Análise de algorimos de aproximação utilizando o método dual-fitting

Leonardo Marchetti (No. USP: 5639102)

8 de fevereiro de 2010

Sumário

1 Introdução

Neste trabalho iremos descrever um sofisticado método para analisar algoritmos de aproximação para problemas de otimização combinatória chamado *dual-fitting* e apresentar algumas aplicações desse método. Começaremos introduzindo conceitos necessários para o entendimento do método assim como a notação que adotaremos, inspirada na adotada no livro *Uma Introdução Sucinta a Algoritmos de Aproximação* [3].

Problema de otimização

Um *problema de otimização* possui três ingredientes principais: um conjunto de *instâncias*, um conjunto $Sol(I)$ de *soluções* para cada instância I, e uma função que atribui um número $val(S)$ a cada solução S. O número $val(S)$ é o *valor* de S. Um problema de otimização pode ser de *minimização* ou de *maximização*. Um problema de minimização está interessado nas soluções de valor mínimo, enquanto um problema de maximização está interessado nas soluções de valor máximo.

Quando o conjunto $Sol(I)$ associado a uma instância I é vazio, dizemos que a instância é *inviável*; caso contrário, a instância é dita *viável*. Uma solução cujo valor é mínimo/máximo é chamada de *solução ótima* do problema de minimização/maximização. O valor de qualquer uma das soluções ótimas de uma instância I será denotado por $opt(I)$. Portanto

$$opt(I) := val(S^*),$$

onde S^* é uma solução ótima de I. É claro que esse número não está definido se a instância I é inviável. Ele também não está definido caso não exista uma solução ótima. Isso acontece, por exemplo, no caso de um problema de minimização, se para todo $S \in Sol(I)$ existe $S' \in Sol(I)$ tal que $val(S') < val(S)$.

Algoritmo de aproximação

Considere um problema de otimização em que $val(S) \geq 0$ para toda solução S de qualquer instância viável I do problema. Seja A um algoritmo que, para toda instância viável I do problema, devolve em tempo polinomial no tamanho de I uma solução $A(I)$ do problema para a instância I. Se o problema é de minimização e

$$val(A(I)) \leq \alpha \cdot opt(I) \tag{1}$$

para toda instância I viável, dizemos que A é uma *α-aproximação* para o problema. O fator α é um número que pode depender de I. Dizemos que α é uma *razão de aproximação* do algoritmo. É claro que $\alpha \geq 1$, uma vez que o problema é de minimização. No caso de um problema de maximização, a definição fica com

$$val(A(I)) \geq \alpha \cdot opt(I)$$

no lugar de (1). É claro que nesse caso $0 < \alpha \leq 1$. Um algoritmo de aproximação é uma α-aproximação para algum α. Uma 1-aproximação para um problema de otimização é um algoritmo polinomial exato para o problema.

Complexidade computacional

Existem centenas de problemas de otimização para os quais não são conhecidos algoritmos polinomiais exatos. Mais do que isso, existem centenas de problemas de otimização que são ditos *NP-difíceis*. O que isso quer dizer?

Em vez de darmos a definição formal deste conceito, passaremos apenas a ideia por trás dele, pois é o suficiente para continuarmos nossa exposição.

Se π é um problema de otimização NP-difícil e projetarmos um algoritmo polinomial exato para π, então esse algoritmo pode ser transformado em um algoritmo polinomial exato para qualquer outro problema de otimização NP-difícil. Ou seja, resolver eficientemente um problema NP-difícil é equivalente a resolver eficientemente todos os problemas NP-difíceis simultaneamente! Por isso tais problemas são considerados difíceis e acredita-se que não exista algoritmo polinomial exato para eles.

Uma maneira de se lidar com problemas de otimização NP-difíceis é por meio de algoritmos de aproximação. Como é improvável que exista um algoritmo polinomial exato para esses problemas, buscam-se bons algoritmos de aproximação para eles.

Programas lineares

Um *problema de programação linear*, ou *programa linear*, consiste no seguinte: dada uma matriz A indexada por $M \times N$, um vetor b indexado por M, um vetor c indexado por N e partições $\{M_1, M_2, M_3\}$ e $\{N_1, N_2, N_3\}$ de M e N

respectivamente, encontrar um vetor x indexado por N que

$$\begin{aligned} \text{minimize } cx & \\ \text{sob as restrições } (Ax)_i &\geq b_i \text{ para cada } i \text{ em } M_1, \\ (Ax)_i &= b_i \text{ para cada } i \text{ em } M_2, \\ (Ax)_i &\leq b_i \text{ para cada } i \text{ em } M_3, \\ x_j &\geq 0 \text{ para cada } j \text{ em } N_1, \\ x_j &\leq 0 \text{ para cada } j \text{ em } N_3. \end{aligned}$$

Veja que cx é o produto interno entre os vetores c e x, ambos indexados por N. Ou seja, $cx = \sum_{i \in N} c_i x_i$.

Usaremos a abreviatura $P(A, b, c)$ para denotar esse programa linear; a sequência de conjuntos $M_1, M_2, M_3, N_1, N_2, N_3$ fica subentendida nessa notação. Embora o programa linear tenha sido formulado como um problema de minimização, nossa definição inclui, implicitamente, problemas de maximização, uma vez que maximizar cx é equivalente a minimizar $-cx$.

Quando A, b e c estão claros do contexto, escrevemos P em vez de $P(A, b, c)$. Uma *solução* do programa linear P é um vetor x indexado por N que satisfaz as restrições de P; o *valor* de uma solução x é o número cx. Com isso, podemos pensar no programa linear como um problema de otimização.

Para um vetor x indexado por N, denotamos por $V(P, x)$ o número cx. Quando x é uma solução de P, vale então que $V(P, x)$ é o valor de x em P.

O conjunto de todas as soluções do problema $P(A, b, c)$ será denotado por $X(A, b)$. Se $X(A, b)$ é *vazio*, dizemos que o problema é *inviável*. Caso contrário, o problema é *viável*. Dizemos que o problema é *limitado* se existe um número ω tal que $cx \geq \omega$ para todo x em $X(A, b)$; caso contrário, o problema é *ilimitado*.

Assim, de maneira análoga ao que fizemos com problemas de otimização, se P é viável e limitado, definimos *solução ótima* de P e *valor ótimo* de P, denotado por $opt(P)$. Problemas inviáveis e ilimitados não têm solução ótima.

Dualidade

Há uma importante relação de dualidade entre programas lineares. O *dual* do programa linear $P(A, b, c)$ é o problema de programação linear $P(-A^T, -c, -b)$, onde A^T é a transposta da matriz A e os conjuntos de índices são $N_1, N_2, N_3, M_1, M_2, M_3$. É claro que esse programa linear também pode

ser escrito assim: encontrar um vetor y indexado por M que

$$\begin{aligned} \text{maximize } & yb \\ \text{sob as restrições } (yA)_j &\leq c_j \text{ para cada } j \text{ em } N_1, \\ (yA)_j &= c_j \text{ para cada } j \text{ em } N_2, \\ (yA)_j &\geq c_j \text{ para cada } j \text{ em } N_3, \\ y_i &\geq 0 \text{ para cada } i \text{ em } M_1, \\ y_i &\leq 0 \text{ para cada } i \text{ em } M_3. \end{aligned}$$

Convém usar uma notação específica para o dual: o programa linear acima será denotado por $D(A, c, b)$ e seu conjunto de soluções viáveis por $Y(A, c)$. O valor de uma solução y de $D(A, c, b)$ é yb.

Em discussões sobre um par dual de programas lineares, é comum dizer que um deles é "o primal" e o outro é "o dual". Adotada essa convenção, um vetor x indexado por N é um "vetor primal" e um vetor y indexado por M é um "vetor dual".

Iremos considerar o caso em que o programa linear considerado primal é um problema de minimização para enunciar o *Teorema Fraco da Dualidade* e o *Teorema Forte da Dualidade*.

O Teorema Fraco da Dualidade nos diz que o valor de qualquer solução do programa dual é sempre menor ou igual ao valor de qualquer solução do programa primal. Já o Teorema Forte da Dualidade diz que se o primal e o dual são viáveis, então ambos possuem solução ótima e o valor ótimo dos dois programas lineares é o mesmo.

Programas lineares inteiros

Um programa linear *inteiro* é um programa linear $P(A, b, c)$ onde, adicionalmente, exige-se que algumas coordenadas específicas das soluções sejam inteiras.

É muito comum formular um problema de otimização como um programa linear inteiro. O programa linear obtido de um tal programa linear inteiro ignorando-se as restrições de integralidade é uma *relaxação linear* do problema de otimização. Relaxações lineares de problemas de otimização são muito usadas no projeto de algoritmos de aproximação para tais problemas. Isso porque programas lineares podem ser resolvidos em tempo polinomial, por exemplo pelo *método dos elipsóides* [6].

2 Dual-Fitting

O método *dual-fitting* nos permite analisar alguns algoritmos de aproximação utilizando a teoria da dualidade de programação linear. Para um problema de minimização, o método pode ser descrito da seguinte maneira. Note que a definição para problemas de maximização é análoga.

Considere, para uma dada instância I do problema, um programa linear $P(I)$ que é uma relaxação de um programa linear inteiro equivalente ao problema para essa instância, e seu dual $D(I)$. Qualquer solução do problema para esta instância corresponde a uma solução de $P(I)$ de mesmo valor. Sendo assim, $opt(P(I)) \leq opt(I)$. Se $D(I)$ for viável, então pelo Teorema Forte da Dualidade, os valores das soluções ótimas de $P(I)$ e de $D(I)$ são iguais. Neste caso, portanto, $opt(D(I)) = opt(P(I)) \leq opt(I)$. Em outras palavras, o valor ótimo do dual do programa linear correspondente à relaxação do problema é uma delimitação inferior para o valor ótimo do problema.

Todo algoritmo para o problema original no fundo devolve uma solução de $P(I)$ (não necessariamente ótima). Em algumas situações, o algoritmo determina implicitamente um vetor dual y que é candidato a solução de $D(I)$ relacionado de alguma maneira à solução de $P(I)$ devolvida.

Se o candidato à solução dual definido pelo algoritmo for de fato uma solução de $D(I)$ e seu valor for pelo menos o valor da solução primal devolvida, teríamos que a solução primal devolvida pelo algoritmo seria ótima, e portanto a solução devolvida pelo algoritmo seria ótima para o problema original. Não é o que geralmente acontece. Mas se para algum número positivo c mostrarmos que $\frac{y}{c}$ é uma solução do dual, onde y é o candidato à solução dual definido pelo algoritmo, e que o valor de y é pelo menos o valor da solução de $P(I)$ devolvida, o algoritmo é uma c-aproximação como mostraremos a seguir.

Seja x a solução primal definida pelo algoritmo e y o candidato à solução dual definido implicitamente pelo algoritmo para uma instância I qualquer. Seja ainda y' definido como $y' = \frac{y}{c}$.

Se mostrarmos que y' é uma solução do problema dual, temos:

$$V(D(I), y') \leq opt(D(I)) = opt(P(I)) \leq opt(I).$$

Finalmente, se mostrarmos que $V(P(I), x) \leq V(D(I), y)$, podemos concluir que

$$V(P(I), x) \leq V(D(I), y) = c \cdot V(D(I), y') \leq c \cdot opt(I).$$

Portanto, o algoritmo é uma c-aproximação para o problema.

A ideia do método dual-fitting portanto é, para um dado algoritmo, detectar se é possível encontrar, juntamente com a solução do problema, um candidato a solução dual e uma boa constante c que satisfaça as condições acima.

Prosseguimos mostrando a aplicação do método a alguns problemas específicos. Os dois primeiros exemplos são tirados do capítulo 13 do livro de Vazirani [8].

3 Cobertura mínima por conjuntos

Para apresentarmos o *Problema da Cobertura Mínima por Conjuntos*, doravante também chamado de MinCC, são necessárias algumas definições.

Seja E um conjunto finito e $\mathcal{S}$ uma coleção de subconjuntos de E. Uma *cobertura* de E em $\mathcal{S}$ é um subconjunto τ de $\mathcal{S}$ tal que para todo $e \in E$ temos que $e \in S$ para algum $S \in \tau$.

Se τ é uma cobertura de E em $\mathcal{S}$ e temos um custo c tal que $c_S \in \mathbb{Q}_{\geq}$ para cada $S \in \mathcal{S}$, definimos $c(\tau)$ como o número $\sum_{S \in \tau} c_S$.

Agora podemos definir o problema MinCC como visto abaixo:

Problema MinCC $(E, \mathcal{S}, c)$: *Dados um conjunto finito E, uma coleção finita $\mathcal{S}$ de subconjuntos não-vazios de E (que cobre E) e um custo $c_S \in \mathbb{Q}_{\geq}$ para cada $S \in \mathcal{S}$, encontrar uma cobertura τ de E em $\mathcal{S}$ que minimize $c(\tau)$.*

3.1 Algoritmo

O algoritmo guloso proposto por Chvátal [4] para este problema pode ser escrito como:

Algoritmo MinCC-Chvátal $(E, \mathcal{S}, c)$

1. $\mathcal{S}' \leftarrow \mathcal{S}$
2. $E' \leftarrow E$
3. $\tau \leftarrow \emptyset$
4. enquanto $E' \neq \emptyset$
5. seja Z em $\mathcal{S}'$ tal que $c_Z/|Z \cap E'|$ é mínimo
6. $E' \leftarrow E' \backslash Z$
7. $\mathcal{S}' \leftarrow \{S \in \mathcal{S}' : S \cap E' \neq \emptyset\}$
8. $\tau \leftarrow \tau \cup \{Z\}$
9. devolva τ

É fácil ver que o seguinte invariante vale no início de cada iteração do enquanto da linha 4: a coleção τ é uma cobertura de $E \setminus E'$ em $\mathcal{S}$. Como ao final da execução do algoritmo $E' = \emptyset$, temos que a coleção τ devolvida é de fato uma cobertura de E em $\mathcal{S}$.

A cada iteração do algoritmo MinCC-Chvátal, pelo menos um elemento é retirado de E', já que $\mathcal{S}'$ no início da iteração possui apenas conjuntos que

possuem intersecção não-vazia com E'. Sendo assim, o bloco de linhas 4-8 é executado $O(|E|)$ vezes. Podemos pensar em uma implementação que representa cada S em $\mathcal{S}$ como um vetor booleano indexado por E, e mantém, para cada S em $\mathcal{S}$, contadores do número de elementos de E' que pertencem a S. O conjunto E' também pode ser representado como um vetor booleano indexado por E. Com isso, a linha 5 consome tempo $O(|\mathcal{S}|)$, a linha 6 consome $O(|E|)$, a linha 7, que pode incluir a atualização dos contadores, consome $O(|\mathcal{S}| \cdot |E|)$, e finalmente a linha 8 consome $O(1)$. Com essa implementação, o consumo de tempo do algoritmo é $O(|\mathcal{S}| \cdot |E|^2)$, que é polinomial no tamanho da entrada que é $\Omega(|\mathcal{S}| + |E|)$.

3.2 Razão de aproximação

Teorema 1: *O algoritmo* MinCC-Chvátal *é uma H_n-aproximação para o problema* MinCC $(E, \mathcal{S}, c)$*, onde* $n = |E|$.

Demonstração: Vamos provar o teorema através do método *dual-fitting*.

Considere o programa linear inteiro $PI(E, \mathcal{S}, c)$ que é equivalente ao MinCC para qualquer instância viável: encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{array}{rl} \text{minimize} & cx \\ \text{sob as restrições} & \sum_{S:e\in S} x_S \geq 1 \text{ para todo } e \in E, \\ & x_S \in \{0,1\} \text{ para todo } S \in \mathcal{S}. \end{array}$$

Seja $P(E, \mathcal{S}, c)$ um programa linear que é a relaxação linear de $PI(E, \mathcal{S}, c)$. Então, $P(E, \mathcal{S}, c)$ consiste em encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{array}{rl} \text{minimize} & cx \\ \text{sob as restrições} & \sum_{S:e\in S} x_S \geq 1 \text{ para todo } e \in E, \\ & x_S \geq 0 \text{ para todo } S \in \mathcal{S}. \end{array}$$

O dual $D(E, \mathcal{S}, c)$ de $P(E, \mathcal{S}, c)$ pode ser escrito como: encontrar um vetor y indexado por E que

$$\begin{array}{rl} \text{maximize} & \sum_{e\in E} y_e \\ \text{sob as restrições} & \sum_{e\in S} y_e \leq c_S \text{ para todo } S \in \mathcal{S}, \\ & y_e \geq 0 \text{ para todo } e \in E. \end{array}$$

O algoritmo MinCC-Chvátal define uma solução primal x, e mostraremos um candidato a solução dual y adequado.

A solução primal x é dada pelos conjuntos escolhidos pelo algoritmo. Se τ é a cobertura devolvida pelo algoritmo, $x_S = 1$ se $S \in \tau$ e $x_S = 0$ caso

contrário.

Para apresentarmos o candidato a solução dual y, precisamos de algumas definições.

Para um elemento $e \in E$, considere a iteração em que e é removido de E' na linha 6. Seja $C_e = E \setminus E'$ para E' no início desta iteração e S_e o conjunto Z escolhido nesta iteração. Observe que $\tau = \{S_e : e \in E\}$.

O candidato a solução dual é definido para cada e em E como

$$y_e = \frac{c_{S_e}}{|S_e \setminus C_e|}.$$

Este candidato a solução dual em geral não é uma solução dual, pois não satisfaz a restrição $\sum_{e \in S} y_e \leq c_S$.

Porém, o valor desta solução é pelo menos o valor da solução primal x encontrada pelo algoritmo. De fato, para cada $S \in \tau$, considere a iteração em que S é o conjunto Z escolhido na linha 5 do algoritmo e seja $C_S = E \setminus E'$ para E' no início dessa iteração. Então

$$\begin{aligned} c_S &= \sum_{e \in S \setminus C_S} \frac{c_S}{|S \setminus C_S|} \\ &= \sum_{e \in S \setminus C_S} \frac{c_{S_e}}{|S_e \setminus C_e|} \qquad (2) \\ &= \sum_{e \in S \setminus C_S} y_e, \end{aligned}$$

onde (2) vale pois, para todo $e \in S \setminus C_S$, temos que $S_e = S$ e $C_e = C_S$. De fato cada $e \in S \setminus C_S$ é removido de E' na linha 6 exatamente nessa iteração.

Então,

$$V(P(E, \mathcal{S}, c), x) = \sum_{S \in \tau} c_s = \sum_{e \in E} y_e = V(D(E, \mathcal{S}, c), y).$$

Considere agora o vetor y' indexado por E tal que $y'_e = y_e / H_n$ para cada e em E, onde H_n é o n-ésimo número harmônico. Vamos mostrar que y' é uma solução do problema dual $D(E, \mathcal{S}, c)$, ou seja, que $\sum_{e \in S} y'_e \leq c_S$ para todo $S \in \mathcal{S}$, já que sabemos que $y'_e \geq 0$ para todo $e \in E$ pois $y_e \geq 0$ para todo $e \in E$.

Seja $S \in \mathcal{S}$, e sejam $e_1, e_2, \ldots, e_k$ os elementos de S na ordem em que são cobertos pelo algoritmo, ou seja, que saem do conjunto E'.

No início da iteração em que o elemento e_i é coberto, existem pelo menos $k-i+1$ elementos descobertos em S. Assim y_{e_i} é no máximo $c_S/(k-i+1)$. De fato, como o algoritmo escolhe sempre um conjunto Z tal que $c_Z/|Z \cap E'|$ é mínimo e $\frac{c_S}{|S \cap E'|} \leq \frac{c_S}{k-i+1}$, temos que $y_{e_i} \leq c_S/(k-i+1)$ e portanto,

$$y'_{e_i} \leq \frac{c_S}{(k-i+1)H_n}.$$

Mas então

$$\sum_{e \in S} y'_e \leq \sum_{i=1}^{k} \frac{c_S}{(k-i+1)H_n} = \frac{c_S}{H_n} \sum_{i=1}^{k} \frac{1}{k-i+1} = c_S \frac{H_k}{H_n} \leq c_S.$$

Portanto, y' é uma solução do problema dual $D(E, \mathcal{S}, c)$ e consequentemente,

$$V(D(E, \mathcal{S}, c), y') \leq opt(D(E, \mathcal{S}, c)) = opt(P(E, \mathcal{S}, c)) \leq opt(E, \mathcal{S}, c).$$

Finalmente, podemos concluir que o custo da cobertura τ devolvida pelo algoritmo MinCC-Chvátal é tal que

$$c(\tau) = V(P(E, \mathcal{S}, c), x) \leq V(D(E, \mathcal{S}, c), y).$$

Como temos $V(D(E, \mathcal{S}, c), y) = H_n \cdot V(D(E, \mathcal{S}, c), y')$, pois a função objetivo de $D(E, \mathcal{S}, c)$ é linear, segue que

$$c(\tau) \leq H_n \cdot V(D(E, \mathcal{S}, c), y') \leq H_n \cdot opt(E, \mathcal{S}, c)$$

e o algoritmo é uma H_n-aproximação para o problema.

□

4 Multi-cobertura mínima por conjuntos

Para apresentarmos o *Problema da Multi-Cobertura Mínima por Conjuntos Restrita*, doravante também chamado de MinMCCR, são necessárias algumas definições.

Seja E um conjunto finito e $\mathcal{S}$ uma coleção de subconjuntos de E. Seja ainda r um vetor indexado por E que associa a cada elemento $e \in E$ um número não-negativo r_e que é a quantidade de vezes que e deve ser coberto. Uma *multi-cobertura* de E em $\mathcal{S}$ sujeita a r é um subconjunto τ de $\mathcal{S}$ tal que, para todo e em E,

$$|\{S \in \tau : e \in S\}| \geq r_e.$$

Observe que uma multi-cobertura de E em $\mathcal{S}$ sujeita a r é um *conjunto* de elementos de $\mathcal{S}$ e não um *multi-conjunto* de elementos de $\mathcal{S}$. O adjetivo "restrita" no nome do problema vem deste fato.

Se temos um custo c tal que $c_S \in \mathbb{Q}_{\geq}$ para cada $S \in \mathcal{S}$ como antes, $c(\tau)$ denota o número $\sum_{S \in \tau} c_S$.

Agora podemos definir o problema MinMCCR como visto abaixo:

Problema MinMCCR $(E, \mathcal{S}, c, r)$: *Dados um conjunto finito E, um vetor r, indexado por E, que associa a cada elemento $e \in E$ um número positivo r_e de vezes que e deve ser coberto, uma coleção finita $\mathcal{S}$ de subconjuntos não-vazios de E, que é uma multi-cobertura de E em $\mathcal{S}$ sujeita a r, e um custo $c_S \in \mathbb{Q}_{\geq}$ para cada $S \in \mathcal{S}$, encontrar uma multi-cobertura τ de E em $\mathcal{S}$ sujeita a r que minimize $c(\tau)$.*

4.1 Algoritmo

Para esse problema, é possível escrever um algoritmo guloso similar ao que mostramos para o MinCC. Este algoritmo, a cada iteração, para cada conjunto $S \in \mathcal{S}$, considera uma relação custo-benefício, que depende do custo do conjunto S e de quantos elementos de S não estão cobertos, ou seja, estão em E' no início da iteração, e então escolhe o conjunto de melhor custo-benefício.

Algoritmo MinMCCR-Guloso $(E, \mathcal{S}, c, r)$

1. $\mathcal{S}' \leftarrow \mathcal{S}$
2. $E' \leftarrow E$
3. $r' \leftarrow r$
4. $\tau \leftarrow \emptyset$

5. enquanto $E' \neq \emptyset$
6. seja Z em $\mathcal{S}'$ tal que $c_Z/|Z \cap E'|$ é mínimo
7. para cada e em $Z \cap E'$ faça
8. $r'_e \leftarrow r'_e - 1$
9. $E' \leftarrow \{e \in E' : r'_e > 0\}$
10. $\mathcal{S}' \leftarrow \mathcal{S}' \setminus \{Z\}$
11. $\mathcal{S}' \leftarrow \{S \in \mathcal{S}' : S \cap E' \neq \emptyset\}$
12. $\tau \leftarrow \tau \cup \{Z\}$
13. devolva τ

Seja $r''(e) = r(e) - r'(e)$ para todo e em E. É fácil ver que os seguintes invariantes são mantidos:

- τ é uma multi-cobertura de E em $\mathcal{S}$ sujeita a r''.
- $r'_e = 0$ para todo e em $E \setminus E'$.

Pela linha 10, um conjunto $S \in \mathcal{S}$ não é escolhido pelo algoritmo mais de uma vez, garantindo a validade do primeiro invariante. Como ao final da execução do algoritmo $E' = \emptyset$ e $r''_e = r_e - r'_e = 0$ para todo e em E, consequentemente τ, na linha 13, é uma multi-cobertura de E em $\mathcal{S}$ sujeita a r.

Note que a cada iteração, para pelo menos um e em E', a linha 8 é executada, pois para todo conjunto Z que é escolhido na linha 6, Z possui intersecção não-vazia com E'. Sendo assim, o bloco de linhas 5-12 é executado $O(|E| \cdot \max\{r_e : e \in E\}) = O(|E| \cdot |\mathcal{S}|)$ vezes, pois $\mathcal{S}$ é uma multi-cobertura de E em $\mathcal{S}$ sujeita a r, logo $r_e \leq |\mathcal{S}|$ para todo e em E. Podemos pensar em uma implementação do MinMCCR-Guloso análoga a que sugerimos para o MinCC-Chvátal. Para isso cada S em $\mathcal{S}$ seria representado como um vetor booleano indexado por E, assim como E', porém r seria representado por um vetor de inteiros indexado por E. Guardando o número de elementos na intersecção de cada conjunto de $\mathcal{S}$ com E' durante a execução do algoritmo, tal implementação consome tempo $O(|\mathcal{S}|^2 \cdot |E|^2)$ e é polinomial no tamanho da entrada que é $\Omega(|E| + |\mathcal{S}|)$.

4.2 Razão de aproximação

Teorema 2: *O algoritmo* MinMCCR-Guloso *é uma* H_n*-aproximação para o problema* MinMCCR $(E, \mathcal{S}, c, r)$*, onde* $n = |E|$.

Demonstração: Vamos provar o teorema através do método *dual-fitting*.

Considere o seguinte programa linear inteiro $PI(E, \mathcal{S}, c, r)$ que é equivalente ao MinMCCR para qualquer instância viável: encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{array}{rl} \text{minimize} & cx \\ \text{sob as restrições} & \sum_{e \in S} x_S \geq r_e \text{ para cada } e \in E, \\ & x_S \in \{0, 1\} \text{ para cada } S \in \mathcal{S}. \end{array}$$

Seja $P(E, \mathcal{S}, c, r)$ um programa linear que é uma relaxação linear de $PI(E, \mathcal{S}, c, r)$. Então $P(E, \mathcal{S}, c, r)$ consiste em encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{array}{rl} \text{minimize} & cx \\ \text{sob as restrições} & \sum_{e \in S} x_S \geq r_e \text{ para cada } e \in E, \\ & x_S \leq 1 \text{ para cada } S \in \mathcal{S}, \\ & x_S \geq 0 \text{ para cada } S \in \mathcal{S}. \end{array}$$

Note que a restrição $x_S \leq 1$ para cada $S \in \mathcal{S}$ não é redundante como era no MinCC pois aqui é necessário exigir que cada conjunto $S \in \mathcal{S}$ seja escolhido no máximo uma vez.

Por causa das restrições adicionais do MinMCCR em relação ao MinCC, para escrevermos o dual $D(E, \mathcal{S}, c, r)$ do programa linear acima, é necessário utilizar-se mais um vetor dual z indexado por $\mathcal{S}$. Sendo assim, $D(E, \mathcal{S}, c, r)$ pode ser escrito como: encontrar um vetor y indexado por E e um vetor z indexado por $\mathcal{S}$ que

$$\begin{array}{rl} \text{maximize} & \sum_{e \in E} r_e y_e - \sum_{S \in \mathcal{S}} z_S \\ \text{sob as restrições} & \sum_{e \in S} y_e - z_S \leq c_S \text{ para cada } S \in \mathcal{S}, \\ & y_e \geq 0 \text{ para cada } e \in E, \\ & z_S \geq 0 \text{ para cada } S \in \mathcal{S}. \end{array}$$

A solução primal x é dada pelos conjuntos escolhidos pelo algoritmo. Se τ é a multi-cobertura devolvida pelo algoritmo, $x_S = 1$ se $S \in \tau$ e $x_S = 0$ caso contrário.

Para apresentarmos um candidato a solução dual conveniente, como fizemos no MinCC, podemos pensar que, quando um conjunto $S \in \mathcal{S}$ é escolhido para entrar na cobertura τ em uma dada iteração do algoritmo, o seu custo é dividido igualmente para cada elemento e de $S \cap E'$ para E' no início dessa iteração. Sendo assim, definimos $y_{(e,j)} = c_S/|S \cap E'|$, sendo S o conjunto Z escolhido na linha 6 do algoritmo no início da iteração em que o elemento e é coberto pela j-ésima vez, para cada e em E e $j = 1, 2, \ldots, r_e$, onde r_e é tal

que a última vez que e é coberto na execução do algoritmo é a r_e-ésima vez.

Como o algoritmo sempre escolhe um conjunto Z na linha 6 tal que $c_Z/|Z \cap E'|$ é mínimo, e $c_Z/|Z \cap E'|$ não decresce a cada iteração, temos que para um dado elemento $e \in E$:

$$y_{(e,1)} \leq y_{(e,2)} \leq \cdots \leq y_{(e,r_e)}.$$

Um par (α, β) que é candidato a solução dual, onde α é indexado por E e β é indexado por $\mathcal{S}$, será definido a seguir. Para todo e em E, tome $\alpha_e = y_{(e,r_e)}$. Para todo $S \in \tau$, tome $\beta_S = \sum_{e \in C_S} (\alpha_e - y_{(e,j_e)})$, onde j_e é a vez em que e foi coberto por S, e $C_S = S \cap E'$ no início da iteração em que S é escolhido como o conjunto Z na linha 6 do algoritmo. Para todo $S \in \mathcal{S} \setminus \tau$, tome $\beta_S = 0$.

Vamos provar agora que o valor deste par (α, β) é pelo menos o valor da solução primal, ou ainda, pelo menos o custo da multi-cobertura τ devolvida pelo algoritmo. Como o custo de cada conjunto $S \in \tau$ é dividido pelos elementos que são cobertos por S segue que,

$$c(\tau) = \sum_{e \in E} \sum_{j=1}^{r_e} y_{(e,j)}.$$

O valor da função objetivo de $D(E, \mathcal{S}, c, r)$ para o par (α, β) é

$$\sum_{e \in E} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S = \sum_{e \in E} r_e \alpha_e - \sum_{e \in E} (r_e \alpha_e - \sum_{j=1}^{r_e} y_{(e,j)}) = \sum_{e \in E} \sum_{j=1}^{r_e} y_{(e,j)} = c(\tau)$$

como queríamos demonstrar.

O par (α, β) que definimos em geral não é uma solução de $D(E, \mathcal{S}, c, r)$. Porém, mostraremos que, a partir dele, usando um fator de escala, é possível construir uma solução dual (α', β') como definiremos a seguir:

$$\alpha'_e = \frac{\alpha_e}{H_n} \text{ para cada } e \in E,$$

$$\beta'_S = \frac{\beta_S}{H_n} \text{ para cada } S \in \mathcal{S},$$

onde $n = |E|$, como definido anteriormente.

Vamos provar que o par (α', β') é uma solução de $D(E, \mathcal{S}, c, r)$.

Seja $S \in \mathcal{S}$ e sejam $e_1, e_2, \ldots, e_k$ os elementos de S na ordem em que são totalmente cobertos pelo algoritmo, ou seja, que saem do conjunto E'.

Suponha que S não pertença à multi-cobertura τ devolvida pelo algoritmo. Neste caso, no início da iteração em que o algoritmo escolheu um conjunto que cobriu por completo um elemento $e_i \in S$, existiam pelo menos $k - i + 1$ elementos em $S \cap E'$. Então

$$y_{(e_i, r_{e_i})} \leq \frac{c_S}{k - i + 1}$$

pela escolha da linha 6 do algoritmo. Como $\beta'_S = 0$, temos que

$$(\sum_{i=1}^{k} \alpha'_{e_i}) - \beta'_S = \frac{1}{H_n} \cdot \sum_{i=1}^{k} y_{(e_i, r_{e_i})} \leq \frac{c_S}{H_n} \cdot (\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}) \leq c_S.$$

Suponha agora que S pertença à multi-cobertura τ. Tome k' como o número de elementos de S que já se encontravam completamente cobertos no início da iteração em que S foi escolhido pelo algoritmo. Note que $k - k' = |S \cap E'|$ no início dessa iteração e $0 \leq k' < k$. Então,

$$\begin{aligned}
(\sum_{i=1}^{k} \alpha'_{e_i}) - \beta'_S &= \frac{1}{H_n} \cdot [\sum_{i=1}^{k} y_{(e_i, r_{e_i})} - \sum_{i=k'+1}^{k} (y_{(e_i, r_{e_i})} - y_{(e_i, j_i)})] \\
&= \frac{1}{H_n} \cdot [\sum_{i=1}^{k'} y_{(e_i, r_{e_i})} + \sum_{i=k'+1}^{k} y_{(e_i, j_i)}],
\end{aligned}$$

onde j_i é a vez que e_i é coberto por S, ou seja, $j_i = (r_{e_i} - r'_{e_i} + 1)$ no início da iteração em que S é adicionado a τ. Note que $\sum_{i=k'+1}^{k} y_{(e_i, j_i)} = c_S$ pois o custo do conjunto S é dividido igualmente entre os elementos que são parcialmente cobertos por S na iteração em que este é escolhido. Agora seja $i \in \{1, 2, \ldots, k'\}$. Ao fim da iteração em que e_i sai de E', o conjunto S não foi escolhido e existem pelo menos $k - i + 1$ elementos de E' que pertencem a S. Sendo assim, temos que $y_{(e_i, r_{e_i})} \leq \frac{c_S}{k-i+1}$. Então,

$$\begin{aligned}
(\sum_{i=1}^{k} \alpha'_{e_i}) - \beta'_S &\leq \frac{c_S}{H_n} \cdot (\frac{1}{k} + \cdots + \frac{1}{k - k' + 1} + 1) \\
&\leq c_S.
\end{aligned}$$

A última desigualdade vale pois $(\frac{1}{k} + \cdots + \frac{1}{k-k'+1} + 1) \leq H_k$ já que $k' < k$ e ainda $H_k \leq H_n$ uma vez que $k \leq n$.

Sendo assim, temos que o par (α', β') é uma solução de $D(E, \mathcal{S}, c, r)$.

Como já mostramos, vale que

$$c(\tau) = \sum_{e \in E} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S$$

e então temos

$$c(\tau) = \sum_{e \in E} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S = H_n \cdot [\sum_{e \in E} r_e \alpha'_e - \sum_{S \in \mathcal{S}} \beta'_S] \leq H_n \cdot opt(E, \mathcal{S}, c, r)$$

onde a última desigualdade vale pois o par (α', β') é uma solução de $D(E, \mathcal{S}, c, r)$ e consequentemente$\sum_{e \in E} r_e \alpha'_e - \sum_{S \in \mathcal{S}} \beta'_S$ é uma delimitação inferior para o valor ótimo do problema. Sendo assim, temos que o algoritmo guloso apresentado é uma H_n-aproximação para o MinMCCR.

□

5 Localização de instalações

O *Problema da Localização de Instalações* (facility location), doravante também chamado de MinLI, pode ser definido como visto abaixo:

> **Problema** MinLI (m, n, f, c): *Dados inteiros positivos m e n, um custo de abertura $f_i \in \mathbb{Q}_{\geq}$ para $i \in \{1, \ldots, m\}$ e um custo de conexão $c_{ij} \in \mathbb{Q}_{\geq}$ para todo $i \in \{1, \ldots, m\}$ e todo $j \in \{1, \ldots, n\}$, determinar um conjunto não-vazio $\mathcal{F} \subseteq \{1, \ldots, m\}$ tal que*
>
> $$\sum_{i \in \mathcal{F}} f_i + \sum_{j=1}^{n} \min\{c_{ij} : i \in \mathcal{F}\}$$
>
> *seja mínimo.*

O conjunto $\{1, \ldots, m\}$ representa m instalações, enquanto que o conjunto $\{1, \ldots, n\}$ representa n cidades. Para simplificar, denotaremos um conjunto $\{1, \ldots, \ell\}$ por $[\ell]$.

Iremos estudar o caso métrico desse problema, o qual chamaremos de *Problema Métrico da Localização de Instalações*, ou ainda MinMLI. No MinMLI os custos de conexão respeitam a desigualdade triangular, ou seja, para quaisquer duas instalações i e i' e duas cidades j e j', vale que:

$$c_{ij} \leq c_{ij'} + c_{i'j'} + c_{i'j}.$$

Agora iremos apresentar e analisar um algoritmo de aproximação para esse problema utilizando a técnica do *dual-fitting*. Esse algoritmo foi proposto e analisado por Mahdian *et al.* [7], juntamente com uma versão melhorada cuja análise é ainda mais complexa.

5.1 Algoritmo

O algoritmo que apresentaremos para o MinMLI possui uma abordagem gulosa. Para entendê-lo são necessárias algumas definições.

Uma estrela $S = (i, C)$ é um par ordenado que possui uma instalação $i \in [m]$ e um subconjunto não-vazio C de $[n]$. O custo de uma estrela (i, C) é dado pelo custo de abertura da instalação i mais a soma dos custos de conexão entre a instalação i e cada cidade de C. Formalmente, para uma estrela $S = (i, C)$,

$$\mathit{custo}(S) = f_i + \sum_{j \in C} c_{ij}.$$

Algumas vezes abusaremos da notação, e para uma estrela $S = (i, C)$ e um conjunto de cidades C', escreveremos $S \cap C'$ no lugar de $C \cap C'$.

O algoritmo é iterativo, e consiste em, a cada iteração, escolher e adicionar à solução uma instalação que corresponde a uma estrela de menor "custo médio" dentre todas as estrelas possíveis de serem formadas com as cidades que ainda não foram conectadas a nenhuma instalação. O algoritmo pode ser escrito da seguinte forma:

Algoritmo MinMLI-Guloso (m, n, f, c)

1 . $\mathcal{C} \leftarrow [n]$
2 . $\mathcal{F} \leftarrow \emptyset$
3 . $f' \leftarrow f$
4 . enquanto $\mathcal{C} \neq \emptyset$
5 . tome uma estrela (i, C), $i \in [m]$ e $C \subseteq \mathcal{C}$, que minimiza $\frac{f'_i + \sum_{j \in C} c_{ij}}{|C|}$.
6 . $\mathcal{F} \leftarrow \mathcal{F} \cup \{i\}$
7 . $f'_i \leftarrow 0$
8 . $\mathcal{C} \leftarrow \mathcal{C} \setminus C$
9 . devolva $\mathcal{F}$.

Note que o $\mathcal{F}$ devolvido pelo algoritmo é de fato uma solução (não necessariamente ótima) para o problema, pois é um subconjunto não-vazio de $[m]$. Isso porque $\mathcal{C} = [n] \neq \emptyset$ no início da execução do algoritmo, e portanto pelo menos uma iteração precisa ocorrer e pelo menos uma instalação é escolhida para entrar em $\mathcal{F}$ para que $\mathcal{C}$ se torne vazio e o algoritmo termine.

Pelo menos uma cidade sai de $\mathcal{C}$ a cada iteração, pois como o par (i, C) escolhido na linha 5 é uma estrela temos $C \subseteq \mathcal{C}$ e $C \neq \emptyset$, e portanto alguma cidade sai de $\mathcal{C}$ na linha 8. Portanto o bloco de linhas 4-8 é executado $O(n)$ vezes. Cada execução de uma linha do algoritmo com exceção da linha 5 consome tempo $O(n^2)$, inclusive a linha 6, pois $|\mathcal{F}| \leq n$ sempre que a linha 6 é executada. A análise do consumo de tempo da linha 5 merece um cuidado especial que tomaremos a seguir.

É importante ressaltar que o número de estrelas que são candidatas à escolha na linha 5 é em geral exponencial no tamanho da entrada do problema. Porém é possível encontrar uma tal estrela em tempo polinomial usando a seguinte estratégia.

Para cada instalação $i \in [m]$, ordene as cidades em $\mathcal{C}$ em ordem não-decrescente do custo de conexão entre i e cada uma dessas cidades. Uma

estrela como se deseja conterá uma determinada instalação i e um conjunto de cidades que contém as k primeiras cidades na ordenação de $\mathcal{C}$ imposta por i para algum k entre 1 e $|\mathcal{C}|$. Ou seja, uma estrela (i, C) que minimiza $\frac{f'_i + \sum_{j \in C} c_{ij}}{|C|}$ é tal que não existe um conjunto $C' \subseteq \mathcal{C}$ com $|C'| = |C|$ e $\sum_{j \in C'} c_{ij} < \sum_{j \in C} c_{ij}$. Assim, desconsiderando-se os empates, os elementos de C são os de menor custo c_{ij} com j em $\mathcal{C}$. Sendo assim, é possível executar a linha 5 em tempo $O(m \cdot n \cdot \log n)$. Podemos concluir que o tempo de execução do algoritmo é $O(n \cdot (n^2 + mn \log n))$, ou seja, polinomial no tamanho da instância, que é $\Omega(mn)$. É possível melhorar essa complexidade utilizando-se algumas estruturas de dados específicas, mas isso não será discutido aqui.

O valor $\frac{f'_i + \sum_{j \in C} c_{ij}}{|C|}$ que é calculado na linha 5 é o que chamamos de "custo médio" da estrela (i, C).

5.2 Razão de aproximação

Como foi mostrado por Balinsky [2], o problema MinLI pode ser escrito como um programa linear inteiro. Mostraremos aqui um programa linear inteiro para o MinMLI que é conveniente para a análise que apresentaremos. Considere que $\mathcal{S}$ é o conjunto de todas as estrelas possíveis a partir de $[m]$ e $[n]$. Tal programa linear consiste em encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{aligned} \text{minimize} \quad & \sum_{S \in \mathcal{S}} custo(S) \cdot x_S \\ \text{sob as restrições} \quad & \sum_{S : j \in S} x_S \geq 1 \text{ para todo } j \in [n], \\ & x_S \in \{0, 1\} \text{ para todo } S \in \mathcal{S}. \end{aligned}$$

Podemos escrever uma relaxação linear correspondente $P(m, n, f, c)$ que consiste em encontrar um vetor x indexado por $\mathcal{S}$ que

$$\begin{aligned} \text{minimize} \quad & \sum_{S \in \mathcal{S}} custo(S) \cdot x_S \\ \text{sob as restrições} \quad & \sum_{S : j \in S} x_S \geq 1 \text{ para todo } j \in [n], \\ & x_S \geq 0 \text{ para todo } S \in \mathcal{S}. \end{aligned}$$

O dual $D(m, n, f, c)$ do programa linear $P(m, n, f, c)$ consiste em encontrar

um vetor y indexado por $[n]$ que

$$\begin{aligned} \text{maximize} \quad & \sum_{j\in[n]} y_j \\ \text{sob as restrições} \quad & \sum_{j\in S\cap[n]} y_j \leq custo(S) \text{ para todo } S \in \mathcal{S}, \\ & y_j \geq 0 \text{ para todo } j \in [n]. \end{aligned}$$

Provaremos que o algoritmo mostrado é uma 1,861-aproximação para o MinMLI, mas para isso é mais conveniente escrever esse algoritmo de uma outra forma. Para chegarmos a essa forma, devemos olhar com cuidado para o vetor dual y. Podemos entender que y_j é a contribuição da cidade j para o custo total. Mostraremos a seguir, que a primeira restrição de $D(m, n, f, c)$ pode ser reescrita como: $\sum_{j\in[n]} \max(0, y_j - c_{ij}) \leq f_i$ para todo $i \in [m]$,

Definimos os conjuntos $\mathcal{S}_i$, com $i \in [m]$, da seguinte forma:

$$\mathcal{S}_i = \{S : S \text{ é uma estrela que possui } i \text{ como instalação}\}.$$

Se $\sum_{j\in S\cap[n]} y_j \leq custo(S)$ para todo $S \in \mathcal{S}$, então para todo $i \in [m]$ vale que

$$\sum_{j\in S\cap[n]} y_j \leq f_i + \sum_{j\in S\cap[n]} c_{ij}, \text{ para todo } S \in \mathcal{S}_i.$$

Juntando-se os somatórios

$$\sum_{j\in S\cap[n]} (y_j - c_{ij}) \leq f_i, \text{ para todo } S \in \mathcal{S}_i.$$

Seja $C_i = \{j \in [n] : y_j - c_{ij} \geq 0\}$. Note que (i, C_i) pertence a $\mathcal{S}_i$. Logo

$$\sum_{j\in C_i} (y_j - c_{ij}) \leq f_i.$$

Pela definição de C_i, temos que $\sum_{j\in C_i} (y_j - c_{ij}) = \sum_{j\in[n]} \max(0, y_j - c_{ij})$ e portanto

$$\sum_{j\in[n]} \max(0, y_j - c_{ij}) \leq f_i,$$

concluindo-se assim a prova de uma direção da equivalência das restrições.

Por outro lado, se $\sum_{j\in[n]} \max(0, y_j - c_{ij}) \leq f_i$ para toda instalação i, temos que

$$\sum_{j\in S\cap[n]} (y_j - c_{ij}) \leq \sum_{j\in[n]} \max(0, y_j - c_{ij}) \leq f_i, \text{ para todo } S \in \mathcal{S}_i.$$

Portanto

$$\sum_{j \in S \cap [n]} y_j \leq f_i + \sum_{j \in S \cap [n]} c_{ij} = custo(S), \text{ para todo } S \in \mathcal{S}_i.$$

Como $\mathcal{S} = \cup_{i \in [m]} S_i$, as desigualdades acima implicam que

$$\sum_{j \in S \cap [n]} y_j \leq custo(S), \text{ para todo } S \in \mathcal{S}.$$

À luz desses fatos e introduzindo uma noção de tempo ao algoritmo apresentado, através da variável t que cresce a cada iteração, podemos reescrevê-lo da seguinte maneira:

Algoritmo MinMLI-Guloso2 (m, n, f, c)

1. $\mathcal{C} \leftarrow [n]$
2. $\mathcal{F} \leftarrow \emptyset$
3. para cada $j \in \mathcal{C}$ faça $y_j \leftarrow 0$
4. enquanto $\mathcal{C} \neq \emptyset$
5. para cada $j \in \mathcal{C}$ faça
6. $t_j \leftarrow \min\{c_{ij} : i \in \mathcal{F}\}$
7. para cada $i \in [m] \setminus \mathcal{F}$ faça
8. Sejam $j_1, \ldots, j_{|\mathcal{C}|}$ os elementos de $\mathcal{C}$ tais que $c_{ij_1} \leq \cdots \leq c_{ij_{|\mathcal{C}|}}$
9. $k_i \leftarrow \arg\min\{\frac{f_i + \sum_{\ell=1}^{k} c_{ij_\ell}}{k} : 1 \leq k \leq |\mathcal{C}|\}$
10. $C_i \leftarrow \{j_1, j_2, \ldots, j_{k_i}\}$
11. $t'_i \leftarrow \frac{f_i + \sum_{j \in C_i} c_{ij}}{|C_i|}$
12. $t \leftarrow \min\{\min\{t_j : j \in \mathcal{C}\}, \min\{t'_i : i \in [m] \setminus \mathcal{F}\}\}$
13. para cada $j \in \mathcal{C}$ faça
14. $y_j \leftarrow t$
15. se $t = \min\{t_j : j \in \mathcal{C}\}$
16. então seja $j \in \mathcal{C}$ tal que $t_j = t$
17. $\mathcal{C} \leftarrow \mathcal{C} \setminus \{j\}$
18. senão seja $i \in [m] \setminus \mathcal{F}$ tal que $t'_i = t$
19. $\mathcal{F} \leftarrow \mathcal{F} \cup \{i\}$
20. $\mathcal{C} \leftarrow \mathcal{C} \setminus C_i$
21. devolva $(\mathcal{F}, y)$.

As linhas 3, 13 e 14 e a devolução do y podem ser omitidas, já que y não é utilizado na determinação de $\mathcal{F}$. Porém, para uma determinada instância I, y pode ser utilizado para determinar uma delimitação superior para a razão de aproximação do algoritmo para I especificamente, como mostraremos mais para frente. Note que esta delimitação pode ser menor que a razão de aproximação que iremos obter analisando o algoritmo no caso geral.

O conjunto $\mathcal{F}$ corresponde ao conjunto $\{i \in [m] : f'_i = 0\}$ da primeira versão do algoritmo. Perceba que as linhas 7-11 juntamente com o cálculo de $\min\{t'_i : i \in [m] \setminus \mathcal{F}\}$ na linha 12 correspondem à determinação, na primeira versão do algoritmo, de uma estrela (i, C) que minimiza $\frac{f'_i + \sum_{j \in C} c_{ij}}{|C|}$ para cada instalação i tal que $f'_i \neq 0$. A escolha que fazemos na linha 9 é consequência de uma propriedade que já mostramos na análise do consumo de tempo da primeira versão do algoritmo. Para esta versão, essa propriedade nos diz que existe uma estrela como procuramos composta de uma determinada instalação i e as k primeiras cidades de $\mathcal{C}$ na ordem obtida na linha 8 da iteração correspondente a i no laço da linha 7, para algum $k \leq |\mathcal{C}|$. Na linha 9 podemos determinar, para cada instalação i, um k_i de forma que se i for a instalação de uma dessas estrelas procuradas (decisão que ocorre no cálculo de $\min\{t'_i : i \in [m] \setminus \mathcal{F}\}$ na linha 12), então (i, k_i) é uma dessas estrelas.

Já as linhas 6 e 7 correspondem à determinação de uma estrela $(i, \{j\})$ em que $f'_i = 0$ que minimiza $f'_i + c_{ij} = c_{ij}$. Note que se uma estrela (i, C), com $f'_i = 0$ minimiza $\frac{f'_i + \sum_{j \in C} c_{ij}}{|C|}$ então o mesmo vale para a estrela $(i, \arg\min\{c_{ij} : j \in C\})$. Sendo assim, a escolha de t na linha 12 da segunda versão do algoritmo corresponde à determinação de uma estrela de menor "custo médio" na linha 5 da primeira versão.

Dizemos que as cidades em C_i *se conectaram* à instalação i incluída em $\mathcal{F}$ na linha 19 do algoritmo. Para cada cidade j removida de C na linha 17, seja i em $\mathcal{F}$ tal que $t_j = c_{ij}$. Dizemos que j *se conectou* a tal i.

Utilizaremos a noção de tempo introduzida ao algoritmo pela variável t durante a análise. Sendo assim, é importante mostrar que essa noção está bem definida.

Lema 3: *O valor da variável t calculado em uma iteração que não a primeira é sempre maior ou igual ao valor de t na iteração anterior.*

Demonstração: Seja v uma variável qualquer presente no algoritmo MinMLI-Guloso2. Definimos v^r como o valor de v após a execução da linha 12 na r-ésima iteração do laço da linha 4.

Mostraremos que $t^{r-1} \leq t^r$ para todo $r > 1$.

Primeiro, vamos analisar o caso em que $t^r = t_j$ para algum $j \in \mathcal{C}^r$. Neste caso, basta mostrar que

$$\min\{c_{ij} : i \in \mathcal{F}^r \text{ e } j \in \mathcal{C}^r\} \geq t^{r-1}.$$

Se $\mathcal{F}^r = \mathcal{F}^{r-1}$ temos que $\mathcal{C}^r \subseteq \mathcal{C}^{r-1}$ e

$$t^r = \min\{c_{ij} : i \in \mathcal{F}^r \text{ e } j \in \mathcal{C}^r\} \geq \min\{c_{ij} : i \in \mathcal{F}^{r-1} \text{ e } j \in \mathcal{C}^{r-1}\} \geq t^{r-1},$$

pois temos menos escolhas no mínimo da r-ésima iteração do que na anterior.

Se $\mathcal{F}^r = \mathcal{F}^{r-1} \cup \{i\}$, então $\mathcal{C}^r \subseteq \mathcal{C}^{r-1}$ e basta mostrar que $c_{i\ell} \geq t^{r-1}$ para todo ℓ em $\mathcal{C}^r$. Suponha que $c_{i\ell} < t^{r-1}$ para algum ℓ em $\mathcal{C}^r$. Note que ℓ não pertence ao C_i^{r-1} pois caso contrário teria sido removido de $\mathcal{C}^{r-1}$ na linha 20 e não poderia estar em $\mathcal{C}^r$. Sendo assim, pela ordenação da linha 8, podemos afirmar que $c_{ij_{k+1}^{r-1}} < t^{r-1}$, para $k = k_i^{r-1}$. Mas então

$$\begin{aligned} \frac{f_i + \sum_{\ell=1}^{k+1} c_{ij_\ell^{r-1}}}{k+1} &= \frac{f_i + \sum_{\ell=1}^{k} c_{ij_\ell^{r-1}} + c_{ij_{k+1}^{r-1}}}{k+1} \\ &= \frac{t^{r-1} \cdot k + c_{ij_{k+1}^{r-1}}}{k+1} < \frac{t^{r-1} \cdot k + t^{r-1}}{k+1} = t^{r-1}, \end{aligned}$$

contrariando a escolha de k_i^{r-1}.

Segundo, vamos analisar o caso em que $t^r \neq t_j$ para todo $j \in \mathcal{C}^r$. Neste caso, $t^r = t'_i$ para algum $i \in [m] \setminus \mathcal{F}^r$. Seja $k = k_i^r$. Pela ordenação da linha 8 e como $\mathcal{C}^r \subseteq \mathcal{C}^{r-1}$ e $i \in [m] \setminus \mathcal{F}^{r-1}$, temos

$$t^{r-1} \leq (t')_i^{r-1} \leq \frac{f_i + \sum_{\ell=1}^{k} c_{ij_\ell^{r-1}}}{k} \leq \frac{f_i + \sum_{\ell=1}^{k} c_{ij_\ell^{r}}}{k} = t^r.$$

Assim sendo, $t^r \geq t^{r-1}$ para todo $r > 1$.

□

Lema 4: *Ao final do algoritmo* MinMLI-Guloso2*, vale que*

$$\sum_{i \in \mathcal{F}} f_i + \sum_{j \in [n]} \min\{c_{ij} : i \in \mathcal{F}\} \leq \sum_{j \in [n]} y_j.$$

Demonstração: Seja D_1 o conjunto das cidades que saíram de $\mathcal{C}$ na linha 17 durante a execução do algoritmo. Seja ainda D_2 o conjunto das cidades que saíram de $\mathcal{C}$ na linha 20. Note que $[n] = D_1 \cup D_2$ e $D_1 \cap D_2 = \emptyset$. Então

$$\sum_{j \in [n]} y_j = \sum_{j \in D_1} y_j + \sum_{j \in D_2} y_j.$$

Para todo j em $D_1 \cup D_2$, seja i_j a instalação a que a cidade j se conectou. Além disso, para todo i em $\mathcal{F}$ considere o conjunto C_i calculado na linha 10 da iteração em que i entrou em $\mathcal{F}$. Então

$$\sum_{j\in[n]} y_j = \sum_{j\in D_1} c_{i_j j} + \sum_{j\in D_2} \frac{f_{i_j} + \sum_{\ell\in C_{i_j}} c_{i_j \ell}}{|C_{i_j}|}.$$

Mas como toda instalação i entra em $\mathcal{F}$ na linha 19 se e somente se $|C_i|$ cidades de D_2 saem de $\mathcal{C}$ na mesma iteração na linha 20, temos que $\sum_{j\in D_2} \frac{f_{i_j}}{|C_{i_j}|} = \sum_{i\in\mathcal{F}} f_i$. Então

$$\begin{aligned}
\sum_{j\in[n]} y_j &= \sum_{j\in D_1} c_{i_j j} + \sum_{i\in\mathcal{F}} f_i + \sum_{j\in D_2} \frac{\sum_{\ell\in C_{i_j}} c_{i_j \ell}}{|C_{i_j}|}, \\
&= \sum_{i\in\mathcal{F}} f_i + \sum_{j\in D_1} c_{i_j j} + \sum_{j\in D_2} c_{i_j j} \\
&\geq \sum_{i\in\mathcal{F}} f_i + \sum_{j\in[n]} \min\{c_{ij} : i \in \mathcal{F}\}.
\end{aligned}$$

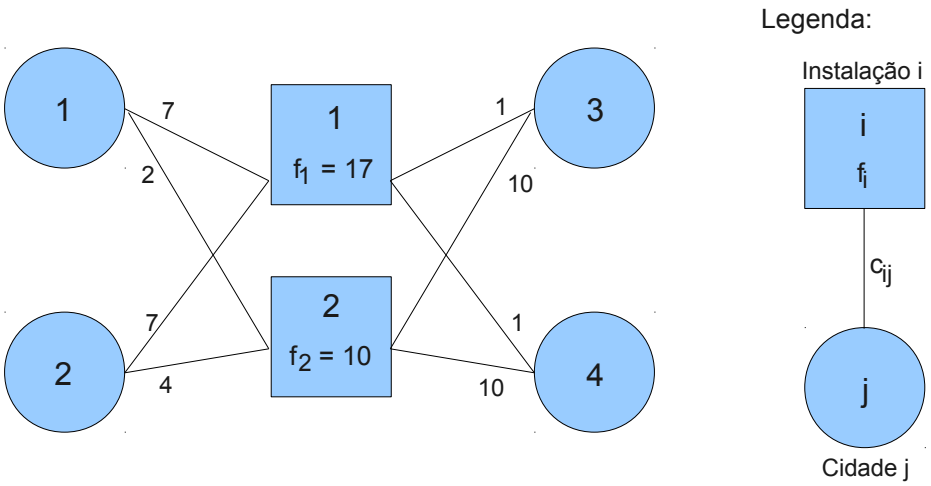
□

Porém, y não é necessariamente uma solução do dual $D(m, n, f, c)$, pois a restrição $\sum_{j\in[n]} \max(0, y_j - c_{ij}) \leq f_i$ pode não ser respeitada para alguma instalação i. O algoritmo garante que para todo $i \in \mathcal{F}$,

$$\sum_{j\in C_i} (y_j - c_{ij}) = f_i$$

onde C_i é o conjunto calculado na linha 10 da iteração em que i entrou em $\mathcal{F}$. Mas, possivelmente, para algum $j \in [n] \setminus C_i$, temos que $y_j > c_{ij}$ ao final da execução do algoritmo.

Vejamos um exemplo. Considere uma instância em que $m = 2$, $n = 4$, $f_1 = 17$, $f_2 = 10$, $c_{11} = 7$, $c_{12} = 7$, $c_{13} = 1$, $c_{14} = 1$, $c_{21} = 2$, $c_{22} = 4$, $c_{23} = 10$ e $c_{24} = 10$, assim como na figura a seguir.



Vamos simular o algoritmo para essa instância.

Antes da 1^a iteração			
y_1	0	$\mathcal{C}$	$\{1,2,3,4\}$
y_2	0	$\mathcal{F}$	$\emptyset$
y_3	0	C_1	$\emptyset$
y_4	0	C_2	$\emptyset$

Após a 1^a iteração			
y_1	8	$\mathcal{C}$	$\{3,4\}$
y_2	8	$\mathcal{F}$	$\{2\}$
y_3	8	C_1	$\emptyset$
y_4	8	C_2	$\{1,2\}$

A instalação 2 foi aberta apenas com as contribuições das cidades 1 e 2, pois o custo de conexão entre as cidades 3 e 4 e a instalação 2 é 10, maior que o tempo corrente igual a 8.

Veja que a contribuição das 4 cidades para a instalação 1 nesse momento é $(8-7)+(8-7)+(8-1)+(8-1) = 16 < f_1$.

Após a 2^a iteração			
y_1	8	$\mathcal{C}$	$\emptyset$
y_2	8	$\mathcal{F}$	$\{1,2\}$
y_3	9,5	C_1	$\{3,4\}$
y_4	9,5	C_2	$\{1,2\}$

Após a segunda iteração, o algoritmo termina pois o $\mathcal{C}$ se encontra vazio nesse momento. Perceba que ao fim da execução do algoritmo temos $\sum_{j\in[n]} \max(0, y_j - c_{1j}) = 19 > 17 = f_1$, violando a restrição como queríamos exemplificar.

Lembrando que estamos utilizando o método *dual-fitting*, devemos achar γ tal que $\frac{y}{\gamma}$ seja solução de $D(m,n,f,c)$. Assim, o valor $\sum_{j\in[n]} \frac{y_j}{\gamma}$ será uma delimitação inferior para o valor ótimo do problema original e consequentemente o algoritmo mostrado será uma γ-aproximação para tal problema. Desejamos determinar o menor valor que pudermos para γ (mais próximo de 1), e é o que seguiremos buscando nessa análise. Para tanto, precisamos mostrar algumas propriedades de y.

Lema 5: *Se a cidade j se conectou à instalação i, então $y_j \geq c_{ij}$.*

Demonstração: Para j removido de $\mathcal{C}$ na linha 17, temos que $y_j = t_j = c_{ij}$, para t_j calculado nessa iteração em que j sai de $\mathcal{C}$. Para j removido na linha 20 é preciso observar que $y_j = t'_i = \frac{f_i + \sum_{j\in C_i} c_{ij}}{|C_i|} = \frac{f_i + \sum_{\ell=1}^{k_i} c_{ij_\ell}}{k_i}$ e, pela linha 11 e pela escolha de k_i na linha 10, temos que $t'_i \cdot k_i = f_i + \sum_{\ell=1}^{k_i} c_{ij_\ell} \geq k_i \cdot c_{ij_{k_i}}$. Logo $y_j = t'_i \geq c_{ij_{k_i}} \geq c_{ij}$ pela ordenação da linha 8. □

Lema 6: *Para quaisquer duas cidades j e j' e qualquer instalação i vale que $y_j \leq y_{j'} + c_{ij'} + c_{ij}$.*

Demonstração: Se $y_j \leq y_{j'}$, a desigualdade se torna óbvia pois todo custo de conexão é não-negativo. Caso contrário, ou seja, quando $y_j > y_{j'}$, seja i' a instalação à que j' se conectou. Claro que i' estava em $\mathcal{F}$ no tempo $y_j > y_{j'}$. Sendo assim temos que $y_j \leq c_{i'j}$ pois caso contrário a cidade j teria se conectado à instalação i' no tempo $t = c_{i'j} < y_j$. Pela desigualdade triangular, vale que $c_{i'j} \leq c_{i'j'} + c_{ij'} + c_{ij} \leq y_{j'} + c_{ij'} + c_{ij}$, pois $y_{j'} \geq c_{i'j'}$ pelo Lema 5. □

Para que $\frac{y}{\gamma}$ seja solução de $D(m,n,f,c)$, o número γ deve ser tal que para

todo $i \in [m]$

$$\sum_{j \in [n]} \max(\frac{y_j}{\gamma} - c_{ij}, 0) \leq f_i.$$

Note que, para uma determinada instalação i, uma cidade j só contribui para o valor do somatório acima se $y_j \geq \gamma \cdot c_{ij}$. Vamos supor, sem perda de generalidade, que esse é o caso para todo $j \in \{1, 2, \ldots, k_i\}$, e apenas para estes, para algum k_i. Além disso, vamos supor, sem perda de generalidade, que $y_\ell \leq y_{\ell+1}$ para todo ℓ tal que $1 \leq \ell \leq k_i - 1$.

Lema 7: *Para cada instalação i e para cada cidade j, vale que $\sum_{\ell=j}^{k_i} \max(y_j - c_{i\ell}, 0) \leq f_i$.*

Demonstração: Se $j > k_i$, então $\sum_{\ell=j}^{k_i} \max(y_j - c_{i\ell}, 0) = 0 \leq f_i$. Admita que $j \leq k_i$ e suponha que $\sum_{\ell=j}^{k_i} \max(y_j - c_{j\ell}, 0) > f_i$. Considere o tempo y_j. Como $y_j \leq y_\ell$ quando $j \leq \ell \leq k_i$, as cidades $\{j, \ldots, k_i\}$ estão desconectadas antes do tempo y_j, ou seja, estas cidades estão em $\mathcal{C}$ em qualquer tempo anterior ao tempo y_j. Mas então, o custo de abertura da instalação i estava todo pago em algum momento antes do tempo y_j, pois este teria sido pago por estas cidades. Logo a instalação i abriu antes do tempo y_j. Como $\sum_{\ell=j}^{k_i} \max(y_j - c_{i\ell}, 0) > f_i \geq 0$, para algum ℓ em $\{j, \ldots, k_i\}$ temos que $y_j > c_{i\ell}$. Mas isso implica que ℓ saiu de $\mathcal{C}$ antes do tempo y_j, uma contradição já que $y_\ell \geq y_j$. Portanto $\sum_{\ell=j}^{k_i} \max(y_j - c_{i\ell}, 0) \leq f_i$. □

Desejamos encontrar o menor γ tal que $\sum_{j=1}^{k_i} (\frac{y_j}{\gamma} - c_{ij}) \leq f_i$, para todo $i \in [m]$. Essa restrição equivale a

$$\frac{1}{\gamma} \cdot \sum_{j=1}^{k_i} y_j - \sum_{j=1}^{k_i} c_{ij} \leq f_i \text{ , para todo } i \in [m],$$

ou ainda a

$$\gamma \geq \frac{\sum_{j=1}^{k_i} y_j}{f_i + \sum_{j=1}^{k_i} c_{ij}} \text{ , para todo } i \in [m].$$

Tal γ deve satisfazer essa desigualdade para toda instalação de qualquer instância do problema. Ou seja, quaisquer que sejam os valores de m, n, f_i, para $i \in [m]$, c_{ij} para $i \in [m]$, $j \in [n]$ e os y_j, para $j \in [n]$ e k_i para $i \in [m]$ vindos do algoritmo, γ deve satisfazer a desigualdade acima.

Vamos apresentar um programa racional $PR(k)$, que para cada valor de k tem como soluções, entre outras coisas, instâncias do MinMLI e os respectivos

valores de y_j do algoritmo: encontrar um vetor y indexado por $[k]$, um vetor d indexado por $[k]$ e um número não-negativo f que

$$\begin{array}{rcl}
\text{maximize} & \frac{\sum_{j=1}^{k} y_j}{f+\sum_{j=1}^{k} d_j} & \\
\text{sob as restrições} & y_j \leq y_{j+1} & \text{para todo } j \in [k-1] \\
& y_j \leq y_\ell + d_j + d_\ell & \text{para todo } j \in [k] \text{ e todo } \ell \in [k] \\
& \sum_{\ell=j}^{k} \max(y_j - d_\ell, 0) \leq f & \text{para todo } j \in [k] \\
& y_j, d_j \geq 0 & \text{para todo } j \in [k] \\
& f \geq 0. &
\end{array}$$

O f representa um custo de abertura genérico de uma instalação, enquanto que d_j representa o custo de conexão entre tal instalação e a cidade j. O Lema 6 garante que os $y'_j s$ produzidos pelo algoritmo para uma instância e custos de conexão satisfazem a segunda restrição. O Lema 7 garante que esses mesmos $y'_j s$ satisfazem a terceira restrição. A primeira restrição reflete nossa hipótese que é válida sem perda de generalidade.

Vamos encontrar um programa linear que seja equivalente a $PR(k)$. Podemos substituir cada restrição do tipo $\sum_{l=j}^{k} \max(y_j - d_l, 0) \leq f$, evitando o cálculo do máximo na restrição, por outras duas restrições lineares adicionando-se um vetor x indexado por $[k]\times[k]$ ao programa. Mostraremos que $PR(k)$ possui o mesmo valor ótimo que o programa linear $PLRRA(k)$ abaixo, que chamaremos de *Programa Linear Revelador da Razão de Aproximação* (factor-revealing LP). O $PLRRA(k)$ consiste em encontrar um vetor y indexado por $[k]$, um vetor d indexado por $[k]$, um vetor x indexado por $[k]\times[k]$ e um número não-negativo f que

$$\begin{array}{rcl}
\text{maximize} & \sum_{j=1}^{k} y_j & \\
\text{sob as restrições} & f + \sum_{j=1}^{k} d_j = 1 & \\
& y_j \leq y_{j+1} & \text{para } j \in [k-1] \\
& y_j \leq y_\ell + d_j + d_\ell & \text{para todo } j \in [k] \text{ e todo } \ell \in [k] \\
& x_{j\ell} \geq y_j - d_\ell & \text{para todo } j \in [k] \\
& \sum_{l=j}^{k} x_{jl} \leq f & \text{para todo } j \in [k] \\
& y_j, d_j \geq 0 & \text{para todo } j \in [k] \\
& x_{j\ell} \geq 0 & \text{para todo } j \in [k] \text{ e todo } \ell \in [k] \\
& f \geq 0. &
\end{array}$$

Para nos livrarmos da divisão na função objetivo que existe em $PR(k)$, vamos mostrar o seguinte: se (y, d, f) é solução de $PR(k)$ então $c \cdot (y, d, f)$ é

solução de $PR(k)$ de mesmo valor para todo número c positivo. Uma vez que isso for mostrado, podemos buscar soluções que satisfazem $f + \sum_{j=1}^{k} d_j = 1$. Isso restringe as soluções àquelas em que a função objetivo é linear, e é o que fazemos em $PLRRA(k)$.

De fato, se (y, d, f) for uma solução de $PR(k)$ e seja c um número positivo. Defina

$$\begin{aligned} y'_i &= c \cdot y_i \\ d'_i &= c \cdot d_i \\ f' &= c \cdot f. \end{aligned}$$

Mostraremos que (y', d', f') também é solução de $PR(k)$, e de mesmo valor que (y, d, f). Como $y_j \leq y_{j+1}$ e $c > 0$, $y'_j = c \cdot y_j \leq c \cdot y_{j+1} = y'_{j+1}$ para todo j. Como $y_j \leq y_\ell + d_j + d_\ell$, temos que $y'_j = c \cdot y_j \leq c \cdot y_\ell + c \cdot d_j + c \cdot d_\ell = y'_\ell + d'_j + d'_\ell$, para todo j e ℓ. Como $\sum_{\ell=j}^{k} \max(y_j - d_\ell, 0) \leq f$, vale que

$$\begin{aligned} \sum_{\ell=j}^{k} \max(y'_j - d'_\ell, 0) &= \sum_{\ell=j}^{k} \max(cy_j - cd_\ell, 0) \\ &= \sum_{\ell=j}^{k} c \cdot \max(y_j - d_\ell, 0) \\ &= c \cdot \sum_{\ell=j}^{k} \max(y_j - d_\ell, 0) \\ &\leq c \cdot f = f'. \end{aligned}$$

Portanto (y', d', f') também é solução de $PR(k)$. Agora veja que

$$\begin{aligned} V(PR(k), (y', d', f')) &= \frac{\sum_{j=1}^{k} y'_j}{f' + \sum_{j=1}^{k} d'_j} = \frac{\sum_{j=1}^{k} cy_j}{cf + \sum_{j=1}^{k} cd_j} = \frac{c \cdot \sum_{j=1}^{k} y_j}{c \cdot (f + \sum_{j=1}^{k} d_j)} \\ &= \frac{\sum_{j=1}^{k} y_j}{f + \sum_{j=1}^{k} d_j} = V(PR(k), (y, d, f)). \end{aligned}$$

Perceba que, em especial, se (y, d, f) é uma solução ótima de $PR(k)$, então $c \cdot (y, d, f)$ também o é para todo c positivo. Ou seja, o valor ótimo de $PR(k)$ é igual ao valor ótimo de $PLRRA(k)$.

Seja z_k o valor ótimo de $PR(k)$ e de $PLRRA(k)$ para um dado $k \geq 1$.

Lema 8: *Seja* $\gamma = \sup_{k\geq 1} z_k$, *então* $\sum_{j\in[n]} \max(\frac{y_j}{\gamma} - c_{ij}, 0) \leq f_i$ *para todo* $i \in [m]$.

Demonstração: Tome $i \in [m]$. Suponha, sem perda de generalidade, que $y_j \geq \gamma c_{ij}$ apenas para $j = 1, 2, \ldots, k_i$ para algum k_i em $[n]$. Ademais, suponha que $y_1 \leq y_2 \leq \ldots \leq y_{k_i}$. Agora seja $d_\ell = c_{i\ell}$ para todo ℓ em $[k_i]$ e $f = f_i$. Dessa forma (y, d, f) é solução do programa linear $PR(k_i)$, como mostraremos a seguir.

Pela definição de f, y e d, temos que $y_j \geq 0$ e $d_j \geq 0$ para todo j em $[k_i]$, assim como $f \geq 0$. Pela suposição que fizemos, temos que $y_\ell \leq y_{\ell+1}$ para todo ℓ em $[k_i - 1]$. Pelo Lema 6, vale que $y_j \leq y_{j'} + d_j + d_{j'}$ para todo j e todo j' em $[k_i]$. Pelo Lema 7, vale que $\sum_{\ell=j}^{k_i} \max(y_j - d_\ell, 0) \leq f$ para todo j em $[k_i]$. Portanto, (y, d, f) é solução do programa $PR(k_i)$, e consequentemente $\frac{\sum_{j=1}^{k_i} y_j}{f_i + \sum_{j=1}^{k_i} c_{ij}} \leq z_{k_i}$. Então, manipulando essa desigualdade chegamos a

$$\sum_{j=1}^{k_i} (\frac{y_j}{z_{k_i}} - c_{ij}) \leq f_i.$$

Como $\gamma \geq z_{k_i}$ e $y_j \geq \gamma \cdot c_{ij}$ para todo $j \in [k_i]$ e apenas estes,

$$\sum_{j\in[n]} \max(\frac{y_j}{\gamma} - c_{ij}, 0) = \sum_{j=1}^{k_i} (\frac{y_j}{\gamma} - c_{ij}) \leq \sum_{j=1}^{k_i} (\frac{y_j}{z_{k_i}} - c_{ij}) \leq f_i.$$

□

Sendo assim, $\gamma = \sup_{k\geq 1} z_k$ é uma razão de aproximação do algoritmo. Mostraremos a seguir que este resultado não pode ser melhorado, ou seja, que esta análise é justa.

Considere uma solução ótima (y, d, f) do programa $PLRRA(k)$ para um dado k. Lembre-se que y é um vetor indexado por $[k]$, d é um vetor indexado por $[k]$, e f é um número não-negativo. Podemos construir uma instância do MinMLI tal que $m = k+1$ e $n = k$, a partir dessa solução, da seguinte maneira.

Tome os custos de abertura f_i de cada instalação i em $[m]$ como

$$f_i = \begin{cases} 0 \text{ , se } 1 \leq i \leq k \\ f \text{ , se } i = k+1, \end{cases}$$

e tome os custos de conexão c_{ij} entre cada instalação i em $[m]$ e cada cidade

j em $[n]$ como

$$c_{ij} = \begin{cases} y_j \text{ , se } i = j \\ d_j \text{ , se } i = k+1 \\ d_i + d_j + y_i \text{ , caso contrário.} \end{cases}$$

Os custos de conexão definidos dessa forma respeitam a desigualdade triangular, como mostraremos. Seja i em $[m]$ e j em $[n]$. Temos três possibilidades:

- Se $i = j$, então $c_{ij} = y_j$, e $c_{i\ell} = d_i + d_\ell + y_i \geq y_i$ para todo $\ell \in [k]$ tal que $\ell \neq j$. Portanto, para qualquer instalação $i' \neq i$ e cidade $j' \neq j$, $c_{ij'} + c_{i'j'} + c_{i'j} \geq y_i + c_{i'j'} + c_{i'j} \geq y_i = y_j = c_{ij}$.

- Se $i = k+1$, então $c_{ij} = d_j$, e para qualquer instalação $i' \neq i$ e cidade $j' \neq j$ temos que $c_{ij} \leq c_{ij'} + c_{i'j'} + c_{i'j}$, pois se $i' = j$ então $c_{i'j'} = d_{i'} + d_{j'} + y_{i'} \geq d_{i'} = d_j = c_{ij}$, e se $i' \neq j$ então $c_{i'j} = d_{i'} + d_j + y_{i'} \geq d_j = c_{ij}$.

- Para as outras possibilidades de i e j, vale que $c_{ij} = d_i + d_j + y_i$, e para qualquer instalação $i' \neq i$ e cidade $j' \neq j$ temos que $c_{ij} \leq c_{ij'} + c_{i'j'} + c_{i'j}$. Se $i' = k+1$, então se $i \neq j'$ temos $c_{ij'} + c_{i'j} = d_i + d_{j'} + y_i + d_j \geq d_i + d_j + y_i = c_{ij}$, e se $i = j'$ temos $c_{ij'} + c_{i'j'} + c_{i'j} = y_{j'} + d_{j'} + d_j = y_i + d_i + d_j = c_{ij}$. Se $i' \neq k+1$, então se $i' = j'$ temos que $i \neq j'$ e $i' \neq j$ e então $c_{ij'} + c_{i'j} = d_i + d_{j'} + y_i + d_{i'} + d_j + y_{i'} \geq d_i + d_j + y_i = c_{ij}$, e se $i' \neq j'$ temos ainda quatro casos:

  - Se $i = j'$ e $i' = j$, então $c_{ij'} + c_{i'j'} = y_{j'} + d_{i'} + d_{j'} + y_{i'} \geq y_{j'} + d_{i'} + d_{j'} = y_i + d_j + d_i = c_{ij}$.
  - Se $i = j'$ e $i' \neq j$, então $c_{ij'} + c_{i'j'} + c_{i'j} = y_{j'} + d_{i'} + d_{j'} + y_{i'} + d_{i'} + d_j + y_{i'} \geq y_{j'} + d_{j'} + d_j = y_i + d_i + d_j = c_{ij}$.
  - Se $i \neq j'$ e $i' = j$, então $c_{ij'} + c_{i'j'} = d_i + d_{j'} + y_i + d_{i'} + d_{j'} + y_{i'} \geq d_i + y_i + d_{i'} = d_i + y_i + d_j = c_{ij}$.
  - Se $i \neq j'$ e $i' \neq j$, então $c_{ij'} + c_{i'j} = d_i + d_{j'} + y_i + d_{i'} + d_j + y_{i'} \geq d_i + y_i + d_j = c_{ij}$.

Para esta instância, o algoritmo conecta a cidade 1 à instalação 1, a cidade 2 à instalação 2, e assim sucessivamente até conectar a cidade k à instalação k. Isso porque a restrição $\sum_{\ell=j}^{k} \max(y_j - d_\ell, 0) \leq f$ para todo $j \in [k]$ garante que na j-ésima iteração a cidade j se conecte com a instalação j antes que as contribuições das cidades não conectadas no início dessa iteração, ou $[k] \setminus [j-1]$, para a instalação $k+1$ sejam suficientes para pagar seu custo de abertura. O custo dessa solução devolvida pelo algoritmo é

$\sum_{\ell=1}^{k} c_{\ell\ell} + \sum_{i=1}^{k} f_i = \sum_{j=1}^{k} y_j = z_k$, pois (y, d, f) é solução ótima de $PLRRA(k)$. Porém a solução ótima do MinMLI para essa instância que construímos consiste em conectar todas as cidades à instalação $k+1$, e o custo dessa solução ótima é $\sum_{j=1}^{k} c_{(k+1)j} + f_{k+1} = f + \sum_{j=1}^{k} d_j = 1$, pela primeira restrição de $PLRRA(k)$.

Sendo assim o algoritmo devolve para esta instância uma solução cujo valor é z_k vezes o valor ótimo do problema para essa instância. Como há uma tal instância para cada valor de k, temos uma família de instâncias na qual o algoritmo atinge uma razão que se aproxima arbitrariamente de $\sup_{k\geq 1} z_k$, o que mostra que a análise apresentada é justa.

Para determinar uma razão de aproximação constante para o algoritmo é necessário encontrar uma delimitação superior para $\gamma = \sup_{k\geq 1} z_k$. Apenas resolver o programa linear $PLRRA(k)$ para algum valor específico de k não é o bastante pois isso apensa reulta em uma delimitação inferior para γ. Para mostrar um limitante superior para γ, Mahdian et al. [7] usaram o Teorema Forte da Dualidade juntamente com um estudo experimental cuidadoso. Eles conseguiram soluções para o dual de cada $PLRRA(k)$ com um valor bem próximo de γ. Especificamente, eles provaram o seguinte.

Lema 9: *Para todo* $k \geq 1$, $z_k \leq 1{,}861$.

Não iremos demonstrar a veracidade do Lema 9. Essa demonstração se encontra no artigo de Mahdian *et al.* [7]. O que iremos fazer é explicar as ideias principais da estratégia utilizada pelos autores que permitiu a Mahdian et al. encontrarem uma solução quase ótima do programa linear dual de cada $PLRRA(k)$, de valor não maior que 1,861.

A análise empírica de soluções ótimas do programa linear dual de $PLRRA(k)$, que chamaremos de $DPLRRA(k)$, para pequenos valores de k permitiu a Mahdian *et al.* deduzir uma solução de $DPLRRA(k)$ para todo $k \geq 1$ com valor estritamente menor que 1,861. Pelo Teorema Forte da Dualidade, os autores puderam concluir então que o valor de z_k é estritamente menor que 1,861 para todo $k \geq 1$, e consequentemente $\gamma = \sup_{k\geq 1} z_k \leq 1{,}861$, mostrando que o algoritmo é uma 1,861-aproximação para o MinMLI.

Os autores também conseguiram uma delimitação inferior razoável para γ. Para isso, utilizaram o programa CPLEX para obter uma solução ótima de $PLRRA(k)$ para valores de k não tão grandes a ponto de ser impraticável calcular essa solução por causa do tempo necessário, e obtiveram que $z_{300} \approx 1{,}81$. Assim, puderam inferir que $1{,}81 \leq \gamma = \sup_{k\geq 1} z_k \leq 1{,}861$.

Já mostramos que o algoritmo MinMLI-Guloso2 garante uma razão

de aproximação estritamente menor que 1,861 para qualquer instância do MinMLI, ou seja, que

$$val(\mathcal{F}) \leq \sum_{j \in [n]} y_j \leq 1{,}861 \cdot opt(I)$$

para qualquer instância I. Agora, para uma instância $I = (m, n, f, c)$ específica, a partir do par $(\mathcal{F}, y)$ devolvido pelo algoritmo MinMLI-Guloso2 para I, podemos calcular

$$\alpha(I) = \frac{val(\mathcal{F})}{\sum_{j \in [n]} y_j} \cdot 1{,}861$$

e temos que

$$val(\mathcal{F}) = \frac{\alpha(I)}{1{,}861} \cdot \sum_{j \in [n]} y_j \leq \alpha(I) \cdot \frac{1{,}861 \cdot opt(I)}{1{,}861} = \alpha(I) \cdot opt(I).$$

Ou seja, o valor de y permite deduzir uma garantia de aproximação $\alpha(I)$ possivelmente melhor para a instância em questão.

6 Conclusão e resultados

O método dual-fitting é muito útil para se determinar a razão de aproximação de alguns algoritmos de aproximação para problemas de otimização combinatória. Nem sempre a análise é simples. Vimos que principalmente a análise do algoritmo para o MinMLI foi bem sofisticada, mas é importante ressaltar que este é um método bem definido. Se identificarmos as propriedades necessárias no problema e no algoritmo, o dual-fitting é uma alternativa.

Algumas vezes, como no caso do algoritmo guloso apresentado para o MinMLI, a análise através do método dual-fitting fornece a razão de aproximação mais justa da literatura, o que mostra o seu grande potencial.

Um outro exemplo de aplicação do método dual-fitting é apresentado por Athanassopoulos *et al.* [1], onde este método é utilizado para analisar dois algoritmos que são modificações de algortimos gulosos para o Problema da Cobertura Mínima por k-Conjuntos (*k-Set Cover Problem*), que é uma variação do MinCC que mostramos neste trabalho. Essa aplicação tem uma dificuldade intermediária entre os dois primeiros exemplos aqui apresentados e o último.

Um outro trabalho interessante é o de Freund e Rawitz [5] que descreve um método equivalente ao dual-fitting, porém com uma abordagem mais combinatória, menos baseada em programação linear. Esse artigo aborda análises de

algoritmos para o MinCC, o MinMLI e um outro problema que não foi visto neste trabalho, que é chamado, no original, de *Prize Collecting and Partial Disk Cover*.

7 Parte subjetiva

Já estudo o tema escolhido para essa monografia desde abril de 2009, através de uma Iniciação Cientifíca sob orientação da Profa. Dra. Cristina Gomes Fernandes. Escolhemos o tema do método dual-fitting, pois eu tinha interesse na área de Algoritmos de Aproximação, apesar do meu contato com a área ser recente quando fizemos essa escolha, e esse parecia ser um assunto bem atual e desafiador. Esse tema ainda envolve diretamente programação linear e indiretamente muitos outros tópicos que gosto em Ciência da Computação.

Inicialmente comecei a tentar entender de forma abstrata o método dual-fitting, porém foi mais fácil tentando entendê-lo junto da análise do Problema da Cobertura Mínima por conjuntos no Capítulo 13 do livro de V. Vazirani [8], que seria o exemplo mais simples que encontramos. Depois disso passamos a procurar outras análises mais complicadas.

7.1 Desafios e frustrações

O assunto estudado se mostrou bem difícil, isso fez com que o ritmo de produção da monografia parecesse baixo em alguns momentos, a despeito da quantidade de trabalho empregada, e isso é um tanto frustrante. Grande parte dessa dificuldade se deve a escolha da análise do algoritmo para o caso métrico do Problema da Localização de Instalações, o MinMLI. Mas por outro lado foi muito proveitoso o desafio proporcionado por esse problema, aprendi bastante, e aprendi a ter mais rigor em demonstrações, e isso eu devo em grande parte a minha orientadora. Um efeito colateral do alto nível de dificuldade dessa última análise foi não ter tido tempo de estudar mais análises ou tentar fazer uma análise inédita na literatura que utilize o método dual-fitting, como havia cogitado. Em suma, apesar da dificuldade do tema escolhido para a monografia, e em especial de certa análise escolhida também, acredito que os benefícios provenientes desse desafio foram maiores que as frustrações causadas.

Muito do tempo que dediquei ao trabalho foi em revisão de texto, na verdade a maior parte do tempo, principalmente das demonstrações dos lemas nas análises. Além disso, algumas vezes, achamos conveniente modificar notações, e isso acabou causando mais trabalho do que o esperado. Acredito que esse tipo de trabalho é a parte mais importante de uma monografia teórica como a minha, mas não posso negar que de certa forma é frustrante ver que às vezes o trabalho de dias não aumenta em uma página a monografia.

Decidimos não estudar um artigo que nos propusemos a estudar na proposta, pois ele se tratava de mais uma variação do Problema da Cobertura

Mínima por Conjuntos, e já haviamos estudado dois problemas desse tipo, foi então que decidimos escolher o artigo com a análise de um algoritmo para o MinMLI, que chamou muita atenção por ser um problema bem diferente do que eu vinha estudando e pelo fato de não termos encontrado na literatura uma análise melhor para esse algoritmo do que a que é fornecida nesse artigo utilizando o método dual-fitting.

As referências estudadas se mostraram bem consistentes, não encontrei eventuais erros de digitação, ou qualquer tipo de problema por parte delas que atrapalhasse significativamente o meu trabalho.

No livro de Vazirani, e principalmente no artigo de Mahdian *et al.*, os autores consideram muitas passagens como triviais e deixam a cargo do leitor, parte importantíssima da minha monografia foi mostrar que essas passagens não eram tão simples, pelo menos não para um aluno de graduação, e então tentar fazer a análise de forma bem detalhada e mais compreensível.

7.2 Disciplinas relevantes

Acredito que todas as disciplinas que cursei foram de alguma forma relevantes à minha formação em Ciência da Computação, e me fizeram melhor em algum sentido, e isso se reflete em um trabalho como esse. Mas é inegável que algumas matérias tiveram maior importância que outras para o desenvolvimento desse Trabalho de Conclusão de Curso. Comentarei um pouco sobre a importância de algumas delas.

- **MAC0110 - Introdução à Computação**

  Por ser o meu primeiro contato direto com a computação na graduação, acredito que essa disciplina tem uma importância especial mesmo não possuindo uma relação direta com a minha monografia. Isso porque ela introduz uma forma de pensar que te permite um futuro aprofundamento em várias áreas da Ciência da Computação.

- **MAT0138 - Álgebra I para Computação**
  MAT0213 - Álgebra II
  MAC0436 - Tópicos de Matemática Discreta

  Essas disciplinas contribuíram no aprendizado de como fazer demonstrações formais de teoremas, lemas e etc. De forma geral, promoveram o uso do raciocínio lógico matemático e mostraram maneiras de explicitar ideias de forma coerente. Sem dúvida, essas disciplinas tiveram,

ainda que indiretamente, forte impacto no desenvolvimento de minha monografia.

- **MAC0122 - Princípios de Desenvolvimento de Algoritmos**

  Essa é uma disciplina bem importante, pois é a primeira matéria no curso com ênfase em algoritmos, e me mostrou um caminho a seguir em Ciência da Computação. Foi nessa disciplina o meu primeiro contato com análise da complexidade de algoritmos, ainda que de forma um tanto rudimentar se comparado ao que é feito em MAC0338, introduziu uma nova preocupação na hora de escrever códigos ou de aprender algum algoritmo.

- **MAC0323 - Estruturas de Dados**

  Apesar da ênfase dessa matéria serem as estruturas de dados, ela abrange muito mais que isso. Para mim, foi como uma continuação do que aprendi em MAC0122 e me fez ter maior interesse na área de Combinatória. Conhecer o funcionamento de estruturas de dados pode se mostrar importante de várias formas, por exemplo permite fazer análises de complexidade mais justas de alguns algoritmos, ou mais que isso, uma estrutura pode ser a base da ideia por trás de um algoritmo, entre muitas outras utilidades.

- **MAC0338 - Análise de Algoritmos**

  A análise do tempo de execução de um algoritmo, além de ser por si só importante nessa monografia, introduz conceitos de como analisar um algoritmo que podem ser usados de forma análoga também em análises da razão de aproximação de um algoritmo de aproximação.

  Nessa matéria tive o meu primeiro contato com Complexidade Computacional, uma área que despertou bastante o meu interesse e que está diretamente relacionada com este trabalho.

- **MAC0328 - Algoritmos em Grafos**
  MAC0330 - Algoritmos Algébricos (Teoria dos Grafos)
  MAC0325 - Otimização Combinatória

  Grafos são estruturas importantíssimas na modelagem e entendimento de uma grande variedade de problemas, e os algoritmos que aprendemos em MAC0328 e MAC0325 abordam problemas conhecidos em grafos com diferentes estratégias, contribuindo muito com a nossa capacidade de entender as particularidades dos grafos e atacar problemas relacionados com diferentes ideias.

  Nessa monografia, pude modelar um dos problemas, o MinMLI, utilizando a linguagem aprendida em Teoria dos Grafos, ainda que eu tenha omitido essa parte da monografia. Isso facilitou muito a criação de exemplos interessantes que me ajudaram a entender o funcionamento do algoritmo proposto e criar a instância que exemplifica um dos lemas.

- **MAT0139 - Álgebra Linear para Computação**

  A importância desta matéria, no contexto deste trabalho, foi permitir melhor entendimento do que foi exposto mais tarde na disciplina Programação Linear.

- **MAC0315 - Programação Linear**

  Uma das duas disciplina que considero mais importantes no desenvolvimento desse Trabalho de Conclusão de Curso. Isso porque o método dual-fitting é totalmente dependente dos conceitos de programação linear. Como o nome sugere, o método tem tudo a ver com o que chamamos de dualidade entre programas lineares, assim como com a interpretação de problemas de otimização como programas lineares inteiros. Essa relação pode ser melhor entendida através da leitura das duas primeiras seções da monografia.

  Além do uso comum de programação linear no dual-fitting, a análise do algoritmo para o MinMLI exigiu uma segunda parte, relacionada a encontrar uma delimitação superior para a razão de aproximação, onde os entendimentos obtidos em MAC0315 foram essenciais.

- **MAC0450 - Algoritmos de Aproximação**

  Essa é a outra disciplina que considero de extrema importância para esse trabalho. Em primeiro lugar, ela é a que mais motivou a escolha do tema. Além disso, os conhecimentos aprendidos em MAC0450 são aplicados diretamente durante toda a monografia.

7.3 Trabalhos Futuros

O método dual-fitting pode ser extremamente eficaz, e ainda que seja razoavelmente recente, a literatura dispõe de várias análises que não abordei em minha monografia. Um caminho para continuar a desenvolver este trabalho é estudar detalhadamente essas análises.

Mais que isso, devido ao grande potencial do método, pode-se procurar algoritmos que nunca foram analisados com o dual-fitting e fazê-lo, eventualmente conseguindo uma análise mais justa da razão de aproximação de algum algoritmo do que a literatura dispõe até agora.

O que também pode ser feito é criar novos algoritmos de aproximação para algum problema de otimização, e tentar analisá-lo utilizando o dual-fitting. Apesar do método não ser tão simples como algumas outras análises, pelo menos é um método bem definido, ou seja, um ponto de partida caso não lhe ocorra uma forma mais simples de fazer a análise.

Quanto aos algoritmos analisados nesse trabalho, ainda que tenhamos tentado fazer a análise detalhada, sempre é possível melhorar ou deixar de forma um pouco mais clara e acessível. Em especial, o fim da análise do MinMLI pode ser tratado mais explicitamente, ainda que fuja um pouco da parte fundamental do dual-fitting.

Referências

[1] Stavros Athanassopoulos, Ioannis Caragiannis, and Christos Kaklamanis. Analysis of approximation algorithms for k-set cover using factor-revealing linear programs. *Theor. Comp. Sys.*, 45(3):555–576, 2009.

[2] M. L. Balinksi. On finding integer solutions to linear programs. In *In Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248. IBM, 1966.

[3] M.H. Carvalho, M.R. Cerioli, R. Dahab, P. Feofiloff, C.G. Fernandes, C.E. Ferreira, K.S. Guimarães, F.K. Miyazawa, J.C. Pina, J. Soares, and Y. Wakabayashi. *Uma Introdução Sucinta a Algoritmos de Aproximação.* Publicações Matemáticas do IMPA, 2001.

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

[5] Ari Freund and Dror Rawitz. Combinatorial interpretations of dual fitting and primal fitting. In *WAOA*, pages 137–150, 2003.

[6] L.G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematical Doklady*, 20:191–194, 1979.

[7] 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*, 50:127–137, 2001.

[8] V.V. Vazirani. *Approximation Algorithms.* Springer, 2001.