Universidade Federal de Pernambuco
Centro de Informática
Departamento de Sistemas da Informação

Graduação em Engenharia da Computação

Um Esquema de Aproximação Eficiente para o Problema do *Bin Packing* Unidimensional

Otávio Lucas Alves da Silva

Trabalho de Graduação

Recife
13 de julho de 2017

Universidade Federal de Pernambuco
Centro de Informática
Departamento de Sistemas da Informação

Otávio Lucas Alves da Silva

Um Esquema de Aproximação Eficiente para o Problema do *Bin Packing* Unidimensional

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Departamento de Sistemas da Informação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Prof. Ricardo Martins de Abreu Silva*

Recife
13 de julho de 2017

A todos aqueles que detém o amor pelo conhecimento.

Agradecimentos

Ao Deus que criou o universo, a existência e toda a ciência. Só por Ele é possível contemplar a razão e a beleza matemática. Sem Ele nada do que foi feito poderia se fazer.

Aos meus pais Otávio Ferreira da Silva e Adriana Patricia de Sousa Ferreira, que através de seus exemplos de vida, me deram motivação para seguir.

Ao meu orientador Ricardo Martins de Abreu Silva. Sua orientação, paciência e compreensão foram de essenciais auxílios nos momentos mais difíceis dessa jornada.

Aos meus familiares e amigos, dos quais completam o mundo em que vivo. Sem estes na minha vida, pouca razão teria para produzir.

A todos que de alguma forma me apoiaram, meus sinceros agradecimentos.

O que as suas mãos tiverem que fazer, que o façam com toda a sua força, pois na sepultura, para onde você vai, não há atividade nem planejamento, não há conhecimento nem sabedoria.

—ECLESIASTES 9:10

Resumo

O trabalho faz uma análise aprofundada sobre algoritmos de aproximação para o Problema do Bin Packing Unidimensional. Em particular, mostra como três algoritmos descritos em [KK82] atingem as taxas de aproximação especificadas com a implementação de uma rotina que resolve o relaxamento linear do problema. Apesar de tal relaxamento apresentar um número enorme de variáveis, é possível obter uma solução quase-ótima em tempo polinomial sobre o tamanho da entrada utilizando um método primal-dual. Este método é alcançado através da modificação da versão método da elipsóide descrita em [GLS81].

Palavras-chave: Empacotamento Unidimensional, Programação Linear, Método da Elipsóide, Algoritmos de Aproximação

Abstract

In this report, we make a deep analysis on approximation algorithms for the one-dimensional bin packing problem. In particular, it shows how three algorithms described in [KK82] can perform within the especified bounds with an implementation of a routine that solves a linear relaxation of the problem. Although this relaxation presents a large number of variables, it is possible to obtain a near-optimal solution in polynomial time on the input size by using a primal-dual method. This method is obtained by modifying the version of the ellipsoid method described in [GLS81].

Keywords: One-Dimensional Bin Packing Problem, Linear Programming, Ellipsoid Method, Approximation Algorithms

Sumário

CAPÍTULO 1

Introdução

O Problema do Bin Packing Unidimensional é um conhecido problema de otimização combinatória. O problema consiste em, dado uma lista L de racionais no intervalo $(0, 1]$, os organizar em um menor número de *bins* de tamanho unitário de modo que todos os números sejam empacotados e que em cada *bin* a soma dos números não exceda o valor 1. Embora seja um problema *NP*-difícil, é possível obter algoritmos de tempo polinomial que obtém soluções com uma qualidade garantida pela análise teórica. Os algoritmos mais conhecidos são os de *Best Fit*, *First Fit*, *Best Fit Decreasing* e *First Fit Decreasing*. Estes algoritmos garantem uma solução assintoticamente limitada por um fator constante em relação à solução ótima ($\frac{17}{10}$ para BF e FF e $\frac{11}{9}$ para BFD e FFD).

O Problema do Bin Packing Unidimensional é de grande importância prática, sendo utilizado em formatação de tabelas, alocação de arquivos, paginação etc.

No **Capítulo 2,** iremos dar uma breve noção de teoria da computação, teoria de complexidade, NP-completude e algoritmos de aproximação. Serão dadas importantes definições e apresentados teoremas que serão utilizados para demonstrar a dificuldade computacional do problema em questão.

No **Capítulo 3**, formalizamos o conceito geral do Problema do Bin Packing Unidimensional; demonstramos sua dificuldade computacional e limites de aproximação; e apresentamos algumas heurísticas utilizadas para resolver o problema de maneira eficiente com uma taxa de aproximação linear.

No **Capítulo 4**, falaremos sobre programação linear, apresentar e demonstrar a eficácia do método da elipsóide para resolver problemas com função objetivo linear em domínios convexos. A teoria de dualidade será brevemente citada para embasar resultados do Capítulo 5.

Finalmente, no **Capítulo 5**, apresentaremos técnicas de agrupamento e demonstraremos alguns lemas que serão úteis para o desenvolvimento dos três algoritmos descritos em [KK82]. Estes três algoritmos utilizarão uma subrotina que resolve uma versão relaxada do problema em programação linear. Mostraremos como essa subrotina pode ser implementada em tempo polinomial utilizando um método primal-dual.

No **Apêndice A**, é possível encontrar noções básicas de Álgebra Linear necessárias para a compreensão de conceitos mostrados no **Capítulo 4**.

CAPÍTULO 2

Preliminares

Este capítulo trará uma introdução sobre definições formais de algoritmo, determinismo, análise de complexidade, a questão P vs NP e algoritmos de aproximação. Não há nenhuma intenção em apresentar detalhadamente esses assuntos, mas apenas mostrar ao leitor noções básicas de análise de algoritmos para que haja um melhor entendimento dos capítulos subsequentes.

2.1 Máquinas de Turing, RAM e Algoritmos

Introduziremos esta sessão com a definição da **Máquina de Turing**, dispositivo teórico concebido pelo matemática britânico Alan Turing (1912 - 1954) para que houvesse a noção geral de algoritmo como um objeto matemático. Trata-se de um modelo abstrato completo de um computador, com memória, transições e estados. Por completo, entende-se que qualquer máquina digital pode ser modelada por uma Máquina de Turing. Esta é a *Tese de Church-Turing*, que afirma que toda função considerada naturalmente computável, pode ser computada por uma Máquina de Turing.

Definição 2.1. *Uma Máquina de Turing determinística é uma tupla*

$$(Q, \Sigma, \Gamma, q_0, \not b, F, \delta)$$

- Q *é um conjunto finito não-vazio de estados.*
- $\Sigma \subset \Gamma$ *é um alfabeto de entrada.*
- Γ *é um alfabeto finito da fita.*
- $q_0 \in Q$ *é o estado inicial.*
- $\not b \in \Sigma$ *é o símbolo em branco.*
- $F = \{q_{\text{acc}}, q_{\text{rej}}\} \subset Q$ *é o conjunto de estados finais.* q_{acc} *é chamado de estado de aceitação e* q_{rej} *de estado de rejeição.*
- $\delta : Q \times \Gamma : Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ *onde* $\leftarrow$ *e* $\rightarrow$ *indicam respectivamente, os movimentos à esquerda e à direita.*

A definição precisa de uma MT não dá ao leitor pouco familiarizado uma boa noção dos princípios de funcionamento da máquina. Para que a máquina realize um processamento é necessário que haja uma fita infinita à direita com infinitas células. Essa fita será a nossa

memória, e cada célula, uma posição de memória capaz de conter apenas um símbolo de Γ. Inicialmente, a fita estará preenchida com um número finito de símbolos de Σ diferentes de $\not b$. Desta forma, *quase todos* os símbolos presentes na fita serão $\not b$. Isso garante que a entrada tenha uma representação finita.

Inicialmente, a máquina estará no estado q_0, e com a cabeça de leitura/escrita na extremidade mais à esquerda da fita. Uma transição geral de uma MT é composta pelos seguintes passos:

1. A MT está no estado q e com a cabeça numa posição da fita com símbolo γ.
2. A MT muda para o estado q', escreve na posição atual o símbolo γ' e realiza o movimento m da cabeça. Essas operações são regidas pela função δ, onde $\delta(q,\gamma) = (q',\gamma',m)$.
3. Se $q' = q_{\text{acc}}$ ou q_{rej}, a máquina **para**. O primeiro estado indica **aceitação** da entrada, enquanto o segundo indica **rejeição** da entrada.

As MTs foram originalmente criadas para decidir se uma dada palavra $\omega \in \Sigma^*$ (sequência finita de símbolos) pertence ou não a uma linguagem $L \subset \Sigma^*$ (conjunto de palavras). Contudo, MTs também podem ser usadas para computar funções, onde a configuração de símbolos deixada na fita representa o retorno da função computada.

Mesmo sendo uma máquina de grande simplicidade em definição, as MTs podem processar qualquer tarefa que os computadores digitais podem. Entretanto, o acesso à memória da máquina pode diferir de um caso para o outro. Nas MTs, o acesso à memória acontece de forma sequencial, sendo necessário migrar de célula em célula para que seja possível ler ou escrever numa posição determinada. Nas máquinas de acesso aleatório (*Random Access Machine*, RAM), é possível ler e escrever em qualquer posição de memória apenas indicando o *endereço* desejado. Isso pode afetar no tempo de execução de um algoritmo, como veremos mais adiante ao introduzirmos o conceito de complexidade.

Definição 2.2. *Uma RAM é uma máquina composta por*

- *Um conjunto finito de variáveis* $\{x_1, \cdots, x_n\}$.
- *Uma sequência finita de instruções.*

As instruções podem ser da forma ***if-then-else***, ***goto***, $x_i \leftarrow x_i + 1$, $x_i \leftarrow x_i - 1$, $x_i \leftarrow -x_i$, $x_i \leftarrow x_j + x_k$, $x_i \leftarrow$ ***read***(x_j), ***write***(x_i) *e* ***halt***. *O* ***if-then-else*** *controla o fluxo de execução através de operadores ordinários entre as variáveis, tais como* $>, <, \geq, \leq$ *e* $=$.

A definição de RAM se torna um pouco mais amigável do que a definição de MT, uma vez que se assemelha com a ideia mais natural de algoritmo. A máquina opera sobre um **vetor** infinito indexado pelas variáveis. As funções de leitura e escrita nesse vetor são respectivamente **read** e **write**. Se em algum momento a máquina chega à instrução **halt**, ela para. Para efeitos de simplificação, consideramos que qualquer instrução leva uma unidade de tempo para ser executada.

A RAM será o modelo de computação que iremos utilizar ao longo deste trabalho. Com um pouco de esforço, é possível verificar que tudo que uma MT pode computar, uma RAM

também pode e vice-versa. Demonstrações formais para essas afirmações estão fora de escopo deste trabalho e serão omitidas. Agora podemos dar início ao desenvolvimento da teoria de complexidade de algoritmos.

2.2 Algoritmos e Complexidade

Definição 2.3. *Sejam $f, g : \mathbb{Z}^+ \to \mathbb{R}$.*

- *Denotamos $f(n) = O(g(n))$ se existem $c, k > 0$ tais que $|f(n)| \leq c|g(n)|$ para todo $n \geq k$.*
- *Denotamos $f(n) = o(g(n))$ se $g(n) = O(f(n))$.*
- *Denotamos $f(n) = \Theta(g(n))$ se $f(n) = O(g(n))$ e $f(n) = o(g(n))$.*

A notação Θ induz uma relação de equivalência entre as funções f e g e podemos escrever $f \sim g$ para denotar $f(n) = \Theta(g(n))$. A classe de equivalência da relação $\sim$ representada por f é chamada de *taxa de crescimento de* $f(n)$.

Definição 2.4. *Dizemos que um algoritmo A roda em tempo $O(f(n))$, onde $f : \mathbb{Z}^+ \to \mathbb{R}$ se o número de intruções executadas $t(n)$ para uma entrada de tamanho n satisfaz a relação $t(n) = O(f(n))$. Ou seja, a quantidade de passos que o algoritmo A leva para executar uma entrada de tamanho n é limitada por um múltiplo da função $f(n)$, para n suficientemente grande.*

Por brevidade, dizemos que A é assintoticamente limitado por $f(n)$.

Definição 2.5. *Um algoritmo A é um algoritmo de tempo polinomial se ele roda em tempo $O(n^k)$ para k positivo.*

Os algoritmos de tempo polinomial são de grande importância para computação, pois são algoritmos que na prática executam em tempo tratável. Definiremos agora a classe de problemas de decisão que rodam em tempo polinomial.

Definição 2.6. *Seja Σ um alfabeto finito não vazio. Definimos P pelo conjunto de todas as linguagens $L \subset \Sigma^*$ tais que existe um algoritmo A que decide se uma dada palavra $\omega \in \Sigma^*$ pertence a L em tempo polinomial sobre o comprimento de ω.*

P é conhecida como a "classe de problemas polinomiais", isto é, a classe de todas as linguagens que podem ser decididas em tempo polinomial. Iremos agora definir uma outra classe de problemas, que contém a classe de problemas P, mas com definição mais abrangente.

Definição 2.7. *Seja Σ como na definição anterior. Definimos NP pelo conjunto de todas as linguagens $L \subset \Sigma^*$ tais que existe um algoritmo A e um certificado $c \in \Sigma^*$ tal que A alimentado pela entrada $\langle \omega, c \rangle$ decide se uma palavra $\omega \in \Sigma^*$ pertence a L em tempo polinomial sobre o comprimento de ω.*

Note que trivialmente $P \subset NP$, pois é só tomar $c = \varepsilon$.[1] Esse certificado c é a palavra do qual é possível verificar se a entrada ω pertence à linguagem L. NP é conhecida como a "classe de problemas verificados em tempo polinomial", isto é, dado uma entrada e um verificador, é possível dizer em tempo polinomial se a entrada pertence à linguagem. Tome por exemplo, o problema que determina se uma fórmula booleana ϕ é satisfatível: um certificado seria uma valoração das variáveis $\psi_1, \cdots, \psi_n$ de ϕ de tal modo que ϕ é calculada como verdadeiro. De fato, o problema acima descrito é conhecido como o problema SAT, e $SAT \in NP$. Até o atual momento, não se sabe se P é um conjunto próprio de NP ou se P é exatamente o conjunto NP. Acredita-se, no entanto, que $P \neq NP$ em vista de que ainda não foram descobertos a existência de algoritmos de tempo polinomial com certificado vazio para certos problemas em NP. Estes problemas, dos quais iremos descrever na próxima sessão, são chamados de problemas NP-completos.

2.3 NP-completude

Seja Σ um alfabeto finito e não vazio.

Definição 2.8. *Sejam duas linguagens $L_1, L_2 \in \Sigma^*$. Dizemos que $L_1 \leq_p L_2$ se existe uma função $f : \Sigma^* \to \Sigma^*$ que executa em tempo polinomial tal que $\omega \in L_1 \iff f(\omega) \in L_2$. Dizemos que L_1 é polinomialmente redutível a L_2.*

Note que a relação $\leq_p$ é transitiva e reflexiva.

Definição 2.9. *Dizemos que $L \in NP$ é NP-completo se para todo $L' \in NP$, $L' \leq_p L$.*

Teorema 2.1. *Se L é NP-completo e $L \in P$, então $P = NP$.*

O teorema acima destaca a importância dos problemas NP-completos. O teorema a seguir dá um elemento maximal em NP segundo a relação $\leq_p$, isto é, o nosso primeiro exemplo de um problema NP-completo.

Teorema 2.2 (Teorema de Cook-Levin)**.** *Seja SAT o problema da satisfatibilidade de fórmulas boolenas na forma normal conjuntiva. SAT é NP-completo.*

Embora não iremos demonstrar esse teorema, utilizaremos este resultado para demonstrar que outros problemas são NP-completos.

Definição 2.10. *Se um problema L é NP-completo e $L \leq_p L'$, dizemos L' é NP-difícil.*

Note que se $L' \in NP$, então L' é NP-completo. Desta forma, o estudo de problemas NP-difíceis também são de grande importância para a teoria de complexidade de algoritmos.

[1] Neste contexto, ε detona a palavra vazia, isto é, a palavra que não contém nenhum símbolo do alfabeto

Alguns Problemas *NP*-completos

A seguir, vamos apresentar alguns outros problemas *NP*-completos, juntamente com a demonstração desse fato.

Lema 2.1. *Seja 3SAT o problema da satistfatibilidade de fórmulas booleanas na forma normal conjuntiva onde cada cláusula tem exatamente três literais. 3SAT é NP-completo.*

Demonstração. $3SAT$ está em NP, pois o verificador é simplesmente uma valoração das variáveis da fórmula booleana. Para demonstrar que $SAT \leq_p 3SAT$, vamos definir uma função f que transforma uma instância de SAT em uma instância de $3SAT$. Como uma entrada de SAT está na forma normal conjuntiva, suas cláusulas têm a forma de um literal, dois literais, três literais ou mais literais. Para cada um dos casos, fazemos:

- Um lietaral: substitua x_1 por $(x_1 \vee a \vee b) \wedge (x_1 \vee a \vee \bar{b}) \wedge (x_1 \vee \bar{a} \vee b) \wedge (x_1 \vee \bar{a} \vee \bar{b})$.
- Dois literais: substitua $x_1 \vee x_2$ por $(x_1 \vee x_2 \vee a) \wedge (x_1 \vee x_2 \vee \bar{a})$.
- Três literais: não substitua.
- Mais literais: substitua $x_1 \vee \cdots \vee x_l$ por $(x_1 \vee x_2 \vee a_1) \wedge (\bar{a}_1 \vee x_3 \vee a_2) \wedge \cdots \wedge (\bar{a}_{l-3} \vee x_{l-1} \vee x_l)$

Note que essa redução claramente executa em tempo polinomial e $\omega \in SAT \iff f(\omega) \in 3SAT$. Portanto $3SAT$ é também NP-difícil, logo NP-completo. □

Lema 2.2. *Seja SS (Subset Sum) o problema de determinar se dada uma lista de inteiros A e um inteiro s, existe um subconjunto de $B \subset A$ tal que*

$$\sum_{b \in B} b = s$$

SS é NP-completo.

Demonstração. SS é claramente NP, pois o certificado consiste apenas na sublista B de A tal que a soma dos inteiros em B resulte em s. Vamos demonstrar que $3SAT \leq_p SS$. Dada uma instância de $3SAT$, iremos transformá-la numa instância de SS em tempo polinomial sobre o tamanho da entrada. Suponha que a entrada para $3SAT$ tem n variáveis x_i e m cláusulas c_j. Para cada variável x_i, forme números t_i e f_i de $n+m$ dígitos cada, de tal forma que:

- O i-ésimo dígito de t_i e f_i é 1.
- Para $n+1 \leq j \leq n+m$, o j-ésimo dígito de t_i é igual a 1 se x_i está em c_{j-n}.
- Para $n+1 \leq j \leq n+m$, o j-ésimo dígito de f_i é igual a 1 se $\bar{x}_i$ está em c_{j-n}.
- Todos os outros dígitos de t_i e f_i são 0.

Para cada cláusula c_j, forme números x_j e y_j de $n+m$ dígitos tais que

- O $(n+j)$-ésimo dígito de x_j e y_j é 1.
- Todos os outros dígitos de x_j e y_j são 0.

Por fim, construa s de $n+m$ dígitos tais que os dígitos de 1 a n são 1 e os dígitos de $n+1$ a $n+m$ são 3. Seja a lista A os números t_i, f_i para $1 \leq i \leq n$ e x_j, y_j para $1 \leq j \leq m$ e s a soma. Primeiro, vamos demonstrar que se a fórmula booleana é satisfatível, então A tem um subconjunto com soma s. Tome t_i se x_i é verdadeiro e f_i caso contrário. Tome x_j se o número máximo de literais em c_j é 2 e tome y_j se o número de literais em c_j é 1. Se todas as cláusulas c_j forem verdadeiras, então um, dois ou três literais são verdadeiros. No caso de apenas um, temos y_j, x_j e um entre os f_i ou t_i. Se dois, então temos y_j e dois entre os t_i e f_i. Se temos três literais verdadeiros, então temos três entre os t_i e f_i, mas não x_j e y_j. Isto resulta que os dígitos de $n+1$ até $n+m$ da soma são de valor 3.

Agora, vamos demonstrar que se a soma existe, então a fórmula booleana é satisfatível. Tome x_i verdadeiro se t_i está no subconjunto e x_i falso se f_i está no subconjunto. Note que no subconjunto, devemos ter exatamente um entre os números t_i e f_i, pois se não tivéssemos algum ou os dois, os primeiros dígitos de s não coincidiriam. Pelo menos um número correspondente a um literal numa cláusula deve estar no subconjunto, pois caso contrário, pelo menos um dos últimos m dígitos da soma seria menor que 3. Se isso acontece, então cada cláusula é verdadeira.

Como essa redução é feita em tempo polinomial, portanto $3SAT \leq_p SS$, o que conclui a argumentação. □

Lema 2.3. *Seja PP (Problema da Partição) o problema de determinar se dada uma lista de inteiros S, existe um modo de partir S em duas listas $A \subset S$ e $A^c = S - A$ de modo que*

$$\sum_{a \in A} a = \sum_{a \in A^c} a$$

PP é NP-completo.

Demonstração. Basta provarmos que $SS \leq_p PP$, pois PP claramente está em NP. Seja A e s a lista e a soma para o problema SS. Seja t a soma dos elementos em A. Faça $A' = A \cup \{t - 2s\}$ e transforme essa lista na entrada para o PP.

Se A' tem uma partição cuja soma dos elementos das listas resultantes são iguais, então esta soma é $t - s$. Tome por B a lista que contém o elemento $t - 2s$. Se o removermos, teremos uma lista cuja soma é exatamente s. Portanto A tem uma sublista com soma s.

Se A tem uma sublista com soma s, então tome essa sublista e adicione o elemento $t - 2s$. As duas listas restantes tem soma $s - t$. A redução é claramente feita em tempo polinomial, portanto $SS \leq_p PP$. □

2.4 Algoritmos de Aproximação

No estudo de algoritmos de otimização combinatória, é de particular interesse problemas NP-difíceis. Obter soluções ótimas para instâncias desses problemas é uma tarefa custosa e muitas

vezes não existem recursos computacionais suficientes para executar tais algoritmos. Nesta sessão iremos dar uma pequena ideia do que são algoritmos de aproximação e seus usos para obter soluções nem sempre ótimas, mas com qualidades garantidas pela teoria.

Considere um problema de otimização e I uma instância. Denote $\text{opt}(I)$ como o valor ótimo da instância I; $\text{val}(\cdot)$ como a função que determina o valor de uma solução factível; e $A(I)$ como a solução factível gerada pelo algoritmo A para a instância I. De início, vamos assumir que estamos apenas lidando com problemas de minimização. Portanto, A é um algoritmo de aproximação se existe um $\alpha \geq 1$ para cada I tal que

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

É notável que se α é 1 para todo I, então A é um algoritmo **exato**. De modo geral, dizemos que A é uma α-aproximação para o problema em questão. O número α pode depender de I, pode ser uma constante e às vezes pode ser tão próximo de 1 quanto se queira. Diremos que um algoritmo de aproximação é polinomial se ele executa em tempo polinomial sobre o tamanho de I.

2.4.1 Inaproximabilidade

Com o estudo dos algoritmos de aproximação, podemos classificar de maneira rigorosa a diferença de dificuldade entre problemas de otimização *NP*-difíceis. Embora, de modo geral, os problemas *NP*-difíceis sejam considerados problemas intratáveis, alguns destes problemas parecem ser mais difíceis que outros. De fato, existem problemas *NP*-difíceis que admitem algoritmos de aproximação que obtém soluções com valor tão perto da solução ótima quanto se queira, enquanto outros problemas sequer admitem um fator constante de aproximação a menos da premissa que $P = NP$.

Nesta sessão, iremos definir cinco classes de problemas de otimização de acordo com seus graus de aproximabilidade. Estas classes são $PO, FPTAS, PTAS, APX$ e NPO. Vamos definir um problema de otimização com mais rigor a seguir.

Definição 2.11. *Um problema de otimização* Π *é caracterizado por: dada uma instância I do problema* Π*, encontrar uma solução S no espaço de soluções factíveis de modo a minimizar* $\text{val}(I,S)$.

Uma solução ótima é uma solução factível que minimize a função $\text{val}(\cdot,\cdot)$. A classe de problemas *NPO*, que é uma extensão da definição de *NP* para problemas de otimização, é formada por problemas com as seguintes características:

- O tamanho de S é assintoticamente limitado por um polinômio sobre o tamanho da instância I.
- Existe um algoritmo polinomial que determina se uma palavra é representação válida de uma instância do problema.
- Existe um algoritmo polinomial que determina se uma representação válida é uma solução factível do problema.

- Existe um algoritmo polinomial que calcula o valor de uma solução para uma determinada instância, isto é, val(I,S) que calcula o valor da solução S para a instância I.

Denotamos PO o conjunto de todos os problemas em NPO para os quais existem algoritmos exatos de tempo polinomial. Vamos agora definir as classes $FPTAS, PTAS$ e APX.

Definição 2.12. *APX é a classe de todos os problemas de otimização em NPO para os quais existem uma α-aproximação que o resolvem para α constante.*

Definição 2.13. *$PTAS$ é a classe de todos os problemas de otimização em NPO para os quais, para qualquer $\varepsilon > 0$ racional constante, existe um algoritmo de aproximação polinomial A_ε*

$$\text{val}(I, A_\varepsilon(I)) \leq (1+\varepsilon)\text{opt}(I)$$

A é denotado por ***esquema de aproximação****.*

Definição 2.14. *$FPTAS$ é a classe de todos os problemas de otimização em NPO para os quais, para qualquer $\varepsilon > 0$ racional, existe um algoritmo de aproximação polinomial A_ε que roda em tempo assintoticamente limitado por um polinômio $p(\langle I \rangle, \varepsilon^{-1})$ com*

$$\text{val}(I, A_\varepsilon(I)) \leq (1+\varepsilon)\text{opt}(I)$$

A é detonado por ***esquema de aproximação polinomial****.*

Das definições, segue imediatamente que $PO \subseteq PTAS \subseteq FPTAS \subseteq APX \subseteq NPO$. Enunciaremos agora um teorema que determina uma hierarquia de dificuldade de problemas de otimização utilizando a relação acima.

Teorema 2.3. *Considere a relação*

$$PO \subseteq PTAS \subseteq FPTAS \subseteq APX \subseteq NPO$$

Se uma dessas relações de estar contido não for estrita, então $P = NP$.

Embora não seja de grande dificuldade, não iremos demonstrar este teorema. No próximo capítulo, porém, iremos mostrar que o Problema do Bin Packing está contido em APX, e que ele estiver contido em $PTAS$, então teríamos $P = NP$.

CAPÍTULO 3

O Problema

3.1 Introdução

Sem que haja preocupação com formalidades, o Problema do Bin Packing Unidimensional, ou simplesmente Problema do Bin Packing (PBP), pode ser definido da seguinte maneira: dada uma coleção finita U de peças de tamanho arbitrário e racional no intervalo $[0,1)$, determinar o número mínimo de *bins* de tamanho unitário que possam empacotar, sem sobreposição, todos os elementos da coleção. De maneira mais precisa, seja o conjunto de índices das peças $\iota = \{1,\cdots,n\}$ e $(s_i)_{i\in\iota} \subset [0,1)\cap\mathbb{Q}$ a sequência dos tamanhos das peças indexadas por ι. Determinar $\phi : \iota \to \iota$ com a restrição de que $\sum_{i\in\phi^{-1}(j)} s_i \leq 1$ para todo $j \in \mathrm{Im}(\phi)$ de modo a minimizar $|\mathrm{Im}(\phi)|$.

Podemos, agora, modelar o problema da seguinte maneira:

$$\min \sum_{i=1}^{n} y_i \tag{3.1}$$

com as restrições

$$\sum_{j=1}^{n} s_j x_{ij} \leq y_i, \quad i \in \iota = \{1,\cdots,n\}, \tag{3.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j \in \iota, \tag{3.3}$$

$$x_{ij} \in \{0,1\}, \qquad i,j \in \iota, \tag{3.4}$$

$$y_1 \in \{0,1\}, \qquad i \in \iota \tag{3.5}$$

onde as variáveis da forma x_{ij} e y_i podem ser interpretadas da seguinte forma:

$$y_i = \begin{cases} 1, & \text{se o } \textit{bin } i \text{ está sendo usado} \\ 0, & \text{c.c.} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{se o } \textit{bin } i \text{ contém a peça } j \\ 0, & \text{c.c.} \end{cases}$$

Este capítulo trará alguns dos principais algoritmos de aproximação existentes na literatura para resolver instâncias arbitrárias PBP. Apenas no **Capítulo 4** serão apresentados os três algoritmos de aproximação descritos em [KK82] e suas respectivas demonstrações das taxas de aproximação.

3.1.1 Notação

Uma instância I do PBP será dada pelo par $(\iota, (s_i)_{i\in\iota})$. Definiremos as seguintes notações:

- $\text{size}(I) := \sum_{i\in\iota} s_i$.
- $n(I) := |\iota|$.
- $m(I) :=$ "o número de peças de tamanhos diferentes da instância I".
- $\alpha(I) := \min_{i\in\iota} s_i$.
- $\text{opt}(I) :=$ "número ótimo de *bins* para empacotar a instância I".
- Se A é um algoritmo factível[1] para o PBP, então $A(I) :=$ "número de *bins* retornado por A".

A instância I poderá ser omitida da notação quando exposta no contexto ou quando não houver necessidade em citá-la explicitamente.

3.1.2 Um Pouco de História

Grande parte dos trabalhos a respeito do PBP mostraram a existência de algoritmos de aproximação que executam em tempo $O(n\log n)$ respeitando inequações da seguinte forma $A(I) \leq \alpha\cdot\text{opt}(I) + o(\text{opt}(I))$. Em uma série de artigos publicados entre 1973 e 1980, a constante α foi gradualmente reduzida de $\frac{17}{10}$ para $\frac{71}{60}$.

Em 1981, Fernandez de la Vega e Lueker mostraram em [dlVL81] um esquema de aproximação A que dados uma instância I e um racional positivo ε, teria-se $A(I,\varepsilon) \leq (1+\varepsilon)\text{opt}(I) + O(\varepsilon^{-2})$ que, para ε constante, executava em tempo $O(n(I))$, mas seu desempenho se tornava pior do que exponencial conforme ε se aproxima de 0. Em [JDU+74], David Johnson mostrou que se ε dependesse da instância I, poderia-se obter um algoritmo de tempo polinomial A tal que $A(I) \leq \text{opt}(I) + o(\text{opt}(I))$, e utilizando um caso particular do esquema de aproximação, Karp e Karmarkar demonstraram em [KK82] que valeria a desigualdade $A(I) \leq \text{opt}(I) + O(\text{opt}(I)^{\delta})$ para $0 < \delta < 1$.

Karp e Karmarkar apresentam em [KK82] três algoritmos que alcançam progressos ainda maiores do que os obtidos por Fernandez de la Vega e Lueker em [dlVL81]. Os resultados dependem exclusivamente de resolver em tempo polinomial um relaxamento em programação linear do PBP. As soluções do problema de programação linear representam empacotamentos de um único *bin*, dentro de um conjunto de configurações permitidas. Cada configuração poderá ser usada um número fracionário de vezes, ao invés de inteiro. Grötschel, Lovász e Schrijver demonstraram em [GLS81] que utilizando uma versão modificada do Método da Elipsóide, o problema de otimização linear no sentido fraco em espaços convexos com certas propriedades[2] poderia ser resolvido em tempo polinomial.

Seja $T(m,n)$ uma função limitada por um polinômio em duas variáveis. Os principais resultados que discutiremos no **Capítulo 4** seguem:

[1] Um algoritmo factível para é um algoritmo que retorna apenas soluções factíveis.

[2] A propriedade requerida é de que exista um *oráculo* que executa em tempo polinomial que determina se um ponto x_0 é factível ou retorna uma direção de busca.

- Existe um algoritmo de aproximação A que roda em tempo $O(T(\text{size}(I), n(I)))$ tal que para toda instância I, $A(I) \leq \text{opt}(I) + O(\log^2 \text{opt}(I))$.
- Existe um algoritmo de aproximação A que roda em tempo $O(T(m(I), n(I)))$ tal que para toda instância I, $A(I) \leq \text{opt}(I) + O(\log^2 m(I))$.
- Para todo $0 < \delta < 1$, existe um algoritmo de aproximação A que roda em tempo $O(T(\text{size}(I)^{1-\delta}, n(I)))$ tal que para toda instância I, $A(I) \leq \text{opt}(I) + O(\text{opt}(I)^{\delta})$.
- Para todo $\varepsilon > 0$, existe um algoritmo de aproximação A que roda em tempo $O(T(\varepsilon^{-2}, n(I)))$ tal que para toda instância I, $A(I) \leq (1+\varepsilon)\text{opt}(I) + O(\varepsilon^{-2})$.

3.2 Inaproximabilidade

O PBP é *NP*-difícil. Determinar se uma instância tem como ótimo dois *bins* é um problema *NP*-completo.

Teorema 3.1. *O problema de decisão para determinar se uma instância I do PBP admite uma solução factível com dois bins é NP-completo.*

Demonstração. Considere o Problema da Partição, que pelo lema 2.3 é *NP*-completo. Dados $\iota = \{1, \cdots, n\}$ e $(s_i)_{i\in\iota} \subset \mathbb{Q}^+$, o problema consiste em determinar se existe $\iota' \subset \iota$ tal que $\sum_{i\in\iota'} s_i = \sum_{j\notin\iota'} s_j$. Agora considere a instância I do PBP tal que $I = (\iota, (a_i)_{i\in\iota})$ onde $a_i = \frac{s_i}{2\sum_{j\in\iota} s_j}$ para todo $i \in \iota$. A instância I admite como solução dois *bins* se, e somente se a instância do Problema da Partição admite resposta positiva. Como essa redução pode ser feita claramente em tempo polinomial, então o problema de decisão associado ao PBP é *NP*-completo. □

Corolário 3.1. *O PBP não admite uma α-aproximação para $\alpha < \frac{3}{2}$, e portanto não pertence a PTAS, a menos que $P = NP$.*

Demonstração. Suponhamos que exista $\alpha < \frac{3}{2}$ para que haja uma α-aproximação A para o PBP. Considerando a redução polinomial vista na demonstração acima, se $\text{opt}(I) = 2$, então $A(I) \leq \alpha \cdot 2 < 3 \Rightarrow A(I) = 2$. Portanto, o Problema da Partição poderia ser decidido em tempo polinomial, o que nos daria $P = NP$. □

Corolário 3.2. $APX = PTAS \Longrightarrow P = NP$.

Para o corolário acima, assumimos que $PBP \in APX$. Veremos que isso é verdade nas próximas seções.

3.3 Heurísticas

As heurísticas para resolver o PBP podem ser classificadas em duas categorias: as *onlines* e as *offlines*. Nas heurísticas *onlines*, os itens chegam em ordens desconhecidas, sendo necessário empacotar uma peça para que se possa empacotar a próxima. Não é permitido remoção de item

em algum *bin*. Para as heurísticas *offlines*, há um conhecimento prévio de toda a entrada, sendo possível realizar remoção e relocação dos itens.

É natural pensar que uma heurística *online* é mais difícil de resolver satisfatoriamente. De fato, podemos construir instâncias de tal modo que force o algoritmo *online* a nunca atingir a solução ótima. Vamos provar o seguinte teorema:

Teorema 3.2. *Não existe heurística online A tal que para qualquer instância I do PBB,* $A(I) \leq \alpha \cdot \text{opt}(I)$, *com* $\alpha < \frac{4}{3}$.

Esta é uma afirmação bastante intuitiva que se tornaria consequência imediata do Corolário 3.1 caso $P \neq NP$.

Demonstração. Considere a instância $I = (\iota, (s_i)_{i \in \iota})$ tal que $\iota = \{1, \cdots, 2m\}$ e $s_i = 1/2 - \varepsilon$, para $1 \leq i \leq m$ e $s_i = 1/2 + \varepsilon$ para $m+1 \leq i \leq 2m$, $0 < \varepsilon < 1/6$. É fácil de observar que o ótimo da instância utiliza $2m$ *bins*. Suponha, por absurdo, que tal heurística exista, com $\alpha < \frac{4}{3}$. Ao receber os m primeiros itens, todos de tamanho $1/2 - \varepsilon$, teremos uma instância I' cuja solução ótima utiliza $m/2$ *bins* (consideramos m par). Logo, temos

$$A(I') \leq \alpha \cdot \frac{m}{2} < \frac{4}{3} \cdot \frac{m}{2} \Longrightarrow \frac{A(I')}{m} < \frac{2}{3} \tag{3.6}$$

Vamos agora considerar o empacotamento dos m itens restantes, todos de tamanho $1/2 + \varepsilon$. Dado o empacotamento até agora, temos $2A(I') - m$ *bins* com espaço para empacotar itens de tamanho $1/2 + \varepsilon$. Todos os itens excedentes, i.e., que não forem empacotados nos *bins* ocupados com as m primeiras peças, deverão ser empacotados em um único *bin* por item. Desta forma, o algoritmo deveria criar pelo menos $m - (2A(I') - m) = 2m - 2A(I')$ *bins* adicionais, o que nos daria

$$2m - A(I') \leq \alpha \cdot m < \frac{4}{3} \cdot m \Longrightarrow \frac{A(I')}{m} > \frac{2}{3} \tag{3.7}$$

Que claramente contradiz a inequação 3.6. □

Corolário 3.3. *Não existe heurística online que sempre retorne soluções ótimas.*

3.3.1 Heurísticas *Online*

Falaremos nesta sessão de algumas heurísticas *online*, sendo estas o *Next Fit*, *First Fit* e *Best Fit*. Devido a extrema simplicidade, todas os algoritmos citados executam em tempo $O(n^2)$, e utilizando estruturas de dados adequadas, podemos alcançar tempo de execução $O(n \log n)$.

Talvez a mais simples de todas seja a *Next Fit*, que executa em tempo $O(n)$. O algoritmo procede da seguinte forma: ao chegar um novo item, verifica se há espaço no último *bin*. Caso não haja, um novo *bin* é criado, e todos os *bins* anteriores nunca são revisitados. É uma heurística incrivelmente simples de implementar. Da mesma forma é sua análise de pior caso.

Teorema 3.3. *Dada uma instância arbitrária I do PBP, se A implementa a heurística de Next Fit, então* $A(I) \leq 2 \cdot \text{opt}(I)$. *Além disso, existem instâncias I tais que* $A(I) \geq 2 \cdot \text{opt}(I) - 2$.

Demonstração. Seja $I = (\iota, (s_i)_{i\in\iota})$ com $\iota = \{1, \cdots, n\}$ e $\phi : \iota \to \iota$ o mapeamento realizado pelo algoritmo A. Vamos mostrar um resultado um pouco mais forte do que o teorema afirma:

$$A(I) \leq 2 \cdot \text{opt}(I) - 1$$

Antes, note que $\lceil \text{size}(I) \rceil$ é um limite inferior para $\text{opt}(I)$. É razoável afirmar que $\max(\text{Im}(\phi)) = A(I)$, caso contrário, poderíamos obter ϕ' com essa propriedade, rearrumando o mapeamento e mantendo uma solução equivalente. Para $j = 1, \cdots, \lfloor A(I)/2 \rfloor$. Pela característica do *Next Fit*, temos inequações da seguinte forma

$$\sum_{\phi(i)\in\{2j-1,2j\}} s_i > 1$$

Somando todas as inequações, temos o seguinte resultado:

$$\text{opt}(I) - 1 \geq \lceil \text{size}(I) \rceil - 1 \geq \lfloor A(I)/2 \rfloor \geq \frac{A(I)-1}{2} \Longrightarrow A(I) \leq 2 \cdot \text{opt}(I) - 1 \tag{3.8}$$

Para o limite inferior, considere a instância $I = (\iota, (s_i)_{i\in\iota})$ onde $\iota = \{1, \cdots, 4n\}$, $s_{2k-1} = 1/2$ e $s_{2k} = \frac{1}{2n}$. A solução ótima utiliza $n+1$ *bins*, todos com capacidade completa. A rotina de *Next Fit* irá utilizar *bins* com um item de tamanho $1/2$ e outro com tamanho $\frac{1}{2n}$ cada, totalizando $2n$ *bins*. Portanto, essa entrada força o algoritmo a utilizar $2 \cdot \text{opt}(I) - 2$ *bins*. □

O algoritmo de *First Fit* faz algo diferente do algoritmo de *Next Fit*. Ele procura pelo primeiro *bin* com espaço suficiente para empacotar o próximo número. Se não houver, ele cria um novo *bin*. O algoritmo de *Best Fit* realiza algo similar, mas ao invés de encaixar no primeiro *bin* com espaço suficiente, ele procura o *bin* com espaço suficiente que está mais próximo de ser cheio. Descreveremos aqui um resultado obtido em [JDU$^+$74] a respeito das heurísticas FF e BF.

Teorema 3.4. *Para qualquer instância I, os algoritmos FF e BF retornam uma solução factível com no máximo*

$$\frac{17}{10} \cdot \text{opt}(I) + 2$$

bins. E para cada n, existe I com $n = n(I)$ tal que os algoritmos retornam pelo menos $\frac{17}{10} \cdot \text{opt}(I) - 8$ bins.

3.3.2 Heurísticas *Offline*

Nesta sessão, analisaremos superficialmente heurísticas com taxas de aproximação similares: *First Fit Decreasing* e a *Best Fit Decreasing*. Para a *FFD*, ordenaremos os itens em ordem não-crescente de tamanho, i.e., dada uma instância $I = (\iota, (s_i)_{i\in\iota})$, geraremos uma instância equivalente $I' = (\iota, (s'_i)_{i\in\iota})$ tal que existe uma bijeção $r : \iota \to \iota$ em que $s'_i = s_{r(i)}$ e $s'_i \geq s'_j \Longrightarrow i \leq j$. Em seguida aplicaremos a rotina de *First Fit* sobre a instância I'. De modo similar, é definida a heurística BFD.

Dada as definições, vamos enunciar um teorema a respeito dos algoritmos FFD e BFD.

Teorema 3.5. *Dada uma instância arbitrária I do PBP, se A implementa a heurística de FFD ou BFD, então $A(I) \leq 3/2 \cdot \text{opt}(I)$. Além disso, A executa em tempo $O(n^2)$.*

Demonstração. Considere uma instância $I = (\iota, (s_i)_{i\in\iota})$ com $(s_i)_{i\in\iota}$ monotônica não-crescente. Seja ϕ o mapeamento realizado pela heurística A e $j = \lceil 2/3A(I) \rceil$ o índice do j-ésimo *bin* mapeado por ϕ (estamos novamente considerando que $\max(\text{Im}(\phi)) = A(I)$ e que o mapeamento acontecerá de forma crescente em relação à ordem dos itens). Se o *bin* j contém uma peça i tal que $s_i > 1/2$, então para todos os *bins* de índice $j' < j$, temos itens de tamanho maior que $1/2$. Logo, temos no mínimo j itens com tamanho maior que $1/2$, que claramente devem ser empacotados em diferentes *bins*. Portanto

$$\text{opt}(I) \geq \left\lceil \frac{2}{3}A(I) \right\rceil \geq \frac{2}{3}A(I) \Longrightarrow \frac{3}{2} \cdot \text{opt}(I) \geq A(I)$$

Suponha agora que o *bin* j não contém um item com tamanho maior que $1/2$, e consequentemente, qualquer *bin* com índice $j' > j$ não contém tal item. Desta forma, todos os *bins* com índices $j, \cdots, A(I)$ contém pelo menos $2(A(I) - j) + 1$ itens, onde nenhum deles coube nos *bins* de índice $1, \cdots, j-1$. Portanto, se $j - 1 \leq 2(A(I) - j) + 1$, então $\text{size}(I) > j - 1$ e de forma similar, se $j - 1 \geq 2(A(I) - j) + 1$, então $\text{size}(I) > 2(A(I) - j) + 1$. Logo

$$\begin{aligned} \text{size}(I) &> \min\{j-1, 2(A(I)-j)+1\} \\ &\geq \min\{\lceil 2/3A(I) \rceil - 1, 2(A(I) - 2/3(A(I)+1)) + 1\} \\ &= \lceil 2/3A(I) \rceil - 1 \end{aligned}$$

Portanto

$$\text{opt}(I) > \left\lceil \frac{2}{3}A(I) \right\rceil - 1 \Longrightarrow \text{opt}(I) \geq \left\lceil \frac{2}{3}A(I) \right\rceil \geq \frac{2}{3}A(I) \Longrightarrow A(I) \leq \frac{3}{2} \cdot \text{opt}(I)$$

O algoritmo claramente executa em tempo $O(n^2)$, o que completa a demonstração. □

Note que, pelo Corolário 3.1, esta possivelmente é a melhor α-aproximação que podemos obter para o PBP. Não iremos dar uma análise mais detalhada para os algoritmos de FFD e BFD. Contudo, em [JDU+74] é possível observar os seguintes resultados:

Teorema 3.6. *Para qualquer instância I, os algoritmos FFD e BFD retornam uma solução factível com no máximo*

$$\frac{11}{9} \cdot \text{opt}(I) + 4$$

bins. E para cada n, existe I com $n = n(I)$ tal que os algoritmos retornam pelo menos $\frac{11}{9} \cdot \text{opt}(I) - 2$ bins.

CAPÍTULO 4

Programação Linear

Antes de discutirmos a respeito de problemas de programação linear, vamos definir uma relação de ordem parcial entre vetores do $\mathbb{R}^n$.

Definição 4.1. *Seja $\geq$ uma relação de ordenação parcial e $>$ uma relação no $\mathbb{R}^n$. Sejam $x = (x_1, \cdots, x_n)^T$ e , $y = (y_1, \cdots, y_n)^T$ pertecentes ao $\mathbb{R}^n$. Definimos*

1. $x \geq y \iff x_i \geq y_i$ *para* $i = 1, \cdots, n$.
2. $x > y \iff x_i > y_i$ *para* $i = 1, \cdots, n$.

De forma similar, definimos as relações $\leq$ e $<$.

4.1 Introdução

Um problema de programação linear (PPL) pode ser definido como um problema de otimização cuja função objetivo e restrições de igualdade/desigualdades são lineares. Sejam, $A_1 \in \mathbb{Q}^{m_1 \times n}$ com $m_1 < n$, $A_2 \in \mathbb{Q}^{m_2 \times n}$, $c \in \mathbb{Q}^n$, $b_1 \in \mathbb{Q}^{m_1}$ e $b_2 \in \mathbb{Q}^{m_2}$. O seguinte problema de otimização é um PPL:

$$\min / \max \ c^T x \tag{4.1}$$
$$A_1 x = b_1 \tag{4.2}$$
$$A_2 x \geq b_2 \tag{4.3}$$
$$x \in \mathbb{R}^n \tag{4.4}$$

4.1 - 4.4 definem o que chamamos de um PPL na *forma geral*. Vamos definir dois tipos especiais de PPL, que são os da *forma padrão* e *forma canônica*.

Definição 4.2. *Sejam $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$ e $b \in \mathbb{Q}^m$.*

1. *Um PPL está na forma canônica se é escrito da seguinte forma:*

$$\begin{aligned} \min \ & c^T x \\ & Ax \geq b \\ & x \geq 0 \end{aligned}$$

2. *Um PPL está na forma padrão se $m < n$ e é escrito da seguinte forma:*

$$\begin{aligned} \min \ & c^T x \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

Vamos enunciar um teorema a respeito da equivalência das formas de um PPL.

Teorema 4.1. *As formas geral, canônica e padrão são equivalentes, i.e., representam a mesma classe de problemas de otimização.*

Demonstração. Obviamente, um PPL na forma padrão ou canônica, já está na forma geral. Considera a conversão da forma geral para a forma canônica. Para a função objetivo, é fácil ver que max $c^T x$ é equivalente a min $-c^T x$. Para as restrições de desigualdade, não há nada a fazer. Para cada restrição de igualdade da forma

$$a_1 x_1 + \cdots + a_n x_n = b$$

substituímos por duas restrições da forma

$$\begin{aligned} a_1 x_1 + \cdots + a_n x_n &\geq b \\ -a_1 x_1 - \cdots - a_n x_n &\geq -b \end{aligned}$$

Para acrescentar as restrições de sinal, para cada variável x_i no PPL na forma geral, substituiremos por um par de variáveis satisfazendo a relação $x_i = x_i^+ - x_i^-$ com $x_i^+, x_i^- \geq 0$. Portanto, para convertermos um PPL da forma geral para a forma canônica, iremos no máximo dobrar o número de restrições e de variáveis.

Para converter um PPL na forma canônica para a forma padrão, para cada restrição do tipo

$$a_1 x_2 + \cdots + a_n x_n \geq b$$

iremos acrescentar uma variável x_s chamada *variável de folga* tal que

$$a_1 x_2 + \cdots + a_n x_n - x_s = b$$

com a restrição $x_s \geq 0$. Portanto, ao converter um PPL da forma canônica para a forma padrão, iremos utilizar no máximo o dobro de restrições e de variáveis. □

4.2 Convexidade, Polítopos e Vértices

Antes de prosseguirmos o estudo de Problemas de Programação Linear, daremos nesta sessão um conjunto de definições úteis a respeito da geometria no $\mathbb{R}^n$.

Definição 4.3. *Dado $0 \neq a \in \mathbb{R}^n$ e $b \in \mathbb{R}$, um hiperplano no $\mathbb{R}^n$ é um conjunto da forma*

$$H = \{x | a^T x = b, x \in \mathbb{R}^n\}$$

De forma similar, um semiespaço no $\mathbb{R}^n$ é um conjunto da forma

$$S = \{x | a^T x \geq b, x \in \mathbb{R}^n\}$$

Definição 4.4. *Uma interseção finita de semiespaços é chamada de poliedro. Se um poliedro está contido numa bola $B(x_0, r)$ para algum $x_0 \in \mathbb{R}^n$ e $r \in \mathbb{R}$, então é chamado de polítopo.*

Note que um PPL tem como conjunto de soluções factíveis F um poliedro e se F é um polítopo, então o PPL sempre admite uma solução ótima. Vamos descrever brevemente algumas propriedades dos poliedros e polítopos no $\mathbb{R}^n$.

Definição 4.5. *Uma combinação convexa de vetores $x_1, \cdots, x_m \in \mathbb{R}^n$ é uma combinação linear $\lambda_1 x_1 + \cdots + \lambda_m x_m$ onde $\lambda_i \geq 0$ para $i = 1, \cdots, m$ e $\sum_{i=1}^m \lambda_i = 1$. Em particular, uma combinação convexa de dois pontos $x, y \in \mathbb{R}^n$ é um ponto da forma $p = \lambda x + (1-\lambda)y$ com $\lambda \in [0,1]$. p é um ponto situado no segmento que liga x a y.*

Dizemos que uma combinação convexa $p = \sum_{i=1}^m \lambda_i x_i$ é estrita se $p \neq x_i$ para $i = 1, \cdots, m$.

Definição 4.6. *Um conjunto $X \subset \mathbb{R}^n$ é convexo se, e somente se toda combinação convexa de quaisquer dois elementos de X é também um elemento de X.*

Uma das principais propriedades de poliedros no $\mathbb{R}^n$ será anunciada a seguir.

Teorema 4.2. *Todo poliedro P no $\mathbb{R}^n$ é convexo.*

Demonstração. Se $P = \emptyset$, então não há nada que possa contrariar o teorema. Caso contrário, tome $x, y \in P$. Existe uma matriz $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$ tal que $P = \{x | Ax \geq b\}$ pela própria definição de poliedro. Portanto, para $\lambda \in [0,1]$, $A(\lambda x + (1-\lambda)y) \geq \lambda b + (1-\lambda)b = b \Longrightarrow \lambda x + (1-\lambda)y \in P$. Não precisamos ter que $x \neq y$, uma vez que P pode ter apenas um único elemento. □

Vamos dar agora a definição de *vértice*, fundamental para a localização de pontos ótimos de um PPL. Iremos, posteriormente, relacioná-los com soluções básicas.

Definição 4.7. *Seja P um poliedro no $\mathbb{R}^n$. $v \in P$ é um vértice ou um extremo de P se, e somente se v não pode ser escrito como combinação convexa estrita de elementos de P.*

Um ponto extremo de um polítopo é necessariamente um ponto na fronteira desse mesmo polítopo, mas o inverso não é verdadeiro. Veremos mais adiante que se um PPL admite valor ótimo, então existe um vértice que o assume. Isso é natural pensar, uma vez que a direção de maior decrescimento de uma função objetivo linear é constante.

Definição 4.8. *Seja $X \subset \mathbb{R}^n$. O fecho convexo de X ou* $\text{conv}(X)$ *é definido como o menor conjunto convexo contendo X. Além disso, as afirmações a seguir são equivalentes:*

1. $\text{conv}(X)$ *é o menor (e único) conjunto convexo contendo X.*
2. $\text{conv}(X) = \{\lambda_1 x_1 + \cdots + \lambda_m x_m | x_1, \cdots, x_m \in X, \lambda_1, \cdots, \lambda_m \geq 0, \sum_{i=1}^m \lambda_m = 1, \forall m \in \mathbb{Z}^+\}$ *ou simplesmente o conjunto de todas as combinações convexas de elementos de X.*
3. $\text{conv}(X)$ *é a interseção de todos os conjuntos convexos contendo X.*

A verificação da veracidade das equivalências das afirmações acima não é difícil e será suprimida desse texto.

Lema 4.1. *As seguintes afirmações são verdadeiras:*

1. *Seja P um polítopo e $V = \{v_1, \cdots, v_m\}$ seu conjunto de vértices. Portanto, $P = \text{conv}(V)$.*
2. *Seja $V = \{v_1, \cdots, v_m\}$ e considere o polítopo $P = \text{conv}(V)$. Se V' é o conjunto de vértices de P, então $V' \subset V$.*

Demonstração. Vamos apenas demonstrar a segunda parte do lema. Note que estamos considerando que o fecho convexo de um conjunto finito de pontos é um polítopo. De fato isso acontece, como veremos na demonstração da primeira parte do lema. Se V' é o conjunto de vértices de P, então $V' \subset P$. Logo, para cada $v \in V'$, podemos escrever $v = \lambda_1 v_1 + \cdots + \lambda_m v_m$ onde $\lambda_i \geq 0$ para $i = 1, \cdots, m$ e $\sum_{i=1}^{m} \lambda_i = 1$. Pela definição de vértice, a combinação convexa não pode ser estrita, e portanto $v = v_j$ para algum $j \in \{1, \cdots, m\}$. Isso implica que $V' \subset V$. □

4.3 Soluções Básicas

Definição 4.9. *Seja uma matriz $A \in \mathbb{R}^{m \times n}$, com $m \leq n$, tal que* $\text{rank}(A) = m$[1]*. O conjunto de índices $\imath \subset \{1, \cdots, n\}$ com $|\imath| = m$ é uma base para a matriz A se, e somente se o conjunto $\{A_i | i \in \imath\}$ é LI. A_i representa o vetor formado pelos elementos da i-ésima coluna da matriz A.*

Definição 4.10. *Seja um PPL na forma padrão dado por*

$$\begin{aligned} \min \ & c^T x \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

com $A \in \mathbb{R}^{m \times n}$ e $\text{rank}(A) = m$. Seja $\mathfrak{B} = \{i_1, \cdots, i_m\} \subset \{1, \cdots, n\}$ uma base arbitrária para A, com $i_j < i_{j'} \iff j < j'$. Definimos $B \in \mathbb{R}^{m \times m}$ da seguinte forma

$$B := (A_{i_1}, \cdots, A_{i_m})$$

i.e., B é formado pelas colunas de A indexadas por $\mathfrak{B}$ em ordem crescente. Claramente B é invertível. Definimos a variável $x = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$ de tal forma que $x_i = 0$ se $i \notin \mathfrak{B}$ e $x_{i_j} = (B^{-1}b)_j$ (j-ésima componente de $B^{-1}b$) se $i_j \in \mathfrak{B}$ para algum j. Cada variável x_i para $i \in \mathfrak{B}$ é chamada de básica e cada variável x_j para $j \notin \mathfrak{B}$ é chamada de não-básica. Note que todas as variáveis não-básicas tem valor identicamente nulo, enquanto as básicas podem assumir valores nulos ou não nulos. Claramente, $Ax = b$, onde x é chamada de solução básica. Se $x \geq 0$, então x é chamada de solução básica factível ou sbf, pois pertence ao conjunto de soluções factíveis do PPL em questão.

[1] A função $\text{rank}(A)$ denota o posto de uma matriz A.

Como veremos a seguir, soluções básicas são extremamente importantes para PPLs. Podemos relacioná-las com vértices da região factível de um PPL na forma padrão: w é uma sbf da instância se, e somente se w é um vértice do conjunto factível.

Lema 4.2. *Seja um PPL na forma padrão que tem $x = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$ como solução factível e $A \in \mathbb{R}^{m \times n}$ a matriz de restrições com* $\text{rank}(A) = m$. *Se o conjunto $B = \{A_i | x_i \neq 0,\ i = 1, \cdots, n\}$ é linearmente independente, então x é uma sbf.*

Demonstração. Seja $\mathfrak{B} = \{i | x_i \neq 0,\ i = 1, \cdots, n\}$. Como B é linearmente independente, temos que $|\mathfrak{B}| = |B| \leq m$. Desta forma, dado que $\text{rank}(A) = m$, existe uma extensão $\mathfrak{B}'$ de $\mathfrak{B}$, i.e., $\mathfrak{B} \subset \mathfrak{B}'$ tal que $|\mathfrak{B}'| = m$ com $B' = \{A_i | i \in \mathfrak{B}'\}$ linearmente independente. Portanto, $\mathfrak{B}'$ é uma base para x, o caracterizando como uma sbf. □

Teorema 4.3. *Seja um PPL na forma padrão, $F \subset \mathbb{R}^n$ seu conjunto factível e $x \in \mathbb{R}^n$. Então v é um vértice de F se, e somente se v é uma sbf da instância.*

Demonstração. ($\Leftarrow$) v é uma sbf, e portanto v é a única sbf para alguma base $\mathfrak{B} \subset \{1, \cdots, n\}$. Logo, se v pode ser escrito como uma combinação convexa estrita de dois vetores $x = (x_1, \cdots, x_n)^T$ e $y = (y_1, \cdots, y_n)^T$ de F, i.e., existe $\lambda \in (0,1)$ tal que $v = \lambda x + (1-\lambda)y$ e $v \neq x, y$; temos que existe um $i \notin \mathfrak{B}$ para o qual $x_i > 0$, pois $x \geq 0$. Logo teríamos $v_i = \lambda x_i + (1-\lambda)y_i > 0$, contradizendo o fato de que v é uma sbf para a base $\mathfrak{B}$. Portanto, v é um vértice.

($\Rightarrow$) $v = (v_1, \cdots, v_n)^T$ é um vértice de F e portanto v é factível. Utilizando a contrapositiva do lema 4.2, temos que se v não é uma sbf, então o conjunto $B = \{A_i | v_i \neq 0, i = 1, \cdots, n\}$ e portanto é linearmente dependente. Desta forma, existe $\lambda = (\lambda_1, \cdots, \lambda_n)^T \in \mathbb{R}^n$ não nulo satisfazendo $\sum_{i \in \mathfrak{B}} \lambda_i A_i = 0$. Para $i \notin \mathfrak{B}$, fazemos $\lambda_i = 0$ para que tenhamos $A\lambda = 0$. Portanto, se $Av = b$, então $A(v \pm \delta\lambda) = b$. Se tomarmos δ não nulo suficientemente pequeno[2], teremos que $v \pm \delta\lambda \geq 0$. Portanto, é válida a igualdade $v = \frac{1}{2}(v + \delta\lambda) + \frac{1}{2}(v - \delta\lambda)$, contrariando o fato de que v é um vértice. Logo, v é uma sbf. □

Corolário 4.1. *Um PPL na forma padrão tem um número finito de vértices.*

Demonstração. $x \in \mathbb{R}$ é um vértice de um PPL se, e somente se x é uma sbf. Contudo, a quantidade de sbfs é limitada superiormente pela quantidade de bases da matriz de restrições $A \in \mathbb{R}^{m \times n}$. Portanto, a quantidade de vértices é de no máximo $\binom{n}{m}$. □

Agora vamos enunciar o principal resultado desta sessão.

Teorema 4.4. *Seja um PPL na forma padrão e $F \neq \emptyset$ seu poliedro factível. Então uma, e apenas uma das afirmações pode ser verdade:*

1. *O problema não é limitado, i.e., $c^T x$ não tem limite inferior com $x \in F$.*
2. *$\forall x \in F$ existe um vértice $v \in F$ tal que $c^T v \leq c^T x$.*

[2] Escolha $\delta = \min_{\lambda_i \neq 0, i \in \mathfrak{B}} \frac{v_i}{|\lambda_i|}$. Claramente $\delta > 0$.

Demonstração. A segunda afirmação claramente implica na negação da primeira. Portanto suponha que a primeira afirmação é falsa. Pelo lema 4.1, se $\{v_1, \cdots, v_n\}$ é o conjunto de vértices de F, então qualquer ponto $v \in F$ pode ser escrito da seguinte maneira

$$v = \sum_{i=1}^{n} \lambda_i v_i$$

com $\sum_{i=1}^{n} \lambda_i = 1$ e $\lambda_i \geq 0$ para $i = 1, \cdots, n$. Seja v_j o vértice que minimiza $\{c^T v_1, \cdots, c^T v_n\}$. Desta forma, $c^T x = \sum_{i=1}^{n} \lambda_i c^T v_i \geq \sum_{i=1}^{n} \lambda_i c^T v_j = c^T v$, o que mostra que $c^T v_j \leq c^T v$ para v arbitrário. □

Corolário 4.2. *Seja um PPL na forma padrão e $F \neq \emptyset$ seu poliedro factível. Se existe* $\inf\{c^T x; x \in F\}$*, então existe um vértice $v \in F$ que minimiza o PPL. Esse vértice é chamado de vértice ótimo.*

4.4 Método da Elipsóide

O primeiro algoritmo para resolver PPLs foi desenvolvido em 1947, por George Dantzig. Se aproveitando do fato enunciado pelo corolário 4.2, o algoritmo do *Simplex* nada mais é do que uma busca ordenada por sbfs de um PPL na forma padrão. Pelo corolário 4.1, essa busca sempre atinge um fim. O algoritmo pode ser descrito da seguinte maneira: dada uma sbf x_0 com base $\mathfrak{B}_0$, o algoritmo procura por uma sbf x_1 adjacente[3] com base $\mathfrak{B}_1$ tal que $c^T x_1 \leq c^T x_0$. Quando todas as sbfs adjacentes tem custo superior a sbf atual, então o algoritmo encontrou o ótimo e para. A maneira de escolher a sbf adjacente com custo inferior à sbf atual pode variar por implementação.

O algoritmo do Simplex sempre retorna uma solução ótima quando o problema é limitado ou reporta se esta característica está presente no PPL. Em alguns casos, uma iteração do algoritmo pode ter ganho nulo, o que damos o nome de *degenerescência*: duas bases diferentes podem representar uma mesma sbf. Em situações raras, os passos nulos causados pela degenerescência da sbf podem causar ciclos se algumas regras de pivoteamento não forem tomadas.

Na prática, o algoritmo do Simplex é bastante eficiente, não tomando mais que $10m$ pivoteamentos em problemas práticos. Entretanto, Klee e Minty construiram uma família de PPLs para os quais o algoritmo do Simplex, utilizando uma regra particular de mudança de base, necessitava de um número exponencial de iterações em relação ao tamanho da entrada. Pouco tempo depois, mostrou-se que para todas as regras conhecidas de mudança de base, existiam instâncias do problema que forçavam o Simplex a usar um número exponencial de pivoteamentos. Por algum tempo, não se soube se de fato a classe de problemas de programação linear poderia ser resolvida em tempo polinomial sobre o tamanho da entrada.

Em 1979, Leonid Khachiyan desenvolveu o *Método da Elipsóide* para resolver ou reportar a infactibilidade de um sistema de desigualdades estritas, i.e., dada uma matriz $A \in \mathbb{Q}^{m \times n}$ e $b \in \mathbb{Q}^m$, achar $x \in \mathbb{Q}^n$ tal que $Ax < b$ ou reportar que não existe x com essa propriedade. Não é difícil de demonstrar que a classe de problemas de programação linear é polinomialmente redutível ao problema de sistemas de desigualdades estritas e sua recíproca. Embora inicialmente

[3] Uma sbf é adjacente a outra quando suas respectivas bases se diferenciam de no máximo uma variável.

o Método da Elipsóide não resolvesse PPLs diretamente, o algoritmo executava em tempo polinomial, implicando que a classe de problemas de otimização linear poderia também ser resolvida em tempo polinomial. Embora tenha sido um resultado de grande importância para a área de pesquisa operacional, o Método da Elipsóide apresentava péssimo desempenho na prática, além de grande instabilidade numérica. O método para resolver PPLs que estudaremos nesta sessão é uma versão modificada do Método da Elipsóide, desenvolvido por Grötschel, Lovász e Schrijver em 1980. Chamaremos esse algoritmo de *algoritmo (ou método) GLS*.

4.4.1 Otimização em Conjuntos Convexos

O algoritmo GLS considera não apenas poliedros como sua região factível, como seria num PPL, mas espaços convexos com certas propriedades.

Definição 4.11. *Seja $X \subset \mathbb{R}^n$ um conjunto convexo e compacto. Definimos dois problemas algorítmicos em relação a X.*

1. ***Problema de otimização no sentido forte**: dado $c \in \mathbb{R}^n$, determinar $x \in X$ que maximiza $c^T x$ em X.*
2. ***Problema de separação no sentido forte**: dado $y \in \mathbb{R}^n$, determinar se $y \in X$ ou encontrar um hiperplano que separa y de X, mais precisamente, achar $d \in \mathbb{R}^n$ para o qual $d^T sy > \sup\{d^T x | x \in X\}$.*

Exemplo 4.1. *Seja X o conjunto de soluções de um sistema de inequações lineares:*

$$X = \{x | Ax \leq b\}$$

onde $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$. O problema de separação no sentido forte pode ser facilmente resolvido: dado um x, verificar se $Ax \leq b$. Caso exista uma linha A^i de A para o qual $A^i x > b_i$, então retorne $(A^i)^T$ como o vetor indicando o hiperplano que separa x de X. O problema de otimização no sentido forte é exatamente o mesmo problema de resolver PPLs.

Exemplo 4.2. *Tome $V = \{v_1, \cdots, v_m\} \subset \mathbb{R}^n$ e defina $X := \text{conv}(V)$. O problema de otimização no sentido forte pode ser resolvido valorando a função objetivo em cada um dos pontos de V e selecionando o máximo, uma vez que todos os vértices de X são também elementos de V. Para resolver o problema de separação no sentido forte para um dado $x \in \mathbb{R}^n$, podemos verificar se $x \in X$, pois X pode ser descrito como um conjunto de inequações lineares. Caso $x \notin X$, devemos achar $d \in \mathbb{R}^n$ para o qual $d^T x > d^T v_i$ para $i = 1, \cdots, m$. Esse problema é equivalente (inclusive em complexidade) a resolver problemas de programação linear, como foi citado anteriormente.*

Note que no exemplo acima, X é um polítopo e portanto pode ser descrito como um conjunto de soluções de inequações lineares. Contudo, esse número de inequações pode ser muito grande em relação a m e n, ou até mesmo exponencial. Portanto checar se um ponto x pertence a X pode levar muito tempo. Isso mostra que o problema de otimização e separação no sentido forte pode depender de como um conjunto é representado, e não apenas do conjunto em si.

Dado $X \subset \mathbb{R}^n$ convexo e compacto, não estamos fazendo nenhuma afirmação aritmética sobre seus elementos. Isso quer dizer que um elemento x de X que maximiza $c^T x$ pode ter coordenadas irracionais, o que nos impossibilida de representar x diretamente utilizando o nosso modelo de computação. Desta forma, vamos definir versões mais fradas do problema de otimização e do problema de separação.

Definição 4.12. *Seja $X \subset \mathbb{R}^n$ um conjunto convexo e compacto. Definimos dois problemas algorítmicos em relação a X.*

1. ***Problema de otimização no sentido fraco**: dados $c \in \mathbb{R}^n$ e $\varepsilon > 0$, determinar $y \in \mathbb{R}^n$ tal que $d(y,X) \leq \varepsilon$ que quase maximiza $c^T x$ em X com respeito a ε, i.e., para todo $x \in X$ temos que $c^T x \leq c^T y + \varepsilon$*

2. ***Problema de separação no sentido fraco**: dados $y \in \mathbb{R}^n$ e $\varepsilon > 0$, determinar se $d(y,X) \leq \varepsilon$ ou encontrar $d \in \mathbb{R}^n$ com $\|d\| \geq 1$ para o qual $d^T y + \varepsilon \geq \sup\{d^T x | x \in X\}$.*

Os problemas definidos acima são mais complicados contudo mais adequados e corretos quando estamos utilizando um modelo de computação com representação finita. Sempre estaremos assumindo que serão dados x_0 e $0 < r \geq R$ tais que

$$B(x_0, r) \subset X \subset B(x_0, R)$$

A primeira inclusão indica que X é um conjunto com dimensão total em $\mathbb{R}^n$, i.e., é capaz de conter objetos de dimensão n. A segunda é pelo simples fato de X ser limitado e seu limite ser conhecido. É natural pensar que esse limite é sempre conhecido na prática (e na teoria).

Definição 4.13. *Um corpo convexo é uma quintupla (X, n, x_0, r, R) onde*

- $n \in \mathbb{Z}^+$.
- $X \subset \mathbb{R}^n$ *é um conjunto compacto e convexo.*
- $x_0 \in X$; $r, R \in \mathbb{R}$.

E a seguinte relação é satisfeita:

$$B(x_0, r) \subset X \subset B(x_0, R)$$

Ao tratarmos de uma classe de corpos convexos, assumiremos que cada elemento da classe terá uma codificação. Uma entrada para o problema de otimização sobre um corpo convexo K é a codificação de K, um vetor $c \in \mathbb{R}^n$ e um número $\varepsilon > 0$. Naturalmente, o tamanho da entrada é limitado inferiormente por $n + |\log r| + |\log R| + |\log \varepsilon|$. Um fato importante é que o algoritmo deve ser polinomial em relação a $|\log \varepsilon|$, uma vez que se tomarmos uma sequência $\varepsilon = 2^{-i}$ para $i \in \mathbb{Z}^+$, nós teremos uma sequência que converge exponencialmente rápido para o ótimo. Se utilizarmos o método GLS para resolver instâncias do PPL, consiguiremos achar, em tempo polinomial, um ótimo exato para o problema.

4.4.2 Transformações Afim e Elipsóides

Definição 4.14. *Uma transformação afim é uma função $T : \mathbb{R}^n \to \mathbb{R}^n$ para o qual existem $A \in \mathbb{R}^{n\times n}$ invertível e $b \in \mathbb{R}^n$ tal que $T(x) = Ax + b$. Uma transformação afim possui inversa, que pode ser descrita por $T^{-1}(x) = A^{-1}x + A^{-1}b$, que também é uma transformação afim.*

Definição 4.15. *Seja T uma transformação afim e $B[0,1]$ a esfera unitária. O conjunto $E = T(B[0,1])$ é uma elipsóide. De modo equivalente, se $T(x) = Ax + b$, então $E = \{Ax + b | x^T x \leq 1\}$.*

Lema 4.3. *Se E é uma elipsóide, então existem $a \in \mathbb{R}^n$ e $B \in \mathbb{R}^{n\times n}$ definida positiva para os quais $E = \{x | (x-a)^T B^{-1}(x-a) \leq 1\}$.*

Demonstração. Se E é uma elipsóde, então E é a imagem de uma função afim $T(x) = Ax + b$ aplicada a esfera unitária. Portanto, fazendo $y = Ax + b$:

$$\begin{aligned} E &= \{Ax + b | x^T x \leq 1\} \\ &= \{y | (A^{-1}(y-b))^T A^{-1}(y-b) \leq 1\} \\ &= \{y | (y-b)^T (A^{-1})^T A^{-1}(y-b) \leq 1\} \\ &= \{y | (y-a)^T B^{-1}(y-b) \leq 1\} \end{aligned}$$

onde nesta última igualdade definimos $a := b$ e $B := AA^T$. Claramente B é definida positiva, portanto o lema é verdadeiro. □

De modo oposto, se B é definida positiva, então existe A invertível para o qual $B = AA^T$. Portanto, todo conjunto da forma $\{x | (x-a)^T B^{-1}(x-a) \leq 1\}$ para B definida positiva, é uma elipsóide. O ponto a é chamado de *centro da elipsóide*.

Lema 4.4. *Seja $T(x) = Ax + b$ uma transformação afim. Seja $P \subset \mathbb{R}^n$ um conjunto limitado. Se P tem volume v, então o volume de $T(P)$ será $|\det A| v$.*

Demonstração. O volume de um conjunto $X \subset \mathbb{R}^n$ é definido por

$$v(X) = \int_X dx_1 \cdots dx_n$$

E portanto, o volume de $T(X)$ é

$$v(T(X)) = \int_{T(X)} dx_1 \cdots dx_n = \int_X dx'_1 \cdots dx'_n$$

onde $(x'_1, \cdots, x'_n)^T = Ax + b$, i.e., $x'_i = A^i x + b_i$ para $i = 1, \cdots, n$. Pelo teorema da Jacobiana, temos que $dx'_1 \cdots dx'_n = \left|\det \frac{\partial T(x_1,\cdots,x_n)}{\partial(x_1,\cdots,x_n)}\right| dx_1 \cdots dx_n$. Definimos

$$\frac{\partial T(x_1, \cdots, x_n)}{\partial(x_1, \cdots, x_n)} := \begin{bmatrix} \frac{\partial T_1}{\partial x_1} & \cdots & \frac{\partial T_1}{x_n} \\ \vdots & \ddots & \cdots \\ \frac{\partial T_n}{\partial x_1} & \cdots & \frac{\partial T_n}{x_n} \end{bmatrix}$$

onde $T_i(x) = A^i x + b_i$ para $i = 1, \cdots, n$. Se $A = (a_{ij})_{n\times n}$ então $\frac{\partial T_i}{\partial x_j} = a_{ij}$, o que nos garante que $\frac{\partial T(x_1,\cdots,x_n)}{\partial(x_1,\cdots,x_n)} = A$. Portanto

$$v(T(X)) = \int_X dx'_1 \cdots dx'_n = \int_X |\det A| dx_1 \cdots dx_n = |\det A| v(X)$$

□

4.4.3 O Método

[4] Antes, vamos dar uma breve descrição geométrica intuitiva sobre o método GLS. Seja (X, n, x_0, r, R) o corpo convexo e $c^T x$ a função objetivo que desejamos maximizar em X, contido na elipsóide $B[x_0, R] = E_0$. A k-ésima iteração do algoritmo, gerará uma elipsóide E_k que contém o conjunto X_k de pontos x em X para os quais $c^T x$ são maiores do que ou igual aos melhores valores encontrados até então. Nessa etapa verificamos se x_k, definido como o centro de E_k, pertence a X. Caso negativo, tomamos um hiperplano que separa x_k de X. O hiperplano separa o espaço em dois semiespaços e tomamos aquele que inclui X e traçamos uma nova elipsóide E_{k+1} com volume menor que E_k. Se $x_k \in X$, então também dividiremos o espaço, mas desta vez o hiperplano será ortogonal ao vetor de custo c. A razão dos volumes das elipsóides geradas por iterações seguidas é menor que uma constante estritamente menor que 1, o que garante que os volumes das elipsóides tenderão para 0 exponencialmente, e portanto, o ótimo será encontrado de maneira exponencialmente rápida.

Estaremos sempre considerando que existirá uma rotina ao qual chamaremos de *SEP* que dado um vetor $y \in \mathbb{R}^n$ e $\delta > 0$, resolve em tempo polinomial sobre a entrada o problema de separação no sentido fraco sobre o conjunto convexo X.

[4] Os conceitos básicos de álgebra linear e normas para esta sessão são apresentados no **Apêndice A**

Algoritmo 1: Método GLS

Entrada: $(n, \varepsilon, x_0, r, R, c, SEP)$
Saída: vetor $x \in \mathbb{R}^n$
início
  $N := 4n^2 \left\lceil \ln \frac{2R^2\|c\|}{r\varepsilon} \right\rceil$;
  $\delta := \frac{R^2 4^{-N}}{300n}$;
  $p := 5N$;
  $A_0 := R^2 I_n$;
  $k := 0$;
  repita
    Execute $d := SEP(x_k, \delta)$;
    se *SEP conclui que* x_k *é factível* **então**
      $a := c$;
    fim
    senão
      $a := -d$;
    fim
    $b_k := \frac{A_k a}{\sqrt{a^T A_k a}}$;
    $x_{k+1} := r_p\left(x_k + \frac{1}{n+1} b_k\right)$;
    $A_{k+1} := r_p\left(\frac{2n^2+3}{2n^2}\left(A_k - \frac{2}{n+1} b_k b_k^T\right)\right)$;
    $k := k+1$;
  até $k > N$;
fim

Onde a a função r_p denota o arredondamento utilizando p dígitos binários de precisão. Chamaremos os índices k de factíveis se x_k é um ponto factível. A sequência x_k para k factível, dará uma boa aproximação para a solução ótima do problema. A cada iteração do laço, estaremos considerando a elipsóide

$$E_k = \{x \in \mathbb{R}^n | (x - x_k)^T A_k^{-1} (x - x_k) \leq 1\}$$

Ao final das N iterações do método, o algoritmo retornará y tal que $c^T y \geq \sup\{c^T x | x \in X\} - \varepsilon$ se X admitir alguma solução factível. Para mostrar isso, vamos enunciar alguns lemas a seguir.

Lema 4.5. *As matrizes A_i, para $i = 0, 1, 2, \cdots$ são definidas positiva. Além disso, valem as seguintes desigualdades*

$$\|x_k\| \leq \|x_0\| + R \cdot 2^k \tag{4.5}$$
$$\|A_k\| \leq R^2 \cdot 4^k \tag{4.6}$$
$$\|A_k^{-1}\| \leq R^{-2} \cdot 4^k \tag{4.7}$$

Demonstração. Vamos provar por indução sobre k. Para $k = 0$, todas as afirmações são óbvias.

Antes note que

$$\|b_k\| = \frac{\|A_k a\|}{\sqrt{a^T A_k a}} = \sqrt{\frac{a^T A_k^2 a}{a^T A_k a}}$$

como $\|a^T\|\|A_k\|^2\|a\| \geq \|A_k\| a^T A_k a \geq a^T A_k^2 a$, então $\|A_k\| \geq \frac{a^T A_k^2 a}{a^T A_k a}$ o que implica que $\|b_k\| \leq \sqrt{\|A_k\|}$. Portanto, temos que

$$\frac{2n^2+3}{2n^2}\|A_k - \frac{2}{n+1} b_k b_k^T\| \leq \frac{2n^2+3}{2n^2} 2\|A_k\| \leq (2+\frac{3}{n^2})R^2 \cdot 4^k$$

Desta forma,

$$\|A_{k+1}\| \leq (2+\frac{3}{n^2})R^2 \cdot 4^k + n \cdot 2^{-p} \leq R^2 \cdot 4^{k+1}$$

Também temos que

$$\|x_k + \frac{1}{n+1} b_k\| \leq \|x_k\| + \frac{1}{n+1}\|b_k\| \leq \|x_0\| + R \cdot 2^k + \frac{1}{n+1} R \cdot 2^k = \|x_0\| + \frac{n+2}{n+1} R \cdot 2^k$$

logo

$$\|x_{k+1}\| \leq \|x_0\| + \frac{n+2}{n+1} R \cdot 2^k + \sqrt{n} \cdot 2^{-p} \leq \|x_0\| + R \cdot 2^{k+1}$$

Defina $\tilde{A}_k := \frac{2n^2+3}{2n^2}\left(A_k - \frac{2}{n+1} b_k b_k^T\right)$, portanto $A_{k+1} = r_p(\tilde{A}_k)$. É fácil de verificar que

$$\tilde{A}_k^{-1} = \frac{2n^2}{2n^2+3}\left(A_k^{-1} + \frac{2}{n-1}\frac{a a^T}{a^T A_k a}\right)$$

Como A_k^{-1} e aa^T são definidas positiva[5], então $\tilde{A}_k^{-1}$ também é, e portanto $\tilde{A}_k$ é definida positiva.

$$\|\tilde{A}_k^{-1}\| \leq \frac{2n^2}{2n^2+3}\left(\|A_k^{-1}\| + \frac{2}{n-1}\frac{\|a\|^2}{a^T A_k a}\right)$$

Perceba que $\|a^T\|\|A_k\|\|A_k^{-1}\|\|a\| \geq \|A_k^{-1}\| a^T A_k a \geq a^T A_k A_k^{-1} a = \|a\|^2 \Longrightarrow \|A_k^{-1}\| \geq \frac{\|a\|^2}{a^T A_k a}$. Desta forma, temos

$$\|\tilde{A}_k^{-1}\| \leq \frac{2n^2}{2n^2+3}\left(\|A_k^{-1}\| + \frac{2}{n-1}\|A_k^{-1}\|\right) \leq \frac{n+1}{n-1}\|A_k^{-1}\|$$

Para mostrar que A_{k+1}^{-1} é definida positiva, temos que mostrar que qualquer de seus autovalores é positivo. Seja λ um autovalor de A_{k+1} e v o seu autovetor correspondente e normalizado, i.e., $\|v\| = 1$. Portanto,

$$\lambda = v^T A_{k+1} v = v^T \tilde{A}_k v + v^T (A_{k+1} - A_k) v$$

Veja que $v^T \tilde{A}_k v \|\tilde{A}_k^{-1}\| \geq v^T \tilde{A}_k \tilde{A}_k^{-1} v = \|v\|^2 = 1$. Desta forma

$$\lambda \geq \|\tilde{A_k}^{-1}\|^{-1} - \|A_{k+1} - \tilde{A}_k\| \geq \frac{n-1}{n+1}\|A_k^{-1}\|^{-1} - n \cdot 2^{-p} \geq \frac{n-1}{n+1} R^2 \cdot 4^{-k} - n \cdot 2^{-p} \geq R^2 \cdot 4^{-(k+1)}$$

[5]Se $a = (a_1, \cdots, a_n)^T$ e $x = (x_1, \cdots, x_n)^T$, então $x^T a a^T x = (a_1 x_1 + \cdots + a_n x_n)^2 > 0$ com $x, a \neq 0$.

Isto prova que A_{k+1} é definida positiva, e portanto

$$\|A_{k+1}^{-1}\| \leq R^{-2} \cdot 4^{k+1}$$

uma vez que todos os autovalores da sua inversa são maiores que $R^2 \cdot 4^{-(k+1)}$. □

Lema 4.6. *Defina* $\xi_k = \max\{c^T x_i | 0 \leq i < k,\ i \text{ índice factível}\}$ *e* $X_k = \{x | c^T x \geq \xi_k, x \in X\}$. *Portanto,* $X_k \subset E_k$ *para* $k = 0, \cdots, N$. *Em outras palavras, os pontos de X que são melhores que o melhor ponto achado até a k-ésima iteração, estão dentro da k-ésima elipsóide. Desta forma, na k-ésima iteração só precisamos considerar os pontos na k-ésima elipsóide.*

Demonstração. Vamos demonstrar o lema por indução sobre k. Para $k = 0$ a afirmação é óbvia, pois $X \subset E_0$. Naturalmente, $X_{k+1} \subset X_k$. Logo, assuma que $x \in X_{k+1} \subset X_k \subset E_k$. Veja também que

$$a_k^T x \geq a_k^T x_k - \delta \tag{4.8}$$

onde a_k é o vetor a definido na k-ésima iteração. Decomponha x da seguinte forma

$$x = x_k + y + t b_k \tag{4.9}$$

onde y está contido no plano perpendicular a a_k, em outras palavras, $a_k^T y = 0$. Esse plano tem dimensão $n-1$ mas não contém o vetor b_k, uma vez que $a_k^T A_k a_k > 0$, pois A_k é definida positiva. Assim fica mostrado que essa decomposição sempre existe. Como $x \in E_k$, então

$$(y + t b_k)^T A_k^{-1} (y + t b_k) = y^T A_k^{-1} y + t^2 b_k^T A_k^{-1} b_k + t(y^T A_k^{-1} b_k + b_k^T A_k^{-1} y) \leq 1$$

Veja que $y^T A_k^{-1} b_k = b_k^T A_k^{-1} y = 0$ pela definição de y. Logo,

$$y^T A_k^{-1} y + t^2 b_k^T A_k^{-1} b_k = y^T A_k^{-1} y + t^2 \frac{a_k^T A_k A_k^{-1} A_k a_k}{a_k^T A_k a_k} = y^T A_k^{-1} y + t^2 \leq 1$$

Como A_k^{-1} é definida positiva, então $t^2 \leq 1 \Longrightarrow -1 \leq t \leq 1$. Substituindo 4.9 em 4.8, temos

$$t a_k b_k = t\sqrt{a_k^T A_k a_k} \geq -\delta$$

Vamos afirmar, sem demonstração, que

$$(x - x_{k+1})^T A_{k+1}^{-1} (x - x_{k+1}) \leq (x - \tilde{x}_k)^T \tilde{A}_k^{-1} (x - \tilde{x}_k) + \frac{1}{12n^2}$$

onde $\tilde{x}_k = x_k + \frac{1}{n+1} b_k$, da mesma forma como definimos $\tilde{A}_k$, temos que $x_{k+1} = r_p(\tilde{x}_k)$. Este fato pode ser demonstrado utilizando métodos similares aos do lema 4.5. Agora, vejamos

$$(x - \tilde{x}_k)^T \tilde{A}_k^{-1} (x - \tilde{x}_k) =$$

$$\frac{2n^2}{2n^2+3} \left[\left(t - \frac{1}{1+n}\right) b_k + y\right]^T \left(A_k^{-1} + \frac{2}{n-1} \frac{a a^T}{a^T A_k a}\right) \frac{2n^2}{2n^2+3} \left[\left(t - \frac{1}{1+n}\right) b_k + y\right] =$$

$$\frac{2n^2}{2n^2+3} \left[\left(t - \frac{1}{n+1}\right)^2 + y^T A_k^{-1} y + \frac{2}{n-1} \left(t - \frac{1}{n+1}\right)^2\right] \leq$$

$$\frac{2n^2}{2n^2+3} \left[\frac{n+1}{n-1} \left(t - \frac{1}{n+1}\right)^2 + 1 - t^2\right] = \frac{2n^2}{2n^2+3} \left[\frac{n+1}{n-1} \left(t^2 + \frac{1}{(n+1)^2} - 2t \frac{1}{n+1}\right) + 1 - t^2\right] =$$

$$\frac{2n^2}{2n^2+3} \left[\frac{n^2}{n^2-1} + \frac{1}{n-1}(2t^2 - 2t)\right] = \frac{2n^2}{2n^2+3} \left[\frac{n^2}{n^2-1} - \frac{2t(1-t)}{n-1}\right] \leq \frac{2n^4}{(2n^2+3)(n^2-1)} + \frac{4\delta}{(n-1)\sqrt{a_k^T A_k a_k}}$$

$$\leq \frac{2n^4}{2n^4+n^2-3} + \frac{4\delta}{n-1} \|A_k^{-1}\| \leq \frac{2n^4}{2n^4+n^2-3} + \frac{4\delta R^{-2} 4^N}{n-1} \leq 1 - \frac{1}{12n^2}$$

Portanto $(x-x_{k+1})^T A_{k+2}^{-1}(x-x_{k+1}) \leq 1$, e $x \in E_{k+1}$. A última desigualdade pode ser verificada através do crescimento de uma função racional. □

Lema 4.7. *Seja v a função que denota o volume de um corpo. Então*

$$\frac{v(E_{k+1})}{v(E_k)} < e^{\frac{-1}{4n}}$$

Demonstração. Demonstração omitada. □

Agora enunciaremos o principal teorema desta sessão.

Teorema 4.5. $\xi_N \geq \sup\{c^T x | x \in X\} - \varepsilon$

Demonstração. Note que

$$v(X_N) \leq v(E_N) \leq e^{\frac{-N}{4n}} v(E_0) \leq e^{\frac{-N}{4n}} R^n v(B_n(0,1))$$

onde $B_n(a,r)$ é a bola n-dimensional com centro em a e raio r. Seja j o índice tal que $c^T x = \xi_N$ e $y \in X$ tal que $c^T y = \sup\{c^T x | x \in X\}$ (tal y sempre existe porque X é compacto). Considere a bola $(n-1)$-dimensional definida por $B_{n-1} := B_n(x_0, r) \cap \{x | c^T(x-x_0) = 0\}$ e o cone definido por $C = \text{conv}(\{y\} \cup B_{n-1})$. Uma vez que $\{y\} \cup B_{n-1} \subset X$, então $C \subset X$, pois X é convexo. Esse cone tem como área da base $v(B_{n-1}(0,1))r^{n-1}$ pelo fato de que se duas figuras n-dimensionais são semelhantes por razão k, então seus volumes tem razão k^n. A altura desse cone é dada por $\frac{c^T(y-x_0)}{\|c\|} = \frac{\|c\|\|y-x_0\|\cos\angle(c,y-x_0)}{\|c\|} = \|y-x_0\|\cos\angle(c,y-x_0)$. Portanto, seu volume é dado por $\frac{v(B_{n-1}(0,1))r^{n-1}c^T(y-x)}{n\|c\|}$. Defina o cone $C' := C \cap \{x | c^T(x-x_j) \geq 0\}$, cujo volume é dado por $v(C)\left(\frac{c^T(y-x_j)}{c^T(y-x_0)}\right)^n$, pois C' é semelhante a C. Pela definição de X_i, temos que $C' \subset X_N$, e desta forma,

$$\frac{v(B_{n-1}(0,1))r^{n-1}c^T(y-x)}{n\|c\|}\left(\frac{c^T(y-x_j)}{c^T(y-x_0)}\right)^n \leq v(X_N) \leq e^{\frac{-N}{4n}} R^n v(B_n(0,1))$$

Realizando algumas manipulações, obtemos que

$$c^T(y-x_j) \leq e^{\frac{-N}{4n^2}} R\left(\frac{c^T(y-x_0)}{r}\right)^{\frac{n-1}{n}}\left(\frac{nv(B_n(0,1))}{v(B_{n-1}(0,1))}\right)^{\frac{1}{n}}\|c\|^{\frac{1}{n}}$$

Sabemos que $c^T(y-x_0) \leq \|c\|\|y-x_0\| \leq R\|c\|$ e que $\frac{nv(B_n(0,1))}{v(B_{n-1}(0,1))} = n\sqrt{\pi}\frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n+3}{2})} \leq 2^n$. Desta forma, temos que

$$c^T(y-x_j) \leq 2e^{\frac{-N}{4n^2}} R\left(\frac{R\|c\|}{r}\right)^{\frac{n-1}{n}}\|c\|^{\frac{1}{n}} \leq 2e^{\frac{-N}{4n^2}}\frac{R^2}{r}\|c\| \leq \varepsilon$$

□

4.5 Dualidade

Tome o seguinte PPL $P = \max\{c^T x; Ax \leq b, x \geq 0, x \in \mathbb{R}^n\}$. O PPL *dual* de P é o PPL definido por $D = \min\{b^T y; A^T y \geq c, y \geq 0, y \in \mathbb{R}^m\}$. Note que o PPL dual do dual do primal é exatamente o PPL primal original. Iremos provar a seguir que qualquer solução factível do dual é um limite superior da solução ótima do primal, enquanto qualquer solução factível do primal é um limite inferior para a solução ótima do primal.

Teorema 4.6 (Teorema Fraco de Dualidade)**.** *Tome os PPLs P e D como descritos acima. Se x é uma solução factível de P e y uma solução factível de D, portanto $c^T x \leq b^T y$.*

Demonstração.

$$c^T x = x^T c \leq x^T (A^T y) = (Ax)^T y \leq b^T y$$

As desigualdades seguem de que x e y são soluções factíveis. □

Do teorema, obtemos que se P é ilimitado, D é infactível e vice-versa. Também existe a possibilidade de P e D serem infactíveis. Tome A como uma matriz nula, b estritamente negativo e c estritamente positivo e obteremos que o primal e dual são ambos infactíveis. O teorema a seguir traz um belo resultado para o caso de P e D serem ambos factíveis.

Teorema 4.7 (Teorema Forte de Dualidade)**.** *Se P e D são um par primal-dual de PPLs como descritos acima e factíveis, então ambos possuem soluções ótimas x, y correspondentes a P e D respectivamente tais que $c^T x = b^T y$.*

Este teorema não será demonstrado neste trabalho. No entanto, este resultado é de extrema importância para o desenvolvimento de um algoritmo que resolve o Problema do Bin Packing Fracionário, como será visto no capítulo a seguir.

CAPÍTULO 5

Os Algoritmos

Este capítulo tratará basicamente dos algoritmos de aproximação apresentados em [KK82] para resolver instâncias arbitrárias do PBP, com respeito às suas complexidades, taxas de aproximação e a subrotina que resolve uma instância relaxada do PBP em programação linear, ao qual chamaremos de *Problema do Bin Packing Fracionário* ou simplesmente PBPF. Os resultados obtidos já foram apresentados no capítulo 2, onde definimos $T(m,n)$ como uma função limitada polinomialmente por m e n. Utilizando o algoritmo GLS incorporado à subrotina que resolve PBPF, temos que o melhor limite que podemos alcançar para $T(m,n)$ é

$$T(m,n) = m^8 \log m \log^2 n + m^4 n \log m \log n$$

5.1 Bin Packing Fracionário

Recordando, uma instância arbitrária do PBP pode ser definida por um par $I = (\imath, (s_i)_{i\in\imath})$ onde $\imath = \{1,\cdots,n\}$ e $(s_i)_{i\in\imath} \subset [0,1) \cap \mathbb{Q}$. $n(I)$ denota a cardinalidade de $\imath$ e $m(I)$ a cardinalidade de $\{s_i | i \in \imath\}$, pois podemos ter itens com tamanhos iguais. Por simplicidade, omitiremos o termo I. Dada uma sequência $(s_i)_{i\in\imath}$ podemos construir $(s'_j)_{j\in\imath'}$ onde $\imath' = \{1,\cdots,m\}$ e os elementos de $(s'_j)_{j\in\imath'}$ são exatamente os elementos de $(s_i)_{i\in\imath}$. Em outras palavras, estamos retirando os elementos repetidos de $(s_i)_{i\in\imath}$. Seja b_i a multiplicidade de s'_i em $(s_i)_{i\in\imath}$, i.e., b_i é a quantidade de peças de tamanho s_i numa instância I.

Definição 5.1. *Uma configuração de uma instância $I = (\imath, (s_i)_{i\in\imath})$ é um vetor $c = (c_1,\cdots,c_m) \in (\mathbb{Z}^+ \cup \{0\})^m$ com a restrição de que $\sum_{i=1}^m c_i s'_i \leq 1$. Em outras palavras, uma configuração representa uma maneira de empacotar um único bin.*

Agora considere o problema de otimização linear: dada uma instância $I = (\imath, (s_i)_{i\in\imath})$, e $\{c_1,\cdots,c_r\}$ o conjunto de todas suas configurações. Defina

$$A := (c_1,\cdots,c_r)$$

e

$$b = (b_1,\cdots,b_m)^T$$

O PPL associado à instância I é definido como

$$\begin{aligned} \min\ & \mathbf{1}^T x \\ & Ax \geq b \\ x \geq 0,\ & x \in \mathbb{R}^r \end{aligned}$$

onde $\mathbf{1}^T = (\overbrace{1,\cdots,1}^{m \text{ vezes } 1})$. Note que neste caso estamos permitindo que uma configuração apareça uma quantidade fracionária de vezes. Esse PPL é chamado de PBPF relacionada à instância I.

Exemplo 5.1. *Suponha que $m = 2$, $s'_1 = 0.3$, $s'_2 = 0.4$, $b_1 = 31$ e $b_2 = 7$. A matriz A é computada como $A = \begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 2 & 3 \\ 1 & 2 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$ e $b = (31,7)^T$. Uma sbf ótima para o problema é $x = (0,0,0,7,0,0,\frac{17}{3})^T$. Como todo PPL limitado possui sbf ótima, então uma solução ótima para PBPF só precisará utilizar m configurações e descartar as demais.*

Seja $\text{lin}(I)$ a solução ótima do PBPF associado à instância I. Lembre-se de que $\text{size}(I)$ denota a soma dos tamanhos das peças e $\text{opt}(I)$ o número ótimo de *bins* para empacotar uma instância I. Seja x_{opt} um empacotamento ótimo de uma instância I. Claramente x_{opt} é uma solução factível do PBPF relacionado à I, e segue que $\text{lin}(I) \leq \text{opt}(I)$.

Lema 5.1. *Seja I uma instância arbitrária do PBP. Portanto,* $\text{opt}(I) \leq 2 \cdot \text{size}(I) + 1$.

Demonstração. Pelo conjunto de inequações dado por (3.8), temos que $2(\text{size}(I)+1) - 1 \geq 2\lceil \text{size}(I) \rceil - 1 \geq A(I) \geq \text{opt}(I)$. O resultado segue imediatamente. □

Lema 5.2. *Seja I uma instância arbitrária do PBP. Portanto,*

$$\text{size}(I) \leq \text{lin}(I) \leq \text{opt}(I) \leq \text{lin}(I) + \frac{m(I)+1}{2}$$

Demonstração. ($\text{size}(I) \leq \text{lin}(I)$) Defina $s := (s'_1, \cdots, s'_m)^T$ e x a solução ótima do PBPF associado à I. Logo,

$$\mathbf{1}^T x \geq (s^T A)x = s^T(Ax) \geq s^T b = \text{size}(I)$$

($\text{lin}(I) \leq \text{opt}(I)$) Já discutido anteriormente.

($\text{opt}(I) \leq \text{lin}(I) + \frac{m(I)+1}{2}$) Como citado anteriormente, ao obtermos uma sbf ótima para o PBPF associado à instância I, utilizaremos no máximo $m(I)$ configurações de *bins*. Cada componente não nula de x pode ser divida entre uma *parte principal* e uma *parte residual*. Seja x_j uma variável básica. A parte principal de x_j é definida por $\lfloor x_j \rfloor$ e sua parte residual por $x_j - \lfloor x_j \rfloor$. Definimos o vetor $\lfloor x \rfloor$ como o vetor formado pelas partes principais das variáveis em x. Empacotando parte da instância I utilizando as configurações representadas por $\lfloor x \rfloor$, chamaremos a instância formada pelas peças remanescentes de I'. Note que $\sum_j (x_j - \lfloor x_j \rfloor) \geq \text{lin}(I') \geq \text{size}(I')$, portanto, ao empacotar parte da instância I utilizando o método acima, utilizaremos no máximo $\text{lin}(I) - \text{size}(I')$ *bins*. Para que possamos empacotar as peças restantes, podemos utilizar uma quantidade menor do que ou igual a $m(I)$ *bins*, cada um com uma configuração do tipo j para cada $x_j - \lfloor x_j \rfloor$ não nulo. Por outro lado, utilizando o empacotamento do lema 5.1, poderemos empacotar em no máximo $2 \cdot \text{size}(I') + 1$ *bins*. Basta escolher o melhor dos empacotamentos, desta forma, utilizaremos no máximo

$$\min\{m(I), 2 \cdot \text{size}(I') + 1\} \leq \text{size}(I') + \frac{m(I)+1}{2}$$

bins. Logo, obtemos que $\text{opt}(I) \leq \text{lin}(I) + \frac{m(I)+1}{2}$. A última desigualdade foi obtida pelo simples fato de que $\min\{a,b\} \leq \frac{a+b}{2}$. □

A demonstraço do lema acima indica um algoritmo para resolver instâncias do PBP quando seus PBPFs relacionados têm soluções conhecidas.

Corolário 5.1. *Existe um algoritmo A que dada uma instância I do PBP e uma sbf x do PBPF associado, retorna uma solução com custo menor do que ou igual a $\mathbf{1}^T x + \frac{m(I)+1}{2}$. Particularmente, se x é uma sbf ótima, então $A(I) \leq \text{lin}(I) + \frac{m(I)+1}{2}$. O algoritmo executa em tempo $O(n(I)\log n(I))$.*

Todos os algoritmos de aproximação para o PBP que iremos descrever mais adiante, terão como subrotina um algoritmo que resolve o PBPF associado em tempo polinomial com uma certa tolerância $\varepsilon > 0$. Ao fim deste capítulo, mostraremos um algoritmo com essas características, utilizando o método GLS como subrotina principal.

5.2 Eliminação e Agrupamento

Vamos descrever nesta sessão três técnicas que auxiliam a construção de algoritmos para o PBP: *eliminação de itens pequenos*, *agrupamento linear* e *agrupamento geométrico*. Antes de descrevermos essas técnicas, vamos definir uma relação de ordem parcial e um operador binário para instâncias arbitrárias do PBP.

Definição 5.2. *Sejam $I = (\kappa, (r_i)_{i\in\kappa})$ e $J = (\iota, (s_j)_{j\in\iota})$ duas instâncias quaisquer do PBP. Seja $\preceq$ uma relação de ordem parcial entre instâncias do PBP. Dizemos que $I \preceq J$ se, e somente se existe uma função injetiva $f : \kappa \to \iota$ para o qual $r_i \leq s_{f(i)}$ para todo $i \in \kappa$.*

Claramente,

$$I \preceq J \Longrightarrow \begin{cases} \text{opt}(I) \leq \text{opt}(J) \\ \text{lin}(I) \leq \text{lin}(J) \\ \text{size}(I) \leq \text{size}(J) \end{cases}$$

Definição 5.3. *Sejam $I = (\kappa, (r_i)_{i\in\kappa})$ e $J = (\iota, (s_j)_{j\in\iota})$ duas instâncias quaisquer do PBP. Seja $\sim$ uma relação de equivalência entre instâncias do PBP. Dizemos que $I \sim J$ se, e somente se $\iota = \kappa$ e existe uma bijeção $f : \iota \to \iota$ tal que $r_{f(i)} = s_i$.*

Claramente,

$$I \sim J \Longrightarrow \begin{cases} \text{opt}(I) = \text{opt}(J) \\ \text{lin}(I) = \text{lin}(J) \\ \text{size}(I) = \text{size}(J) \end{cases}$$

Essa relação diz exclusivamente que duas instâncias são iguais a menos da ordem dos itens.

Definição 5.4. *Sejam $I = (\kappa, (r_i)_{i\in\kappa})$ e $J = (\iota, (s_j)_{j\in\iota})$ duas instâncias quaisquer do PBP. Definimos o operador binário $\cup$ como $I \cup J := (\eta, (v_k)_{k\in\eta})$ onde $\eta = \{1, \cdots, |\iota| + |\kappa|\}$ e*

$$v_k = \begin{cases} s_k, \text{ se } 1 \leq k \leq |\iota| \\ r_{k-|\iota|}, \text{ se } |\iota| < k \leq |\iota| + |\kappa| \end{cases}$$

Esse operador é a ideia intuitiva de "unir" os itens de duas instâncias arbitrárias do PBP. Embora o operador não seja comutativo, a comutação preserva a relação de equivalência definida acima, i.e., $I \cup J \sim J \cup I$.

5.2.1 Eliminação de Itens Pequenos

Uma estratégia para resolver instâncias do PBP é pôr de lado peças suficientemente pequenas, empacotar as peças maiores e depois inserir as peças descartadas criando um novo *bin* apenas quando necessário. O seguinte lema justifica essa tática.

Lema 5.3. *Sejam $I = (\iota, (s_i)_{i\in\iota})$ uma instância do PBP e $\varepsilon \in (0,1)$. Denomine de pequeno um item i tal que $s_i \leq \frac{\varepsilon}{2}$ e de grande caso contrário. Considere a instância I' formada apenas por peças grandes da instância I. Seja A um algoritmo que resolve instâncias do PBP e B um algoritmo que executa da seguinte maneira: rode A sobre a instância I' e a partir desta solução, insira as peças pequenas, usando um novo bin apenas quando necessário. Portanto, $B(I) \leq \max\{A(I'), (1+\varepsilon)\text{opt}(I)+1\}$.*

Demonstração. Se ao inserir os itens pequenos não for necessário o uso de um novo *bin*, então $B(I) = A(I')$. Se novos *bins* são usados, então existe no máximo um *bin* cuja soma das peças é menor do que $1-\frac{\varepsilon}{2}$. Portanto, temos que $\text{size}(I) \geq (1-\frac{\varepsilon}{2})(B(I)-1) \Longrightarrow B(I) \leq \frac{1}{1-\frac{\varepsilon}{2}}\text{size}(I) + 1 \leq (1+\varepsilon)\text{opt}(I)+1$. Para a última desigualdade, usamos o fato de que $\left(1-\frac{\varepsilon}{2}\right)(1+\varepsilon) \geq 1$ se $\varepsilon \in (0,1)$. □

5.2.2 Agrupamento Linear

A rotina de agrupamento linear tem como objetivo diminuir o número de tamanhos diferentes de peças sem que aumentemos muito o tamanho delas. A entrada do algoritmo é uma instância I do PBP e um inteiro positivo k.

Seja $I = (\iota = \{1,\cdots,n\}, (s_i)_{i\in\iota})$ uma instância do PBP com $(s_i)_{i\in\iota}$ não-crescente. Divida I em instâncias $G_1,\cdots,G_m$ com $G_1 \cup \cdots \cup G_m = I$ satisfazendo

$$G_i = \begin{cases} (\{1,\cdots,k\}, (s_{(i-1)k+1},\cdots,s_{i\cdot k})) \text{ se } i\cdot k \leq n \\ (\{1,\cdots,n-(i-1)k\}, (s_{(i-1)k+1},\cdots,s_n)) \text{ caso contrário} \end{cases}$$

Em outros termos, G_1 é formado pelos k maiores itens de I, G_2 pelos k seguintes maiores e assim sucessivamente. Obviamente, existe a relação

$$G_1 \succeq G_2 \succeq \cdots \succeq G_m$$

onde $|G_i| = k$ para $i = 1,\cdots,m-1$ e $|G_m| \leq k$. Se $G_i = (\kappa_i, (s_{(i)j})_{j\in\kappa_j})$ então definimos $G_i' = (\kappa_i, (s_{(i)1})_{j\in\kappa_j})$. Em outras palavras, G_i' tem a mesma cardinalidade de G_i e é só composta pelo maior item em G_i. Portanto:

$$G_1 \succeq G_2' \succeq G_2 \succeq \cdots \succeq G_m' \succeq G_m$$

A rotina de agrupamento linear retorna um par (J,J') onde $J' = G_1$ e $J = \bigcup_{i=2}^{m} G_i$. Claramente, $J \succeq I \succeq J \cup J'$. Portanto, o seguinte lema é verdadeiro.

Lema 5.4. *Seja I uma instância do PBP e $k \in \mathbb{Z}^+$. Se ao aplicarmos sobre (I,k) a rotina de agrupamento linear e o resultado for o par (J,J'), então temos que as seguintes desigualdades são válidas.*

1. $\text{opt}(J) \leq \text{opt}(I) \leq \text{opt}(J) + k$.
2. $\text{lin}(J) \leq \text{lin}(I) \leq \text{lin}(J) + k$.
3. $\text{size}(J) \leq \text{size}(I) \leq \text{size}(J) + k$.

Demonstração. Basta apenas demonstrar para uma das afirmações, as restantes são análogas.

$$\text{opt}(J) \leq \text{opt}(I) \leq \text{opt}(J) + \text{opt}(J') \leq \text{opt}(J) + k$$

□

5.2.3 Agrupamento Geométrico

Nesta sessão apresentaremos duas versões do agrupamento geométrico. As rotinas têm a mesma entrada que a rotina de agrupamento linear, com o mesmo objetivo de reduzir o número de tamanhos diferentes de itens.

Primeira Versão

Seja $I = (\iota, (s_i)_{i\in\iota})$ uma instância do PBP com $(s_i)_{i\in\iota}$ não-crescente e $k \in \mathbb{Z}^+$. Definimos I_j pela instância formada pelas peças de I cujos tamanhos estão no intervalo $(2^{-(j+1)}, 2^{-j}]$ para $j = 0, 1, \cdots, \left\lfloor \log \frac{1}{\alpha(I)} \right\rfloor$. Seja (J_j, J'_j) o resultado da aplicação da rotina de agrupamento linear sobre $(I_j, k \cdot 2^j)$. Evidentemente, $I = \bigcup_j I_j$. Definimos $J := \bigcup_j J_j$ e $J' := \bigcup_j J'_j$.

Lema 5.5. *Sejam I, J e k como os definidos acima. Então valem as seguintes desigualdades:*

1. $\text{opt}(J) \leq \text{opt}(I) \leq \text{opt}(J) + k \left\lceil \log \frac{1}{\alpha(I)} \right\rceil$.
2. $\text{lin}(J) \leq \text{lin}(I) \leq \text{lin}(J) + k \left\lceil \log \frac{1}{\alpha(I)} \right\rceil$.
3. $\text{opt}(J) \leq \text{opt}(I) \leq \text{opt}(J) + k \left\lceil \log \frac{1}{\alpha(I)} \right\rceil$.
4. $m(J) \leq \frac{2}{k}\text{size}(I) + \left\lceil \log \frac{1}{\alpha(I)} \right\rceil$.

Demonstração. Para as três primeiras inequações, basta mostrar que $\text{opt}(J') \leq k \left\lceil \log \frac{1}{\alpha(I)} \right\rceil$. Como $J' = \bigcup_r J'_r$, então $\text{opt}(J') \leq \sum_r J'_r$. Cada J'_r contém $k \cdot 2^r$ itens cujos tamanhos são menores do que ou iguais a 2^{-r}, o que implica que $\text{opt}(J'_r) \leq k \Longrightarrow \text{opt}(J') \leq k \left\lceil \log \frac{1}{\alpha(I)} \right\rceil$.

Para a última inequação, considere que $\text{size}(I_r) \geq 2^{-(r+1)} n(I_r)$. Como realizamos um agrupamento linear com parâmetro $k2^r$, então temos que $\left\lceil \frac{n(I_r)}{k2^r} \right\rceil = m(J_r) \Longrightarrow \frac{n(I_r)}{k2^r} \geq m(J_r) - 1$. Logo, temos que $\text{size}(I_r) \geq 2^{-(r+1)}((m(J_r) - 1)k2^r) \Longrightarrow \frac{2}{k}\text{size}(I_r) \geq m(J_r) - 1$. Portanto, somando todas as inequações, temos

$$m(J) - \left\lceil \log \frac{1}{\alpha(I)} \right\rceil \leq \sum_r (m(J_r) - 1) \leq \sum_r \frac{2}{k}\text{size}(I_r) = \frac{2}{k}\text{size}(I)$$

onde a desigualdade segue imediatamente. Portanto o lema está provado. □

Segunda Versão

Antes de apresentarmos o algoritmo, vamos enunciar um lema que será útil na análise que daremos.

Lema 5.6. $\ln(a+b)-\ln a \geq \frac{1}{a+1}+\cdots+\frac{1}{a+b}$ *onde b é inteiro positivo.*

Demonstração. Note que

$$\int_n^{n+1}\frac{1}{x}dx = \ln(n+1)-\ln n$$

Como $\frac{1}{x}$ é integrável e contínua em todo o seu domínio, então, pelo Teorema do Valor Médio, existe $n_0 \in (n,n+1)$ tal que $\frac{1}{n_0} = \ln(n+1)-\ln n$. Como $\frac{1}{x}$ é monotônica decrescente, então $\frac{1}{n_0} \geq \frac{1}{n+1}$, o que implica que $\ln(n+1)-\ln n \geq \frac{1}{n+1}$. Logo

$$\int_a^{a+b}\frac{1}{x}dx = \sum_{i=1}^{b}\int_{a+i-1}^{a+i}\frac{1}{x}dx \geq \sum_{j=1}^{b}\frac{1}{a+j}$$

.

O que prova o lema. □

O procedimento a seguir recebe uma instância I e um inteiro k como parâmetros. Ordene as peças da instância em ordem não-decrescente de acordo com o tamanho dos itens. Começando dos maiores itens para os menores, tome a instância G_1 formada pelos primeiros itens cuja soma dos tamanhos excede k. Repita o processo para as peças restantes até que não sobrem itens. Naturalmente, para a última instância G_q, poderemos ter que $\text{size}(G_q) < k$. Neste caso, podemos completar G_q com peças de tamanho 0 até que tenhamos $n(G_q) \geq n(G_{q-1})$. Definimos $l_i := n(G_i)$ para $i = 1,\cdots,q$. Logo, temos que $k+1 \leq l_1 \leq \cdots \leq l_q \leq \lceil \frac{k}{\alpha(I)} \rceil$. Desta forma, não podemos mais dizer que $G_i \succeq G_{i+1}$ uma vez que $n(G_i) \leq n(G_{i+1})$. Desta forma, definiremos ΔG_i e δG_i como as instâncias com as l_{i-1} maiores peças de G_i e com os $l_i - l_{i-1}$ itens restantes, respectivamente. Seja G_i' a instância gerada por δG_i aumentando o tamanho de cada item de ΔG_i para o tamanho do maior item presente nesta instância. Obviamente, $\Delta G_i \preceq G_i' \preceq G_{i-1}$ e $G_i \preceq G_i' \cup \delta G_i$ para $i = 2,\cdots,q$.

Defina $J = \bigcup_{i=2}^{q} G_i'$ e $J' = \left(\bigcup_{i=2}^{q}\delta G_i\right) \cup G_1$. Já que $J \preceq I \preceq J \cup J'$, então temos que

$$\text{size}(J) \leq \text{size}(I) \leq \text{size}(J) + \text{size}(J')$$

Lema 5.7. *Sejam I, J e k como os definidos acima. Então valem as seguintes desigualdades:*

1. $\text{size}(J) \leq \text{size}(I) \leq \text{size}(J) + k\left[2+\ln\frac{1}{\alpha(I)}\right]$.
2. $\text{lin}(J) \leq \text{lin}(I) \leq \text{lin}(J) + 2k\left[2+\ln\frac{1}{\alpha(I)}\right] + 1$.
3. $\text{opt}(J) \leq \text{opt}(I) \leq \text{opt}(J) + 2k\left[2+\ln\frac{1}{\alpha(I)}\right] + 1$.
4. $m(J) \leq \frac{\text{size}(J)}{k} + \ln\frac{1}{\alpha(I)}$.

Demonstração. A demonstração do lema pode ser feita apenas encontrando um bom limite superior para $\text{size}(J')$ e um bom limite inferior para $\text{size}(J)$. Note que

$$\frac{\text{size}(\delta G_i)}{n(\delta G_i)} \leq \frac{\text{size}(G_i)}{n(G_i)} \leq \frac{k}{l_i - 1} \Longrightarrow \text{size}(\delta G_i) \leq \frac{l_i - l_{i-1}}{l_i - 1} k$$

Logo, temos

$$\text{size}(J') \leq \text{size}(G_1) + \sum_{i=2}^{q} \text{size}(\delta G_i) \leq k \left(\frac{l_i}{l_i - 1} + \sum_{i=2}^{q} \frac{l_i - l_{i-1}}{l_i - 1} \right)$$

Perceba que

$$\frac{l_i - l_{i-1}}{l_i - 1} \leq \underbrace{\frac{1}{l_{i-1}} + \frac{1}{l_{i-1}+1} + \cdots + \frac{1}{l_{i-1}-1}}_{(l_i - l_{i-1}) \text{ termos}}$$

Portanto,

$$\begin{aligned} \text{size}(J') &\leq k \left(1 + \frac{1}{l_1 - 1} + \frac{1}{l_1} + \cdots + \frac{1}{l_q - 1} \right) \\ &\leq k \left(1 + \ln \frac{l_q - 1}{l_1 - 2} \right) \\ &\leq k \left(1 + \ln \frac{k}{(k-1)\alpha(I)} \right) \\ &\leq k \left(2 + \ln \frac{1}{\alpha(I)} \right) \end{aligned}$$

A segunda desigualdade decorre do lema 5.6, a penúltima de $l_q - 1 \leq \frac{k}{\alpha(I)}$ e $l_1 \geq k+1$ e a última do fato $\ln \frac{k}{k-1} \leq 1$ para $k > 1$. Portanto, temos

$$\text{lin}(J') \leq \text{opt}(J') \leq 2 \cdot \text{size}(J') + 1 \leq 2k \left(2 + \ln \frac{1}{\alpha(I)} \right) + 1$$

o que demonstra as três primeiras desigualdades. Para demonstrar a última, vamos achar um bom limite inferior para $\text{size}(J)$. Note que

$$\frac{\text{size}(G_i')}{n(G_i')} \geq \frac{\text{size}(G_i)}{n(G_i)} \geq \frac{k}{l_i} \Longrightarrow \text{size}(G_i') \geq k \frac{l_{i-1}}{l_i}$$

Logo,

$$\begin{aligned}
\text{size}(J) &\geq \sum_{i=2}^{q} k\frac{l_{i-1}}{l_i} \\
&= k\sum_{i=2}^{q}\left(1-\frac{l_i-l_{i-1}}{l_i}\right) \\
&\geq k(q-1)-k\left(\frac{1}{l_1+1}+\cdots+\frac{1}{l_q}\right) \\
&\geq k(q-1)-k\ln\frac{l_q}{l_1} \\
&= k(q-1-\ln\frac{l_q}{l_1}) \geq k(q-1-\ln\frac{1}{\alpha(I)})
\end{aligned}$$

Portanto, temos que $m(J) = q-1 \leq \frac{\text{size}(J)}{k} + \ln\frac{1}{\alpha(I)}$, o que demonstra a última desigualdade. □

5.3 Esquemas de Aproximação

Vamos apresentar nesta sessão os três principais algoritmos deste trabalho, que foram implementados e cuja análise experimental será dada no próximo capítulo. Cada um dos algoritmos executa a subrotina que resolve o PBPF para uma instância I com uma tolerância $\delta > 0$ dada, i.e., retornará uma sbf com custo menor do que ou igual a $\text{lin}(I)+\delta$. Se tivermos $n(I), \frac{1}{\alpha(I)} \leq n$, $m(I) \leq m$ e $\delta \geq 1$, então o tempo de execução da subrotina que resolve o PBPF não excede $T(m,n)$ onde T foi definida na primeira sessão deste capítulo.

Primeiro Algoritmo

Este algoritmo recebe uma instância I do PBP e um real positivo ε.

Algoritmo 2: ALGORITMO 1

Entrada: (ε, I)

Saída: Empacotamento para a instância I

início

Descarte todas as peças com tamanho menor do que ou igual a $\max\left\{\frac{1}{n}, \frac{\varepsilon}{2}\right\}$;
Chame de J a instância resultante e J' a instância formada pelas peças descartadas;
Realize agrupamento linear com entrada $(\lceil n(J)\varepsilon^2/2\rceil, J)$. Seja (K, K') o retorno da subrotina;
Empacote a instância K' em um *bin* por peça;
Aplique a subrotina do PBPF à instância K com tolerância $\delta = 1$. Seja x a sbf retornada pela subrotina;
Empacote a instância K com no máximo $\mathbf{1}^T x + \frac{m(K)+1}{2}$ *bins* (corolário 5.1);
Reduza as peças para o tamanho original;
Crie um empacotamento para a instância J com o empacotamento anterior junto com o empacotamento da instância K';
Crie um empacotamento para a instância I utilizando o empacotamento de J e com as peças de J', criando um novo *bin* só quando necessário.

fim

Teorema 5.1. *Se A implementa o* ALGORITMO 1*, então o seu tempo de execução é de*

$$O\left(n(I)\log n(I) + T\left(\frac{1}{\varepsilon^2}, n(I)\right)\right)$$

e ainda

$$A(I) \leq (1+\varepsilon)\text{opt}(I) + O(\varepsilon^{-2})$$

Demonstração. Para que possamos dar um limite superior para o tempo de execução do algoritmo, vamos quantificar um limite superior para $m(K)$:

$$m(K) \leq \frac{n(K)}{\lceil n(J)\varepsilon^2/2\rceil} \leq \frac{n(J)}{\lceil n(J)\varepsilon^2/2\rceil} \leq \frac{2}{\varepsilon^2}$$

Logo, a execução da subrotina que resolve o PBPF executa em tempo assintoticamente igual a $T(\varepsilon^{-2}, n(K)) \leq T(\varepsilon^{-2}, n(I))$. Para as demais operações, o tempo é limitado por $O(n(I)\log I)$. Portanto, o tempo total tem como limite superior $O(n(I)\log n(I) + T(\varepsilon^{-2}, n(I)))$.

Note que $n(J)\frac{\varepsilon}{2} \leq \text{size}(J) \leq \text{opt}(J) \Longrightarrow n(J)\frac{\varepsilon^2}{2} \leq \varepsilon \cdot \text{opt}(I) \Longrightarrow \lceil n(J)\varepsilon^2/2\rceil \leq \varepsilon \cdot \text{opt}(J) + 1$. Ao empacotar as instâncias K e K', utilizaremos no máximo $\mathbf{1}^T x + \frac{1+m(K)}{2} + \lceil n(J)\varepsilon^2/2\rceil \leq \text{opt}(I) + 1 + \frac{1+\frac{2}{\varepsilon^2}}{2} + \varepsilon \cdot \text{opt}(I) + 1 \leq (1+\varepsilon)\text{opt}(I) + \frac{1}{\varepsilon^2} + 3$ *bins*. Pelo lema 5.3, temos que

$$A(I) \leq \max\{(1+\varepsilon)\text{opt}(I) + \frac{1}{\varepsilon^2} + 3, (1+\varepsilon)\text{opt}(I) + 1\} = (1+\varepsilon)\text{opt}(I) + \frac{1}{\varepsilon^2} + 3$$

O que demonstra o teorema. □

Segundo Algoritmo

Este algoritmo recebe uma instância I do PBP, um real positivo ε e um inteiro positivo k.

Algoritmo 3: ALGORITMO 2

Entrada: (k, ε, I)

Saída: Empacotamento para a instância I

início

Elimine todas as peças com tamanho menor do que ou igual a ε;

repita

Realize a segunda forma de agrupamento geométrico com parâmetro k para criar instâncias J e J';

Empacote J' em no máximo $2k\left(2+\ln\frac{1}{\varepsilon}\right)+1$ *bins*;

Execute a subrotina que resolve o PBPF sobre a instância J com tolerância $\delta = 1$. Seja x a sbf retornada;

Para cada i cujo $x_i \geq 1$, crie $\lfloor x_i \rfloor$ *bins* com configuração i e elimine os itens empacotados;

até $\text{size}(I) \leq 1 - \frac{k}{k-1}\ln\varepsilon$;

Empacote as peças restantes em no máximo $2 - \frac{2k}{k-1}\ln\varepsilon$ *bins*;

Insira as peças eliminadas no primeiro passo criando um novo *bin* apenas quando necessário;

fim

Vamos dar uma análise para o tempo de execução e taxa de aproximação do ALGORITMO 2.

Teorema 5.2. *Existe um algoritmo de tempo polinomial A que resolve o PBP tal que, para uma dada instância I, temos que*

$$A(I) \leq \text{opt}(I) + O(\log^2 \text{opt}(I))$$

e executa em tempo $O(T(n(I), n(I)) + n(I)\log n(I))$.

Demonstração. Suponha que A implementa o ALGORITMO 2. Seja t o número de vezes que em que o laço principal é executado. Para $i = 1, 2, \cdots, t$, defina I_i, J_i, J_i' as instâncias criadas em cada iteração do laço: I_i a instância existente no início da i-ésima iteração e J_i, J_i' as instâncias criadas a partir do agrupamento geométrico de I_i. Defina X_i como o número de *bins* criados na i-ésima iteração utilizando x (gerado pela subrotina que resolve o PBPG sobre a instância J_i) e X_i' o número de *bins* utilizados para empacotar J_i'. Vamos obter um limite superior para t, o número de iterações do laço principal. Note que $I_1 = I$.

$$\text{size}(I_{i+1}) \leq \text{lin}(I_{i+1}) \leq \sum_j (x_j - \lfloor x_j \rfloor)$$

Onde, na última desigualdade, x é o resultado da execução da subrotina que resolve o PBPG sobre a instância J_i. Portanto:

$$\sum_j (x_j - \lfloor x_j \rfloor) \leq m(J_i) \leq \frac{\text{size}(J_i)}{k} + \ln\frac{1}{\varepsilon} \leq \frac{\text{size}(I_i)}{k} + \ln\frac{1}{\varepsilon}$$

Desta forma, temos, por indução em i que

$$\text{size}(I_{i+1}) \leq \frac{1}{k^i}\text{size}(I) - \frac{k}{k-1}\ln\varepsilon$$

Como $\text{size}(I_t) > 1 - \frac{k}{k-1}\ln\varepsilon$, portanto temos que $t \leq \frac{\ln \text{size}(I)}{\ln k} + 1$. Para obter um limite superior no número de *bins* produzidos pelo algoritmo, vamos utilizar as seguintes desigualdades:

$$\text{lin}(I_{i+1}) \leq \text{lin}(J_i) + 1 - X_i \leq \text{lin}(I_i) + 1 - X_i$$

Utilizando somatório telescópico, obtemos que $\sum_{i=1}^{t} X_i \leq \text{lin}(J) + t$. Ao empacotar as peças restantes ao final do laço principal, a quantidade de *bins* até então será limitada por

$$\sum_{i=1}^{t} X_i + t[2k(2 - \ln\varepsilon)] + 2 - \frac{2k}{k-1}\ln\varepsilon$$

Portanto, pelo lema 5.3,

$$A(I) \leq \max\left\{(1+2\varepsilon)\text{opt}(I) + 1, \text{opt}(I) + \left[1 + \frac{\ln \text{size}(I)}{\ln k}\right][1 + 4k - 2k\ln\varepsilon] + 2 + \frac{2}{1 + \frac{1}{k}\ln\varepsilon}\right\}$$

Tome $k = 2$ e $\varepsilon = \frac{1}{\text{size}(I)}$ e obteremos o limite do número de *bins* do teorema. A análise de tempo será omitida.

□

Teorema 5.3. *Existe um algoritmo de tempo polinomial A que resolve o PBP tal que, para uma dada instância I e um real* $0 < \alpha < 1$*, temos que*

$$A(I) \leq \text{opt}(I) + O(\text{opt}(I)^{\alpha} \cdot \log \text{opt}(I))$$

e executa em tempo $O(T(n(I)^{1-\alpha}, n(I)), n(I)\log n(I))$

Demonstração. Tome $k = \text{size}(I)^{\alpha}$ e $\varepsilon = \text{size}(I)^{\alpha-1}$ para o ALGORITMO 2 e obteremos o limite do número de *bins* do teorema. A análise de tempo será omitida. □

Terceiro Algoritmo

Algoritmo 4: ALGORITMO 3

Entrada: I
Saída: Empacotamento para a instância I
início
  Elimine todas as peças com tamanho menor do que ou igual a $\frac{\log^2 m(I)}{\text{size}(I)}$. Chame a instância resultante de K;
  se $m(K) \leq \text{size}(K)$ **então**
    Execute a rotina que resolve o PBPF sobre a instância K com tolerância $\delta = 1$.
    Seja x a sbf retornada;
    Para cada i tal que $x_i \geq 1$, crie $\lfloor x_i \rfloor$ *bins* com configuração i e delete os itens empacotados;
  fim
  Execute o laço do ALGORITMO 2 sobre a instância definida pelas peças restantes;
  Insira os itens eliminados no primeiro passo, usando um novo *bin* apenas quando necessário;
fim

Teorema 5.4. *Se A implementa o* ALGORITMO 3, *então seu tempo de execução é de*

$$O(T(m(I),n(I)) + n(I)\log n(I))\log m(I)$$

E ainda, $A(I) \leq \text{opt}(I) + O(\log^2 m(I))$

Demonstração. Demonstração omitida. Similar à demonstração para o ALGORITMO 1. □

5.4 Rotina do Bin Packing Fracionário

A rotina que resolve o PBPF utiliza o método GLS para resolver instâncias arbitrárias do problema de otimização com função objetivo linear, dado o fato de que existe o "oráculo separador de hiperplanos", o qual chamamos de *SEP* no capítulo anterior. Este algoritmo será construído através da resolução de instâncias simplificadas do Problema Knapsack a cada passo.

5.4.1 O Dual do Problema

Resolver diretamente o PBPF pode ser uma tarefa muito custosa computacionalmente: haverá uma variável para cada configuração possível dentro de um único *bin*, o que pode levar a uma um número enorme de variáveis. Consideramos o problema dual: teremos apenas $m(I)$ variáveis, mas uma restrição por cada configuração. Contudo, podemos escolher apenas m restrições de modo que o ótimo do problema não seja afetado. Para descobrir como essa escolha pode ser feita, vamos analisar alguns fatos.

O problema primal tem a seguinte forma:

$$\begin{aligned} \min \ & \mathbf{1}^T \cdot x \\ & Ax \geq b \\ & x \geq 0 \end{aligned}$$

enquanto seu dual será

$$\begin{aligned} \max \ & b^T \cdot u \\ & A^T u \leq \mathbf{1} \\ & u \geq 0 \end{aligned}$$

Sejam x^* e u^* os ótimos dos problemas primal e dual, respectivamente. O teorema 4.7 nos diz que, $x^* = u^*$. Esse é um fato primordial para que possamos selecionar um subconjunto de restrições do problema dual de modo a minimizar o número de variáveis do problema primal.

Podemos dar a seguinte interpretação para o problema dual. Cada variável u_i de $u = (u_1, \cdots, u_m)^T$ é o preço de um item do problema, cuja função objetivo busca maximizar a soma total dos preços com a restrição de que a soma dos preços das peças em um único *bin* é menor do que ou igual a 1.

Nós iremos resolver o problema dual utilizando o método GLS com uma dada tolerância δ que iremos definir posteriormente. Considere a tarefa realizada pelo algoritmo *SEP*. Para o problema dual, existem duas classes de restrições sobre as variáveis: a não negatividade (cujo problema de sepração é trivial) e a restrição de que a soma das peças em um único *bin* não pode exceder 1. Seja $s = (s_1, \cdots, s_m)^T$ o vetor com o tamanho dos itens. Desta forma, uma solulção u é factível se, e somente se o valor ótimo do problema Knapsack

$$\begin{aligned} \max \ & v^T \cdot u \\ & s^T \cdot u \\ & v \geq 0, \ v \in \mathbb{Z}^n \end{aligned}$$

é menor do que ou igual a 1. O vetor v indica a configuração de um *bin* que apresenta maior custo utilizando o vetor de custos u, e portanto, só precisamos considerar o valor desta configuração e desprezar as demais. Portanto, não precisamos explicitamente determinar a factibilidade de uma solução através da verificação das restrições definidas por $A^T u \leq \mathbf{1}$.

Estamos longe de querer que a cada chamada à rotina *SEP*, seja resolvida uma instância arbitrária de um problema *NP*-difícil. Desta forma, podemos aproximar a instância do Problema Knapsack gerada pela iteração por uma instância que conseguimos resolver facilmente.

A cada iteração do método GLS, teremos uma elipsóide com centro u. A partir desta solução, iremos obter um vetor $\tilde{u}$ próximo de u, em que cada componente é um múltiplo de $\frac{\delta}{n}$. Sejam $u = (u_1, \cdots, u_m)^T$ e $\tilde{u} = (\tilde{u}_1, \cdots, \tilde{u}_m)^T$. Logo, $u_i = k_i \frac{\delta}{n}$ onde k_i é um inteiro positivo que satisfaz as seguintes desigualdades

$$k_i \frac{\delta}{n} \leq u_i < (k_i + 1) \frac{\delta}{n}$$

portanto, teremos $k_i \leq \frac{n}{\delta}u_i < k+1 \Longrightarrow k_i = \lfloor \frac{n}{\delta}u_i \rfloor$. Logo, teremos $\tilde{u}_i = \frac{\delta}{n}\lfloor \frac{n}{\delta}u_i \rfloor$. Como vetor $\tilde{u}$ é formado apenas por múltiplos de $\frac{\delta}{n}$, então vamos considerar o vetor $\bar{u} = \frac{n}{\delta}\tilde{u}$, cujas componentes são claramente inteiras. Seja $s = (s_1, \cdots, s_m)^T$ o vetor cujo i-ésimo elemento é o tamanho do i-ésimo item. Considere $F(k)$ o tamanho mínimo de um conjunto de peças cujo preço total é $k \cdot \frac{\delta}{n}$. Portanto, temos que $F(0) = 0$ e

$$F(k) = \min_{k-\bar{u}_i \geq 0} \{F(k-\bar{u}_i) + s_i\}, \quad \text{se } k > 0$$

O vetor F é calculado utilizando programação dinâmica, onde k varia de 0 até $\lceil \frac{n}{\delta} \rceil$. Desta forma, o vetor $\tilde{u}$ é factível se, e somente se $k > 1 \Longrightarrow F(k) > 1$. O custo para preencher a tabela F, que determina se $\tilde{u}$ é factível, é $O\left(\frac{nm}{\delta}\right)$. Dependendo do resultado da saída do cálculo do Knapsack, duas possibilidades surgem:

1. O ponto $\tilde{u}$ é infactível. Nesse caso, o cálculo da programação dinâmica determina um vetor α que corresponde a alguma configuração, tal que $\tilde{u}^T \cdot \alpha > 1$. Como $u^T \cdot \alpha > \tilde{u}^T \cdot \alpha > 1$, portanto u é infactível. Um corte de "factibilidade" é feito, impondo a restrição $u^T \cdot \alpha \geq \tilde{u}^T \cdot \alpha$.

2. O ponto $\tilde{u}$ é factível. Nesse caso, u pode ser ou não factível. De todo modo, um corte de otimalidade através de u é feito impondo a restrição $u^T \cdot b \geq \tilde{u}^T \cdot b$.

O limite do número de iterações do método GLS para resolver um PPL com tolerância δ também se aplica a esse algoritmo. Temos que $B(u_0, r) \subseteq F \subseteq B(u_1, R)$ onde F é a região factível para o problema dual. Cada componente de u_0 é $\frac{\alpha(I)}{2}, r = \frac{\alpha(I)}{2}$ e cada componente de u_1 é $\frac{1}{2}, R = \frac{\sqrt{m}}{2}$. O algoritmo modificado pode terminar em $M = 4m^2 \left\lceil \ln \frac{mn}{\alpha\delta} \right\rceil$ iterações com a garantia de que o resultado obtido não excede o valor ótimo em mais de δ. O tempo de execução do algoritmo modificado, levando em consideração o cálculo do Knapsack e os passos do método GLS, é limitado por $O\left(Mm\left(Mm + \frac{n}{\delta}\right)\right)$.

5.4.2 A Subtorina do Bin Packing Fracionário

Como agora conseguimos encontrar uma solução ótima para o dual do PBPF, vamos desenvolver uma rotina que resolve o PBPF com um número muito menor de variáveis. O PBPF e seu dual são:

$$\min \mathbf{1}^T \cdot x \text{ sujeito a } x \geq 0,\ Ax \geq b \tag{5.1}$$

$$\max b^T \cdot u \text{ sujeito a } u \geq 0,\ A^T u \leq \mathbf{1} \tag{5.2}$$

Seja z o valor ótimo do PPL descrito em (5.1). A subrotina que resolve o PBPF retorna uma solução factível com custo limitado superiormente por $z+h$, onde h é a tolerância especificada. A subrotina involve um outro parâmetro δ que será especificado abaixo. O primeiro passo da subrotina é executar o método GLS modificado para o PPL (5.2) para obter uma solução u^* com valor limitado inferiormente por $z - \delta$.

O objetivo desse passo é eliminar uma grande quantidade de configurações de *bins*. Lembremos que cada variável em (5.1) corresponde a uma configuração, que por sua vez correponde a uma restrição em (5.2). A partir da execução do método GLS, é possível obter um PPL D' com mesma função objetivo e variáveis que (5.2) mas com apenas as seguintes restrições

- Restrições do tipo $0 \leq u_i \leq 1,\ i = 1, \cdots, m$
- Restrições do tipo $u^T \cdot v \leq 1$ retornadas pelo oráculo do hiperplano separador, em todos os casos em que o ponto $\tilde{u}$ foi considerado infactível.

Não há como distinguir a execução do método GLS para (5.2) e para D', pois o comportamento do oráculo do hiperplano separador vale para os dois problemas. Portanto, tome z' como o valor ótimo de D'. Como D' é um relaxamento de (5.2), então temos que $u^{*T} \cdot b \leq z \leq z' \leq u^{*T} \cdot b + \delta$. Segue que $z \leq z' \leq z + \delta$; o que nos mostra que escolher o dual de D' introduz um pequeno erro ao invés de escolher o dual de (5.2) que é (5.1).

A subrotina que resolve o PBPF continua eliminando gradualmente restrições em D' sem que o valor do ótimo cresça significantemente. Eventualmente, teremos apenas m restrições linearmente independentes que serão da forma $-u_i \leq 0$ ou $u^T \cdot \alpha \leq 1$, onde α corresponderá a uma configuração de *bin*. Teremos um novo PPL D'' da forma $\max b^T \cdot u$ sujeito a $B^T u \leq r$ onde r é um vetor de 0s e 1s e B é uma matriz quadrada invertível de dimensões $m \times m$. O dual deste PPL é um PPL que tem como única sbf o vetor $x = B^{-1}b$. Cada componente de x corresponderá a uma variável de folga para restrição do tipo $-u_i \leq 0$ ou uma configuração de *bin* para a restrição de tipo $u^T \cdot \alpha \leq 1$. As componentes que correspondem a configurações serão utilizadas para determinar uma solução quase-ótima para o PBPF.

Em um determinado passo do procedimento de eliminação de restrições, teremos um PPL $\tilde{D}$ com mesma função objetivo de D' mas com apenas um subconjunto de restrições de D'. Também teremos uma solução factível $\tilde{u}$, tal que $\tilde{z} - \delta \leq \tilde{u}^T \cdot b \leq \tilde{z}$ onde $\tilde{z}$ é o valor ótimo de $\tilde{D}$. O procedimento de eliminação continua por selecionar um subgrupo de restrições em $\tilde{D}$, as retira e realiza um teste para verificar se o valor da solução obtida para o novo PPL está dentro de uma tolerância δ de $\tilde{u} \cdot b$. Se obtiver sucesso, o grupo de restrições é retirado permanentemente e $\tilde{D}$ é atualizado. Caso contrário, o procedimento seleciona outro subgrupo de restrições de $\tilde{D}$ e continua o processo.

É um fato fundamental em programação linear que em qualquer PPL limitado com m variáveis, existe um conjunto de m restrições que determina o valor ótimo, isto é, o valor ótimo não iria melhorar caso as outras restrições fossem eliminadas. Se T é esse conjunto crítico de restrições e S é um conjunto disjunto de T a ser eliminado do conjunto total de restrições, então o método GLS irá suceder em eliminar as restrições S desnecessárias. Em um passo geral do procedimento de eliminação de restrições, iremos particionar o conjunto de restrições em $m+1$ subconjuntos de tamanho similar e testar cada subgrupo até obter sucesso em eliminar. Pelo princípio de casa de pombos, pelo menos um desses grupos será disjunto de T, e o processo se repetirá até que tenhamos apenas m restrições.

Seja Q o número de restrições em D'. Note, que pela definição de D', $Q < M + 2m$, sendo M o número máximo de iterações do método GLS modificado. É fácil demonstrar que o número de tentativas bem sucedidas no processo de eliminação de restrições para reduzir ao número m

de restrições é limitado por

$$\left\lceil \frac{Q}{m+1} \right\rceil + (M+1)\ln \left\lceil \frac{Q}{m+1} \right\rceil + 1$$

Portanto, o número de execuções do método GLS é limitado por

$$(m+1)\left[\left\lceil \frac{Q}{m+1} \right\rceil + (M+1)\ln \left\lceil \frac{Q}{m+1} \right\rceil + 1 \right]$$

Uma versão aleatória do procedimento de eliminação de restrições pode realizar a tarefa em $O(m+1)\left(\ln \left\lceil \frac{Q}{m+1} \right\rceil + 1\right)$ chamadas ao método GLS. Faça

$$\delta := \frac{h}{\left\lceil \frac{Q}{m+1} \right\rceil + (M+1)\ln \left\lceil \frac{Q}{m+1} \right\rceil + 2}$$

Portanto o custo do empacotamento obtido não excederá o valor ótimo por mais de h. Desde modo, o algoritmo de tempo polinomial que resolve o PBPF para uma instância I do PBP pode ser especificado. Note que ele sempre retornará uma solução factível de valor limitado superiormente por $\mathrm{lin}(I) + h$.

Algoritmo 5: SUBROTINA QUE RESOLVE O PBPF

Entrada: (h, I)

Saída: Uma factível do PBPF para a instância I com tolerância h

início

Faça $\delta := \frac{h}{\left\lceil \frac{Q}{m+1} \right\rceil + (M+1)\ln \left\lceil \frac{Q}{m+1} \right\rceil + 2}$;
Execute o método GLS modificado com tolerância δ no PPL descrito em (5.2);
Extraia da execução do método GLS modificado uma instância D' com as mesmas variáveis e função objetivo que (5.2) mas com apenas $Q \leq M + 2m$ restrições;
Utilize o procedimento de eliminação de restrições para obter um PPL D^* com as mesmas variáveis e função objetivo de D' mas apenas m restrições;
Obtenha uma solução do PBPF para a instância I com tolerância h resolvendo diretamente o dual de D^*.

fim

Segue que o tempo de execução desse algoritmo para uma versão aleatória do procedimento de eliminação de restrições é limitada por

$$O\left(m^7 \log m \log^2\left(\frac{mn}{\alpha h}\right) + \frac{m^4 n \log m}{h} \log\left(\frac{mn}{\alpha h}\right) \right)$$

Daí temos a função de custo $T(m,n)$ definida no início do capítulo.

CAPÍTULO 6

Conclusão

Se o contexto do Problema do Bin Packing Unidimensional é um monolito, com este trabalho, fizemos apenas arranhar a superfície. Mostramos que é possível obter algoritmos eficientes que têm uma taxa de aproximação assintótica sublinear sobre o valor ótimo do problema. Contudo, entendamos o termo "eficiente" por tempo polinomial. O método GLS, apesar de polinomial, é lento na prática devido ao seu custo computacional com grande expoente. Esse fato sugere trabalhos posteriores neste sentido.

6.1 Trabalhos Futuros

Em trabalhos futuros, iremos implementar os algoritmos descritos no **Capítulo 5** e comparar os resultados em termos de qualidade e tempo de execução com os demais algoritmos de aproximação que podemos obter na literatura. Um dos grandes desafios é lidar com a grande instabilidade numérica decorrente de inúmeros iterações do método GLS e do método GLS modificado. Também iremos analisar a possibilidade de substituir, para a subrotina que resolve o PBPF, o método GLS modificado pelo Simplex e até mesmo o Método dos Pontos Interiores. Tal substituição pode não ser possível devido à natureza dos algoritmos de programação linear. Contudo, até mesmo a verificação desta possibilidade parece ser um grande desafio pela frente.

APÊNDICE A

Noções Básicas de Álgebra Linear

A.1 Norma e Produto Interno

Neste apêndice, se F é um corpo, então qualquer espaço vetorial definido sobre F será chamado de F-espaço vetorial. A seguir, daremos a definição de produto interno.

Definição A.1. *Seja F um corpo podendo ser $\mathbb{R}$ ou $\mathbb{C}$ e V um F-espaço. Uma operação*

$$\langle \cdot, \cdot \rangle : V \times V \longrightarrow F$$

é chamada de produto interno se, $\forall \alpha \in F$ e $\forall u, v, w \in V$, temos as seguintes propriedades

1. *Simetria conjugada: $\langle u, v \rangle = \overline{\langle v, u \rangle}$.*
2. *Linear à esquerda: $\langle \alpha u + v, w \rangle = \alpha \langle u, w \rangle + \langle v, w \rangle$.*
3. *Definidade positiva: $\langle u, u \rangle \geq 0$ e $\langle u, u \rangle = 0 \iff u = 0$.*

Naturalmente, se $F = \mathbb{R}$, então $\langle \cdot, \cdot \rangle$ é simétrica e bilinear. Veremos a seguir a definição de norma e sua relação com produto interno em um espaço vetorial.

Definição A.2. *Seja F um corpo podendo ser $\mathbb{R}$ ou $\mathbb{C}$ e V um F-espaço. Uma operação*

$$\| \cdot \| : V \longrightarrow \mathbb{R}$$

é chamada de norma se, $\forall \alpha \in F$ e $\forall u, v \in V$, temos as seguintes propriedades

1. *Positividade: $\|u\| \geq 0$, e $\|u\| = 0 \iff u = 0$.*
2. *Homogeneidade: $\|\alpha u\| = |\alpha| \|u\|$.*
3. *Desigualdade triangular: $\|u + v\| \leq \|u\| + \|v\|$.*

Exemplo A.1. *Se V é um espaço com produto interno $\langle \cdot, \cdot \rangle$ então a aplicação*

$$\begin{aligned} \| \cdot \| : V &\longrightarrow \mathbb{R} \\ v &\longmapsto \langle v, v \rangle^{1/2} \end{aligned}$$

é uma norma. A demonstração desse fato é simples, e será omitida deste texto.

Exemplo A.2. *Seja $x = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$. No $\mathbb{R}^n$ como $\mathbb{R}$-espaço, os seguintes operadores são normas:*

1. *Norma euclidiana: $\|x\| := (x_1^2 + \cdots + x_n^2)^{1/2}$.*
2. *Norma-p: $\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}$, para $1 \leq p < \infty$. Com $p = 2$ temos o caso da norma euclidiana.*
3. *Norma do máximo: $\|x\|_\infty = \max\{|x_1|, \cdots, |x_n|\}$.*

A.2 Matrizes e Positividade

Nesta sessão iremos introduzir o conceito de matriz (semi)definida positiva e de normas matriciais induzidas. Neste texto, denotaremos por $GL_n(\mathbb{R})$ o conjunto de todas as matrizes quadradas com entradas reais e invertíveis.

Definição A.3. *Seja $A \in GL_n(\mathbb{R})$ e simétrica. A é (semi)definida positiva se, para qualquer $x \in \mathbb{R}^n$ não nulo, temos*

$$x^T A x (\geq) > 0$$

Lema A.1. *Seja $A = (a_{ij})_{n\times n} \in GL_n(R)$ definida positiva. Então $a_{ii} > 0$ para $i = 1, \cdots, n$.*

Demonstração. Seja $\{e_1, \cdots, e_n\}$ a base canônica do $\mathbb{R}^n$. Portanto, $e_i^T A e_i = a_{ii} > 0$ para $i = 1, \cdots, n$. □

Teorema A.1. *Seja $A = (a_{ij})_{n\times n} \in GL_n(\mathbb{R})$ e $Q(x) = x^T A x$ sua forma quadrática. Definimos $A_k := (a_{ij})_{k\times k}$ para $k = 1, \cdots, n$. Então as seguintes afirmações são equivalentes:*

1. *$Q(x) > 0$ para qualquer $x \in \mathbb{R}^n$ não nulo.*
2. *Todos os autovalores de A são positivos.*
3. *Toda forma quadrática de Q_k de A_k é positiva definida.*
4. *$\det A_k > 0$ para $k = 1, \cdots, n$.*

Demonstração. Já que A é simétrica, então todos os seus autovalores são reais. Considere, por absurdo, que existe um autovalor de λ de A não positivo. Seja v um autovetor correspondente. Desta forma, temos que $v^T A v = \lambda v^T v = \lambda \|v\|^2 \leq 0$, o que contradiz (1). Logo, $(2) \Longrightarrow (1)$. Por outro lado, já que A é simétrica, então existe uma base ortogonal de autovetores $v_1, \cdots, v_n$ com os respectivos autovalores $\lambda_1, \cdots, \lambda_n$. Desta forma, seja $P = [v_1 \cdots v_n]$ a matriz cujas colunas são os autovetores de A e $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_n)$. Portanto, $\Lambda = PAP^T$. Defina $y := P^{-1}x$. Desta forma teremos

$$Q(x) = x^T A x = y^T P^T A P y = y^T \Lambda y = \sum_{i=1}^{n} \lambda_i y_i^2$$

Logo (1) e (2) são equivalentes.

Assuma que Q é positiva definida. Desta forma, para $1 \leq k \leq n$, temos que

$$0 < Q(x_1, \cdots, x_k, 0, \cdots, 0) = (x_1, \cdots, x_k, 0, \cdots, 0)A(x_1, \cdots, x_k, 0, \cdots, 0)^T = (x_1, \cdots, x_k)A(x_1, \cdots, x_k)^T$$

Desta forma, $(1) \Longrightarrow (3)$. Obviamente $(3) \Longrightarrow (1)$.

Assumindo (3), temos que cada forma quadrática Q_k é positiva definida, portanto todos os autovalores de cada A_k são positivos. Como o determinante é o produto dos autovalores, então $(3) \Longrightarrow (4)$. □

A.3 Matrizes e Normas

Nesta sessão iremos dar uma definição de norma de uma matriz real.

Definição A.4. *Uma norma de matriz $\|\cdot\|$ no espaço de matrizes quadradas $n \times n$ em $M_n(\mathbb{R})$ (espaço de todas as matrizes quadradas $n \times n$ com entradas reais) tem a mesma definição de norma no $\mathbb{R}^n$ com a propriedade adicional de que*

$$\|AB\| \leq \|A\| \cdot \|B\|$$

Para todos $A, B \in M_n(\mathbb{R})$.

Definição A.5. *Seja uma matriz $A \in M_n(\mathbb{R})$ com autovalores $\lambda_1, \cdots, \lambda_n$. O* ***raio espectral*** *de A é detonado por $\rho(A)$ cujo valor é dado por*

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i|$$

Definição A.6. *Se $\|\cdot\|$ é uma norma em $\mathbb{R}^n$, então definimos a função $\|\cdot\|$ em $M_n(\mathbb{R})$ tal que*

$$\|A\| = \sup_{\|x\|=1, x \in \mathbb{R}^n} \|Ax\|$$

A função $A \to \|A\|$ é chamda de ***norma subordinada de matriz*** *ou* ***operador norma*** *induzido pela norma $\|\cdot\|$*

E é possível verificar que a função acima define uma norma para as matrizes do espaço $M_n(\mathbb{R})$. O teorema a seguir descreve um modo de calcular a norma de matriz induzida pela norma euclidiana.

Teorema A.2. *Seja $\|\cdot\|_2$ a norma euclidiana para $\mathbb{R}^n$. Portanto, para $A \in M_n(\mathbb{R})$, temos*

$$\|A\|_2 = \sqrt{\rho(A^T A)}$$

A norma $\|\cdot\|_2$ de matriz induzida pela norma euclidiana, é frequentemente chamada de **norma espectral**.

Referências Bibliográficas

[ACG+02] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2002.

[Chv83] Vasek Chvátal. *Linear Programming*. W. H. Freeman and Company, 1983.

[dlVL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. 1981.

[FMCF01] Cristina G. Fernandes, Flávio K. Miyazawa, Márcia Cerioli, and Paulo Feofiloff. *Uma Introdução Sucinta aos Algoritmos de Aproximação*. Impa, 2001.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Macmillan Higher Education, 1979.

[GLS81] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. 1981.

[GLS93] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1993.

[JDU+74] David S. Johnson, S. Demers, J. D. Ullman, M. R. Garey, and L. R. Graham. Worst-case performance bound for simple one-dimensional packing algorithms. 1974.

[Kar91] Howard Karloff. *Linear Programming*. Birkhäuser, 1991.

[KK82] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. 1982.

[PS98] Christos H. Papadimitriou and Keneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[Vaz02] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2002.