são 3-conjuntos (conjuntos de tamanho 3), vale que $|C'| = |X|/3$.

Então, o número de *clusters* criados é

$$3 \cdot |C| + 1 \cdot |C'| = 3m + \frac{|X|}{3} = 3m + q = k.$$

Finalmente, em todo *cluster* vale que os vértices estão conectados apenas por arestas de peso 1. Portanto, obtivemos um k -split de S com peso $1 \leq w$.

(\Rightarrow) Seja (B_1, \dots, B_k) um k -split de S com valor $\text{val}(B_1, \dots, B_k) \leq w = 1$.

Note que, na nossa instância do k -tMM, não há nenhum conjunto de 4 vértices totalmente conectados por arestas de peso 1. Além disso, $|V| = 3q + 9m = 3(q + 3m) = 3k$.

Então, cada B_i tem cardinalidade $|B_i| = 3$. Pela regra de construção dessa instância, todo subgrafo que representa um $Y \in C$ está agrupado, pelo nosso k -split, como em

Como todo vértice está em exatamente um *cluster*, para que tenhamos k *clusters*, certamente há exatos q subgrafos representando elementos em C que estão agrupados como em e cada um deles deve incluir v_ℓ (representantes dos x_ℓ) todos diferentes, uma vez que cada vértice desses só está em *clusters* dos subgrafos como em

Dessas observações, segue que, se tomarmos os $Y \in C$ representados por subgrafos que estão agrupados como em teremos uma cobertura exata de X . \square

Esse resultado tem duas consequências imediatas.

Corolário 4.6. *O problema de otimização k -tMM é NP-difícil.*

Corolário 4.7. *O problema de otimização k -gMM é NP-difícil.*

Outro resultado segue não do Teorema 4.5, mas da redução usada em sua prova.

Teorema 4.8. *O problema da $(2 - \varepsilon)$ -aproximação para o k -tMM é NP-difícil para todo $\varepsilon > 0$.*

Demonstração. Seja (X, C) uma instância do RXC3. Seja (G, d, k) a instância do k -tMM obtida pela mesma redução usada na prova do Teorema 4.5.

O valor ótimo para (G, d, k) é 1, se a resposta do RXC3 para (X, C) é “sim”, e 2, caso contrário.

Fixe um $\varepsilon > 0$. Uma $(2 - \varepsilon)$ -aproximação daria, para a instância (G, d, k) , portanto, uma resposta exata: se o valor ótimo da instância é 2, então o algoritmo daria uma resposta de valor ≥ 2 , que só pode ser, de fato, 2, uma vez que não há aresta com custo maior que 2; e se o valor ótimo da instância é 1, o algoritmo obrigatoriamente forneceria uma solução de valor $\leq (2 - \varepsilon) \cdot 1 < 2$, que no caso só pode ser, de fato, 1 (pois toda aresta tem valor 1 ou 2).

Dessa maneira, nossa $(2 - \varepsilon)$ -aproximação resolveria, em tempo polinomial, o problema RXC3, que é NP-completo: dada a resposta do algoritmo para a instância (G, d, k) , conseguimos determinar a resposta do problema RXC3 para a instância (X, C) .

Ou seja, o problema da $(2 - \varepsilon)$ -aproximação para o k -tMM é NP-difícil. \square

4.2 Problemas de k -centros com tolerância a falhas

Nessa seção, e possivelmente outras a serem adicionadas, estudaremos a variante do problema em que pensamos na localização de instalações com tolerância a falhas. Serão apresentados diversos resultados, como em [5] e outras referências a serem adicionadas.

5 Conclusões

Seção em que serão apresentadas as conclusões diante do trabalho realizado.

Referências

- [1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT Press, 3 edition, 2009.
- [2] Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness*. Freeman, 1980.
- [3] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- [4] Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.
- [5] Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242:237–245, 2000.
- [6] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

Parte II

Parte Subjetiva

6 Desafios e frustrações

Nesta seção será descrito o desenvolvimento do trabalho de formatura, e os principais desafios enfrentados no processo.

7 Relação entre o curso e o trabalho de formatura

Esta seção será dedicada a uma descrição da relação entre o BCC e o trabalho de formatura, incluindo as disciplinas consideradas cruciais para o desenvolvimento do TCC, etc.

8 Trabalhos futuros

Seção em que se descreverá a perspectiva para continuação do trabalho iniciado com esse TCC.