CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Modelagem Matemática e Computacional

Algoritmos e Modelos Matemáticos para o Problema da k-Partição de Números

Alexandre Frias Faria

Orientador : Prof. Dr. Sérgio Ricardo de Souza

Co-Orientador : Prof. Dr. Carlos Alexandre Silva

Belo Horizonte
Fevereiro de 2017

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Diretoria de Pesquisa e Pós-Graduação

Programa de Pós-Graduação em Modelagem Matemática e Computacional

Algoritmos e Modelos Matemáticos para o Problema da k-Partição de Números

Dissertação submetida ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Alexandre Frias Faria

Orientador : Prof. Dr. Sérgio Ricardo de Souza

Co-Orientador : Prof. Dr. Carlos Alexandre Silva

Belo Horizonte
Fevereiro de 2017

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

"ALGORITMOS E MODELOS MATEMÁTICOS PARA O PROBLEMA DA K-PARTIÇÃO DE NÚMEROS"

Dissertação de Mestrado apresentada por **Alexandre Frias Faria**, em 16 de fevereiro de 2017, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Prof. Dr. Sérgio Ricardo de Souza (Orientador)
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Carlos Alexandre Silva (Coorientador)
Instituto Federal de Minas Gerais

Profª. Drª. Alessandra Martins Coelho
Instituto Federal Sudeste de Minas Gerais

Prof. Dr. Rodrigo Tomás Nogueira Cardoso
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida a impressão,

Prof. Dr. José Geraldo Peixoto de Faria
Coordenador do Programa de Pós-Graduação em
Modelagem Matemática e Computacional

Faria, Alexandre Frias
F224a Algoritmos e modelos matemáticos para o problema da k-partição de números. / Alexandre Frias Faria. – – Belo Horizonte, 2017.

x, 67 f. : il.

Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais, Programa de Pós-Graduação em Modelagem Matemática e Computacional, 2017.

Orientador: Prof. Dr. Sérgio Ricardo de Souza
Coorientador: Prof. Dr. Carlos Alexandre Silva

Bibliografia

1. Otimização Combinatória. 2. Programação (Matemática). 3.Heurística. I. Souza, Sérgio Ricardo de. II. Centro Federal de Educação Tecnológica de Minas Gerais. III. Título

CDD 519.6

Elaboração da ficha catalográfica pela Biblioteca-Campus II / CEFET-MG

Agradecimentos

Aos meus amigos, que sempre me ajudam muito;
Aos colegas, que proporcionaram boas trocas durante o mestrado;
Em especial ao meu orientador Prof. Dr. Sérgio Ricardo de Souza pela orientação, paciência, compromisso e conselhos.
Ao meu co-orientador Prof. Dr. Carlos Alexandre Silva por acompanhar o desenvolvimento deste trabalho;
Aos meus professores do curso de Mestrado em Modelagem Matemática e Computacional do CEFET-MG, pelos bons ensinamentos;
Ao CEFET-MG, FAPEMIG e CAPES pelo apoio financeiro.

Resumo

Esta dissertação de mestrado apresenta uma formulação matemática e algoritmos para a versão de otimização do Problema da k-Partição de Números. Este problema consiste em distribuir os elementos de uma sequência numérica em k subconjuntos disjuntos, de modo que os resultados das somas dos elementos de cada subconjunto fiquem contidos no menor intervalo possível. A formulação matemática proposta é baseada em dois problemas semelhantes da literatura. Elimina-se a multiplicidade de soluções do modelo de programação inteira, reduzindo ao mínimo o número de variáveis. Mostra-se a NP-completude do problema estudado. Estudam-se algoritmos heurísticos, aproximados e exatos da literatura aplicáveis aos problemas relacionados. Aplica-se a meta-heurística ILS (*Iterated Local Search*) à solução gerada por um algoritmo aproximado. A estratégia utilizada é aplicar método guloso em todas as fases do algoritmo, tanto na inicialização quanto para a escolha de movimentos do ILS. Propõe-se uma melhoria de métodos exatos *Branch-and-Bound* aplicados ao problema e presentes na literatura . Esta modificação refina os limitantes do método, usando a relaxação do modelo matemático e ILS propostos. Insere-se um critério de busca para que os nós mais promissores tenham prioridade. Os resultados mostram a solução do ILS superando as soluções de heurísticas construtivas listadas na literatura. A qualidade da solução do ILS é certificada por métodos exatos, um da literatura e outro com as melhorias propostas neste trabalho. As modificações do algoritmo exato se mostram eficazes em uma comparação entre a solução alcançada sobre o número de nós explorados nas duas versões.

PALAVRAS-CHAVE: Problema da k-Partição de Números, Problema da Partição de Números, *Iterated Local Search*, *Complete Karmarkar-Karp Algorithm.*

Abstract

This dissertation presents a mathematical formulation and algorithms for the optimization version of the Multi-Way Number Partitioning Problem. The problem consists in distributing the elements of a given set into k disjoint subsets such that the sum of each subset elements fits in the shortest interval. The proposed mathematical formulation is based on two similar problems in the literature. The multiplicity of solutions of the integer programming model are eliminated by reducing the number of variables to a minimum. The NP-completeness of the problem studied is shown. Heuristic, approximate and exact algorithms of the literature for related problems are studied. The ILS meta-heuristic (Iterated Local Search) is applied to the solution generated by an approximate algorithm. The strategy is to apply greedy method in all phases of the algorithm, both in initialization and in the choice of ILS movements. It is proposed to improve the exact Branch-and-Bound methods present in the literature. This modification refines the limitations of the method using the relaxation of the proposed mathematical model and ILS. A search criterion is inserted so that the most promising nodes have priority. The results show the solution of ILS overcoming the solutions of constructive heuristics listed in the literature. The quality of the solution ILS is certified by exact methods, one of the literature and another with the improvements proposed in this work. The modifications of the exact algorithm are shown to be effective in comparing the ration between solution achieved and number of nodes exploited in the two versions.

KEY WORDS: *Multi-way Number Partitioning Problem*, *Number Partitioning Problem*, *Iterated Local Search*, *Complete Karmarkar-Karp Algorithm.*

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

Entende-se por partição de um conjunto X a uma coleção de subconjuntos, dois a dois disjuntos, cuja união forma X. Uma k-partição ocorre quando a partição tem exatamente k subconjuntos não vazios. Durante o decorrer do texto, os subconjuntos pertencentes à partição são denominados de partes.

O Problema da Partição de Números (*Two-Way Number Partitioning Problem* – TWNPP) já foi chamado de "*O mais fácil problema NP-completo*" por Hayes (2002), mas os problemas generalizados que derivam dele são muito mais complicados. Variações menos gerais desse problema se limitam a encontrar uma partição com partes de tamanho fixo, como o *3-Partition Problem*, encontrado em Joosten e Zantema (2013), ou mudam o objetivo original para um problema de escalonamento, como em Moffitt (2013).

Embora seja um dos 21 problemas de classe NP-completos discutidos por Karp (1972), existem algoritmos exatos para o TWNPP, como em Schroeppel e Shamir (1981), Horowitz e Sahni (1974) e Korf (1998)), como também muitos algoritmos aproximados aplicáveis ao problema, analisados em Gent e Walsh (1998) e Ausiello et al. (2003a).

Neste Capítulo introdutório apresenta-se, na Seção 1.1, a família de Problemas de Partição de Números. Os objetivos gerais e específicos desta dissertação são postos na Seção 1.6. Em seguida, apresenta-se, na Seção 1.7, a metodologia empregada para sua realização. Logo após, na Seção 1.8, uma justificativa para este estudo é incluída. Por fim, a Seção 1.9 mostra a organização geral da Dissertação e sua divisão por capítulos.

1.1 Definições Iniciais

Os conceitos de norma, diâmetro e bola são importantes para melhor compreensão da Seção 1.1 pois com eles é possível enunciar todos os problemas das Seções 1.2, 1.3, 1.4 e 1.5, mantendo a mesma natureza da função objetivo. Esses conceitos se encontram formalmente descritos em Lima (2004).

Definição 1.1.1 *Uma função* $\|.\|_p : \mathbb{R}^n \to \mathbb{R}^+$ *é uma norma de* $\mathbb{R}^n$ *se, para quaisquer* $x, y \in \mathbb{R}^n$ *e* $a \in \mathbb{R}$*:*

- $\|x\|_p \geq 0$

- $\|x\|_p = 0 \Leftrightarrow x = \vec{0}$
- $\|a.x\|_p = |a|.\|x\|_p$
- $\|x + y\|_p \leq \|x\|_p + \|y\|_p$

Nesse texto usa-se $p = \infty$, ou seja, a norma do máximo, que retorna a maior coordenada do vetor.

Definição 1.1.2 *Sejam dois vetores $x, y \in \mathbb{R}^n$. A distância entre x e y é dada por $d(x, y) = \|x - y\|_p$.*

Definição 1.1.3 *Seja um subconjunto $X \subseteq \mathbb{R}^n$. O diâmetro de X é $diam(X) = \max_{x_1, x_2 \in X}\{d(x_1, x_2)\}$.*

Definição 1.1.4 *Seja um ponto $v_0 \in \mathbb{R}^n$ e um número real positivo r. Uma bola de centro v_0 e raio r é o conjunto de pontos $v \in \mathbb{R}^n$ tais que $\|v - v_0\|_p \leq r$.*

Proposição 1.1.1 *O diâmetro de um conjunto é duas vezes o raio da menor bola que contém todos os elementos deste conjunto.*

Com o objetivo de apresentar o Problema da k-Partição de Números, nas seções seguintes mostra-se uma sequência de outros problemas diretamente relacionados com o mesmo. Nota-se que é possível enunciar, genericamente, todos os problemas da seguinte forma:

Definição 1.1.5 *Seja um conjunto de índices $I_n = \{1, 2, ..., n\}$, uma sequência de vetores $V = \{v_i \in \mathbb{Z}_+^m : i \in I_n\}$ e um inteiro k. O objetivo é encontrar uma k-partição de I_n de modo que os valores das somas dos elementos v_i de cada parte fiquem contidos dentro da menor bola possível.*

Nessa dissertação, usa-se $V^k = (V, k)$ denotando uma instância do Problema da k-Partição de Números, sendo V a sequência a ser particionada e k o número de partes não vazias. A expressão "particionar V" tem o mesmo sentido que particionar o conjunto de índices que indexam seus elementos. Em seguida, mostra-se um enunciado para cada versão desse problema, com exemplo associado logo em seguida.

1.2 Problema da Partição de Números

O Problema da Partição de Números (*Two-Way Number Partitioning Problem* – TWNPP) é a tarefa de minimizar a diferença das somas de dois subconjuntos disjuntos de uma sequência V. Essa é a versão mais simples e popular do problema, apresentada por Karp (1972) e tratada em diversos artigos, como Pedroso e Kubo (2010) e Gent e Walsh (1998). O tamanho da bola, nesse caso, é o comprimento do intervalo formado pelo resultado das duas somas. O Exemplo 1.2.1 mostra essa situação.

Exemplo 1.2.1 *A sequência* $V = \{87, 6, 5, 45, 34, 2, 24, 12, 7, 6, 54, 34\}$ *pode ser particionada em:*

$$\underbrace{\{87, 34, 24, 6, 5\}}_{\sum_{i\in A_1} v_i=156}, \underbrace{\{54, 45, 34, 12, 7, 6, 2\}}_{\sum_{i\in A_2} v_i=160}$$

O valor da função objetivo dessa partição é $160 - 156 = 4$*, mas a partição ótima é:*

$$\underbrace{\{87, 34, 24, 6, 5, 2\}}_{\sum_{i\in A_1} v_i=158}, \underbrace{\{54, 45, 34, 12, 7, 6\}}_{\sum_{i\in A_2} v_i=158}$$

com valor de função objetivo $158 - 158 = 0$.

O enunciado matemático do Problema da Partição de Números consiste em encontrar um conjunto $A \neq \emptyset$, tal que $A \subset I_n$, que minimize a função:

$$f(A) = \left| \sum_{i\in A} v_i - \sum_{i\in I_n \setminus A} v_i \right| \tag{1.1}$$

A partir desse problema, encontram-se duas generalizações distintas, abordadas nos tópicos a seguir.

1.3 Problema da k-Partição de Números

A primeira generalização do TWNPP é o Problema da k-Partição de Números (*Multi-way Number Partitioning Problem* – MWNPP), que expande o número de subconjuntos nos quais se pretende distribuir os elementos da sequência V.

Dada uma sequência numérica V, o objetivo é encontrar uma partição de dimensão k, de modo que as somas dos elementos de cada parte sejam as mais próximas possíveis umas das outras. Isso se resume a ter a parte de maior soma se aproximando da parte de menor soma tanto quanto for possível. Os textos que abordam essa versão são, dentre outros, Karmarkar e Karp (1982), Korf (2009), Korf et al. (2014). Um trabalho recente de forte interesse, pela sua completude, é Schreiber (2014).

O exemplo a seguir mostra uma situação desse problema.

Exemplo 1.3.1 *A sequência* $V = \{87, 6, 5, 45, 34, 2, 24, 12, 7, 6, 54, 34\}$ *pode ser particionada em* $k = 3$ *partes das seguintes maneiras:*

$$\underbrace{\{87, 12, 6\}}_{\sum_{i\in A_1} v_i=105}, \underbrace{\{45, 34, 6, 24\}}_{\sum_{i\in A_2} v_i=109}, \underbrace{\{5, 7, 2, 54, 34\}}_{\sum_{i\in A_3} v_i=102}$$

O valor da função objetivo dessa partição é $109 - 102 = 7$*, mas a partição ótima é:*

$$\underbrace{\{87, 12, 6\}}_{\sum_{i\in A_1} v_i=105}, \underbrace{\{45, 34, 2, 24\}}_{\sum_{i\in A_2} v_i=105}, \underbrace{\{5, 7, 6, 54, 34\}}_{\sum_{i\in A_3} v_i=106}$$

com valor de função objetivo $106 - 105 = 1$*. Observe que as partes possuem dimensões (número de elementos) diferentes.*

A entrada do problema é uma sequência V e um inteiro positivo k e a saída é uma partição de V, na forma $\{A_1, A_2, ..., A_k\}$. O objetivo é minimizar a função:

$$f(\{A_1, A_2, ..., A_k\}) = \max_j \left\{ \sum_{i \in A_j} v_i \right\} - \min_j \left\{ \sum_{i \in A_j} v_i \right\} \tag{1.2}$$

A função (1.2) é uma versão simplificada do diâmetro de um conjunto com k números reais, $\sum_{i \in A_j} v_i$, mostrada abaixo:

$$f(\{A_1, A_2, ..., A_k\}) = \max_{j' \neq j} \left\{ \sum_{i \in A_{j'}} v_i - \sum_{i \in A_j} v_i \right\} \tag{1.3}$$

1.4 Problema Multidimensional da Partição de Números

A segunda generalização do TWNPP é o Problema Multidimensional da Partição de Números (*Multidimensional Two-Way Number Partitioning Problem* – MDTWNPP). Neste problema, tem-se a dimensão dos elementos, $m > 1$, transformando uma sequência V de inteiros em uma sequência de vetores inteiros positivos $V = \{v_i \in \mathbb{Z}_+^m : \quad \forall i \in I_n\}$. Para definir este problema, faz-se necessário o uso de uma norma em $\mathbb{R}^m$.

Seu enunciado matemático, proposto por Kojić (2010), é muito parecido com o TWNPP, porém, após somar os elementos de cada um dos subconjuntos, tem-se um vetor, cuja norma será o valor do objetivo. A busca é um conjunto $A \subseteq I_n$ não vazio que minimize a função:

$$f(A) = \left\| \sum_{i \in A} v_i - \sum_{i \in I_n \setminus A} v_i \right\|_p \tag{1.4}$$

Quando $p = \infty$, a função $f(\cdot)$ retorna a maior coordenada do vetor $\sum_{i \in A} v_i - \sum_{i \in I_n \setminus A} v_i$, sendo equivalente à função objetivo proposta para o problema no artigo citado. Então, seja v_{il} um inteiro positivo, com i indexando o vetor v_i, variando de 1 até n, e l sendo a coordenada do vetor v_i, variando de 1 até m. O conjunto I_n é o conjunto dos índices i e a variável de decisão A é um subconjunto de I_n.

$$f(A) = \max_l \left\{ \sum_{i \in I_n \setminus A} v_{il} - \sum_{i \in A} v_{il} \right\} \tag{1.5}$$

Exemplo 1.4.1 *O conjunto*

$$V = \{(8,3,6,3), (8,3,12,3), (8,3,0,3), (17,5,10,5), (10,5,10,5), (10,5,10,3), (8,3,6,1), (5,3,6,3)\}$$

com $m = 4$ e $k = 2$ pode ser particionado como:

$$\{v_i \ : \ i \in A_1\} \ = \ \underbrace{\{(17,5,10,5),(8,3,0,3),(10,5,10,5),(10,5,10,3)\}}_{\sum_{i\in A_1} v_i=(45,18,30,16)},$$

$$\{v_i \ : \ i \in A_2\} \ = \ \underbrace{\{(8,3,6,3),(8,3,12,3),(8,3,6,1),(5,3,6,3)\}}_{\sum_{i\in A_2} v_i=(29,12,30,10)}$$

com um valor de função objetivo

$$\max\{|(45,18,30,16)-(29,12,30,10)|\} = \max\{|(16,6,0,6)|\} = 16$$

mas tem partição ótima dada por:

$$\{v_i \ : \ i \in A_1\} \ = \ \underbrace{\{(17,5,10,5),(10,5,10,5),(10,5,10,3)\}}_{\sum_{i\in A_1} v_i=(37,15,30,13)},$$

$$\{v_i \ : \ i \in A_2\} \ = \ \underbrace{\{(8,3,6,3),(8,3,12,3),(8,3,0,3),(8,3,6,1),(5,3,6,3)\}}_{\sum_{i\in A_2} v_i=(37,15,30,13)}$$

com valor de função objetivo:

$$\max\{|(37,15,30,13)-(37,15,30,13)|\} = \max\{|(0,0,0,0)|\} = 0$$

1.5 Problema Multidimensional da k-Partição de Números

O Problema Multidimensional da k-Partição de Números (*Multidimensional Multi-Way Number Partitioning Problem* – MDMWNPP) é uma generalização tanto do MWNPP quanto do MDTWNPP. Neste problema, uma sequência de vetores em $\mathbb{Z}_+^m$ deve ser dividida em k subconjuntos, de modo que o diâmetro desses k vetores resultantes seja o menor possível. O Exemplo 1.5.1 ilustra essa situação.

Exemplo 1.5.1 *Seja o conjunto:*

$$S \ = \ \{(37,15,30,13),(8,3,6,3),(8,3,12,3),(8,3,0,3),(17,5,10,5),\\ (10,5,10,5),(10,5,10,3),(8,3,6,1),(5,3,6,3)\}$$

Para $m = 4$ e $k = 3$, pode ser particionado em:

$$\{v_i \ : \ i \in A_1\} \ = \ \underbrace{\{(17,5,10,5),(10,5,10,5),(10,5,10,3)\}}_{\sum_{i\in A_1} v_i=(37,15,30,13)},$$

$$\{v_i \ : \ i \in A_2\} \ = \ \underbrace{\{(37,15,30,13),(8,3,0,3)\}}_{\sum_{i\in A_2} v_i=(45,18,30,16)},$$

$$\{v_i \ : \ i \in A_3\} \ = \ \underbrace{\{(8,3,6,3),(8,3,12,3),(8,3,6,1),(5,3,6,3)\}}_{\sum_{i\in A_3} v_i=(29,12,30,10)}$$

com valor de função objetivo dada por:

$$\begin{aligned}&\max\{|(45,18,30,16)-(29,12,30,10)|,|(37,15,30,13)-(29,12,30,10)|,\\&|(45,18,30,16)-(37,15,30,13)|\}\\&=\max\{|(16,6,0,6)|,|(8,3,0,3)|,|(8,3,0,3)|\}=16\end{aligned}$$

mas com partição ótima:

$$\{v_i \ : \ i\in A_1\} = \underbrace{\{(17,5,10,5),(10,5,10,5),(10,5,10,3)\}}_{\sum_{i\in A_1} v_i=(37,15,30,13)},$$

$$\{v_i \ : \ i\in A_2\} = \underbrace{\{(37,15,30,13)\}}_{\sum_{i\in A_2} v_i=(37,15,30,13)},$$

$$\{v_i \ : \ i\in A_3\} = \underbrace{\{(8,3,6,3),(8,3,12,3),(8,3,0,3),(8,3,6,1),(5,3,6,3)\}}_{\sum_{i\in A_3} v_i=(37,15,30,13)}$$

e com valor de função objetivo igual a 0*, pois todas as partes têm igual soma. Sendo assim, o diâmetro desse conjunto é a maior coordenada de* 3 *vetores nulos.*

Ou seja, o objetivo é equivalente a minimizar a maior distância entre dois de todos os vetores, dada pela expressão:

$$f(\{A_1,A_2,...,A_k\}) = \max_{(j',j)}\left\{\left\|\sum_{i\in A_{j'}} v_i - \sum_{i\in A_j} v_i\right\|_p\right\} \tag{1.6}$$

A expressão (1.6) pode ser utilizada como a função objetivo de qualquer um dos problemas enunciados acima, seja do Problema apresentado na Seção 1.2, seja do Problema apresentado na Seção 1.5 atual, sem qualquer inconsistência.

O trabalho de Pop e Matei (2013) adapta um outro modelo para o MDMWNPP, a partir da formulação matemática apresentada em Kojić (2010) para o MDTWNPP mostrada na Subseção 1.4.

Seja $v_i \in \mathbb{Z}_+^m$ e $S_j = \sum_{i\in A_j} v_i$. A função objetivo é a maior diferença entre duas coordenadas de $S_{j'}$ e S_j, para $j' \neq j$, na forma:

$$f(\{A_1,A_2,...,A_k\}) = \max_{l}\left\{\left|\max_{j'}\sum_{i\in A_{j'}} v_{il} - \min_{j}\sum_{i\in A_j} v_{il}\right|\right\} \tag{1.7}$$

Nesta expressão, j e j' indexam as partes e variam de 1 até k; i vai de 1 até n e indexa o vetor v dentro da parte A_j; l é a coordenada do vetor v e vai de 1 até m.

Note que, ao contrário de todos os enunciados até aqui, a expressão (1.7) não é igual à expressão (1.6) quando $p = \infty$. Esta formulação está fora do padrão da Definição 1.1.5.

A Figura 1.1 resume as relações entre os quatro problemas apresentados.

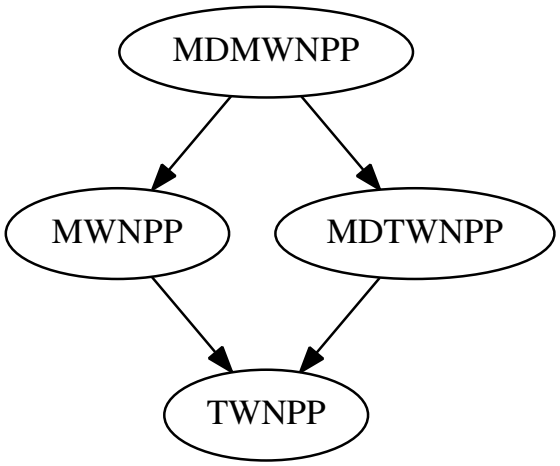

Figura 1.1: Relações entre os 4 problemas.

1.6 Objetivos

O objetivo geral deste trabalho é o estudo teórico e empírico do Problema da k-Partição de Números baseado na análise de suas soluções heurísticas e exatas.

Os objetivos específicos do presente trabalho são avaliar a aplicação de algoritmos heurísticos com movimentos "gulosos" de troca e realocação, comparando-os a heurísticas construtivas, testar suas possibilidades de aplicação como limitantes de algoritmos exatos de enumeração implícita e verificar se as heurísticas propostas geram limites superiores melhores que as da literatura.

1.7 Metodologia

Para alcançar os objetivos expostos na Seção 1.6, usam-se instâncias geradas aleatoriamente. A avaliação do desempenho dos algoritmos propostos será composta por análises descritivas, comparações de resultados obtidos por algoritmos da literatura e estratégias abordadas nos trabalhos correlatos, citados na Seção 2.4 do Capítulo 2.

Para alcançar esse objetivo, destaca-se a seguinte sequência metodológica:

(i) Realizar a pesquisa bibliográfica sobre o MWNPP e sobre as técnicas utilizadas para solucioná-lo mesmo em artigos que modificaram o objetivo original do problema.

(ii) Verificar outras formas de resolver o MWNPP usando algoritmos desenvolvidos para problemas relacionados.

(iii) Analisar os resultados obtidos com cada método desenvolvido.

(iv) Analisar os resultados de forma comparativa.

1.8 Justificativas

O MWNPP é um problema NP-completo facilmente adaptável às classes de problemas reais de corte e empacotamento. Métodos aplicáveis ao MWNPP podem ser aproveitados no estudo das soluções de outros problemas NP-completos. O MWNPP apresenta diversos algoritmos exatos baseados em métodos de enumeração implícita e programação dinâmica. Embora a complexidade desses algoritmos seja exponencial, na prática, eles são bem mais rápidos que o esperado para instâncias de tamanho razoável, de até 100 elementos. Esses fatos motivam a investigação de heurísticas simples e eficientes para aumentar a velocidade de convergência dos métodos de enumeração implícita aplicados ao MWNPP. Esse tipo de estudo, por um lado, tem foco em garantir a otimalidade da solução encontrada quando o método exato tem tempo o suficiente para finalizar sua busca, e por outro lado, reduzir o tempo necessário para sua convergência. Em outras palavras, espera-se que a técnica empregada apresente uma solução melhor em um período de tempo menor enquanto o método não convergir.

1.9 Organização da Dissertação

O restante do texto está organizado segundo os comentários em cada capítulo a seguir. No Capítulo 2, mostra-se o problema tratado com uma visão mais detalhada usando enunciado, modelo matemático proposto, uma visão teórica sobre como encontrar suas soluções e um histórico de trabalhos sobre temas relacionados. No Capítulo 3, apresentam-se alguns resultados relacionados a classe de complexidade do problema estudado. O Capítulo 4 contém algoritmos para problemas auxiliares aos métodos propostos do Capítulo 5. O Capítulo 5 apresenta algoritmos propostos, heurísticos e exatos implementados e testados. Apresentam-se os resultados de cada teste desse trabalho e uma análise dos mesmos no Capítulo 6. O Capítulo 7 finaliza a dissertação com uma discussão geral sobre o trabalho desenvolvido e uma apresentação de parâmetros para trabalhos futuros.

Capítulo 2

Caracterização do Problema

Este Capítulo apresenta o problema abordado nesta dissertação. Está organizado da seguinte forma: a Seção 2.1 apresenta a definição do problema e discute sua dificuldade combinatória. A Seção 2.2 introduz uma formulação matemática para o problema em Otimização Linear Inteira. A Seção 2.3 mostra uma solução teórica exata.

2.1 Definição do Problema da k-Partição de Números

O Problema da k-Partição de Números (*Multi-Way Number Partitioning Problem* - MWNPP), apresentado na Seção 1.3, é a versão abordada, originalmente, em Karmarkar e Karp (1982). Sua entrada é um conjunto V e a saída é uma partição de V com tamanho k.

Enunciado 2.1.1 *Seja $V = \{v_1, v_2, \ldots, v_n\}$ um conjunto de inteiros positivos v_l, $l = 1, \ldots, n$, e k um número inteiro positivo. O Problema da k-Partição de Números consiste em encontrar uma k-partição dos índices de V, na forma $\{A_1, A_2, \ldots, A_k\}$, que minimize a função $f(\cdot)$ definida por:*

$$f(\{A_1, A_2, \ldots, A_k\}) = \max_{j'} \left\{ \sum_{i \in A_{j'}} v_i \right\} - \min_{j} \left\{ \sum_{i \in A_j} v_i \right\} \tag{2.1}$$

A dificuldade combinatória desse problema, quando resolvido por uma enumeração ingênua das k-partições de V, é dada pelo Número de Stirling $S(n, k)$[1], que conta a quantidade de formas distintas de se repartir um conjunto de tamanho n em k partes. Encontra-se uma análise mais detalhada do Número de Stirling em Stanley (1997, pág. 80ff) e em Griffiths e Mezo (2010).

Defina $S(n-1, k)$ como o número de partições com k partes de um conjunto com $n-1$ elementos. Claro que o número de n-partições de um conjunto com n elementos é 1 (ou seja, $S(n, n) = 1$) e o número de 1-partição de qualquer conjunto é 1 (ou seja, $S(n, 1) = 1$). Se um n-ésimo elemento for adicionado ao conjunto, é possível descobrir $S(n, k)$ usando duas informações: $S(n-1, k-1)$ e $S(n-1, k)$.

[1] `http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html`

Inserindo o n-ésimo elemento como a k-ésima parte, existem $S(n-1,k-1)$ possibilidades. Inserindo o n-ésimo elemento em uma das k partes já existentes, contadas por $S(n-1,k)$, tem-se $k\ S(n-1,k)$ possibilidades:

$$S(n,k) = S(n-1,k-1)\ +\ k\ S(n-1,k) \tag{2.2}$$

Exemplo 2.1.1 *Seja a sequência de inteiros $V = \{1,2,3\}$, para $n = 3$, e considere $k = 2$. Assim, o Número de Stirling produz $S(3,2) = 3$. Desse modo, as partições são dadas por:*

$$\{1,2\},\{3\};$$
$$\{1,3\},\{2\};$$
$$\{2,3\},\{1\};$$

Seja a mesma sequência de inteiros $V = \{1,2,3\}$, para $n = 3$, e considere $k = 1$. Assim, o número de Stirling leva a $S(3,1) = 1$ e à partição:

$$\{1,2,3\}$$

Considere agora o conjunto de inteiros $V = \{1,2,3,4\}$, para $n = 4$, e seja $k = 2$. Desse modo, as partições são dadas por:

$$\{1,2,3\},\{4\};\{1,2\},\{3,4\};$$
$$\{1,2,4\},\{3\};\{1,3\},\{2,4\};$$
$$\{1,3,4\},\{2\};\{1,4\},\{2,3\};$$
$$\{2,3,4\},\{1\}$$

Ou seja, a partir da expressão (2.2), tem-se que:

$$S(4,2) = S(3,1) + 2.S(3,2) = 7$$

Ao usar um algoritmo ingênuo de enumeração, além de enumerar todas as partições de tamanho k, ainda é justo inserir um fator multiplicador $O(n)$, responsável pelo custo de se avaliar a função objetivo. No fim dos cálculos, a ordem de complexidade do algoritmo seria $O\left(n.S(n,k)\right)$.

O Número de Stirling não tem fórmula fechada. Uma outra alternativa para expressar essa quantidade, vista em Stanley (1997), é a expressão:

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n \tag{2.3}$$

Esse número supera as complexidades de ordem exponenciais quando o valor de k segue a sequência A024417[2]. Se $a(n) \in A024417$ então $S(n,a(n)) \geq S(n,k), \quad \forall k$. Esse fato pode ser conferido em A002870[3], a sequência do Maior Número de Stirling indexada em n. Se $a(n) \in A024417$, então $S(n,a(n)) \in A002870$. O gráfico abaixo mostra casos exponenciais em escala logarítmica comparados ao n-ésimo Maior Número de Stirling. Este gráfico é composto por retas sendo cortadas por uma função crescente.

[2] `http://oeis.org/A024417`
[3] `http://oeis.org/A002870`

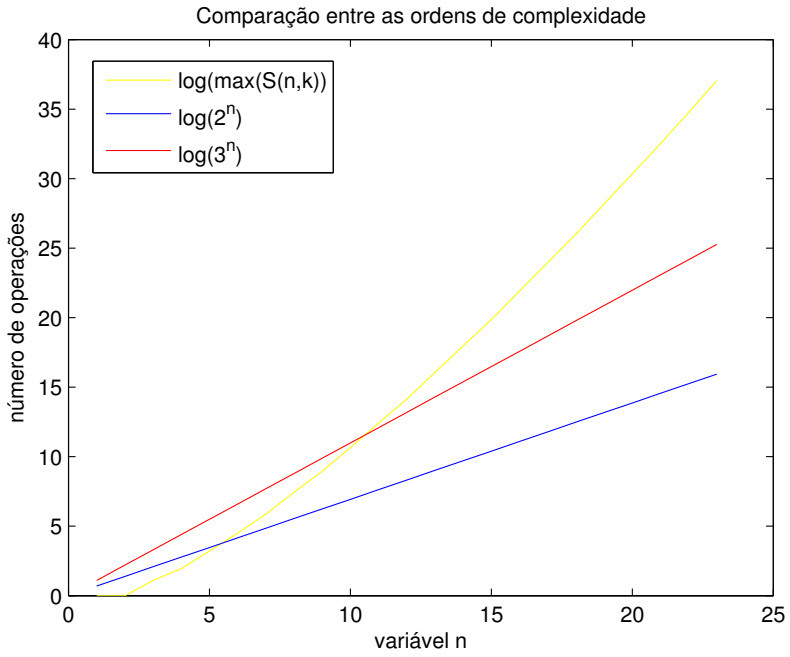

Figura 2.1: O Maior dos Números de Stirling vs Funções Exponenciais. Dados retirados de Sloane (1991).

A *On-Line Encyclopedia of Integer Sequences* (OEIS) é uma base de dados pública com variadas sequências de números inteiros, criada por Sloane (1991). Nela é possível buscar uma sequência numérica apenas digitando seus primeiros termos e, também, encontrar os principais artigos ligados à mesma. É a fonte das sequências A002870 e A024417.

A Figura 2.1 mostra que o Maior Número de Stirling domina assintoticamente funções exponenciais, ou seja, para todo $c > 1$, existe n suficientemente grande tal que $S(n, a(n)) > c^n$.

2.2 Formulação Matemática

Dentre os problemas de Otimização Inteira mais conhecidos da literatura, os mais próximos ao MWNPP são, certamente, o Problema do Empacotamento (*Bin Packing Problem* - BPP) e o Problema de Escalonamento em Máquinas Idênticas (*Identical Machine Scheduling Problem* - IMSP). Um modelo matemático para o MWNPP deve herdar características dos modelos de Otimização Inteira para IMSP e BPP, encontrados em Crescenzi et al. (1997) e Ausiello et al. (2003b). No texto a seguir, usa-se a notação $I_h = \{y \in \mathbb{Z} \ : \ 1 \leq y \leq h\}$, denotando o conjunto de todos os inteiros entre 1 e h, inclusive. A literatura apresenta os seguintes enunciados e formulações para esses dois problemas:

Problema do Empacotamento

Enunciado 2.2.1 *Problema do Empacotamento (BPP): Sejam uma sequência de itens* $\{a_i \in \mathbb{Z}_+ \ : \ i \in I_n\}$ *e pacotes de capacidade* $C \in \mathbb{Z}_+$*, determinar qual é o*

menor número de pacotes suficientes para empacotar todos os itens.

Para resolver esse problema, deve-se encontrar uma partição da sequência dada com o menor número de subconjuntos possível. A soma dos itens de cada um desses subconjuntos não pode ultrapassar C. A formulação matemática, encontrada em Ausiello et al. (2003b), está posta da seguinte maneira:

$$\min \quad \sum_{j=1}^{n} y_j \tag{2.4}$$

$$\text{suj. a} \quad \sum_{i=1}^{n} a_i x_{ji} \leq C y_j, \quad \forall j \in I_n \tag{2.5}$$

$$\sum_{j=1}^{n} x_{ji} = 1, \quad \forall i \in I_n \tag{2.6}$$

$$y_j \in \{0,1\}, \quad \forall j \in I_n \tag{2.7}$$

$$x_{ji} \in \{0,1\}, \quad \forall (j,i) \in I_n \times I_n \tag{2.8}$$

A função objetivo (2.4) mede a quantidade de pacotes usados na solução. As desigualdades (2.5) indicam que todo item deve estar em algum pacote ativo e que a capacidade máxima de cada pacote é C. As restrições (2.6) indicam que um item não pode estar em mais de um pacote. As expressões (2.7) mostram a natureza binária das variáveis y_j, de modo que se o pacote j está ativo, então $y_j = 1$; caso contrário, $y_j = 0$. Já as expressões (2.8) mostram a natureza binária das variáveis x_{ji}, de modo que, se o item i está no pacote j, então $x_{ji} = 1$; caso contrário, $x_{ji} = 0$.

Problema de Escalonamento em Máquinas Idênticas

Enunciado 2.2.2 *Problema de Escalonamento em Máquinas Idênticas (IMSP): dados uma sequência de tarefas $\{t_i \in \mathbb{Z}_+ \ : \ i \in I_n\}$ e m máquinas paralelas idênticas, determinar qual é o menor tempo suficiente para executar todas as tarefas nas m máquinas.*

Para resolver esse problema, deve-se encontrar uma m-partição, partição esta com exatamente m subconjuntos, da sequência dada. O subconjunto com a maior soma de tarefas deve ser minimizado. A formulação matemática, também encontrada em Ausiello et al. (2003b), é dada por:

$$\min \quad T \tag{2.9}$$

$$\text{suj. a} \quad 0 < \sum_{i=1}^{n} t_i x_{ji} \leq T, \quad \forall j \in I_m \tag{2.10}$$

$$\sum_{j=1}^{m} x_{ji} = 1, \quad \forall i \in I_n \tag{2.11}$$

$$T \in \mathbb{Z}_+ \tag{2.12}$$

$$x_{ji} \in \{0,1\}, \quad \forall (j,i) \in I_m \times I_n \tag{2.13}$$

A função objetivo (2.9) mede o tempo de finalização da máquina mais atarefada. As desigualdades (2.10) mostram que a variável T é um limitante máximo e que nenhuma máquina pode ficar sem tarefas. As restrições (2.11) indicam que uma tarefa i não pode estar em mais de uma máquina j. A expressão (2.12) mostra a natureza de T, limite superior do tempo de execução das tarefas em qualquer das m máquinas. As expressões 2.13 mostram a natureza das variáveis x_{ji}, de modo que, se a tarefa i está na máquina j, então $x_{ji} = 1$; caso contrário, $x_{ji} = 0$.

Análise e Discussão das Formulações do BPP e do IMSP

Esta análise tem o objetivo de apresentar melhorias aos modelos matemáticos discutidos até agora e é parte das contribuições efetivas desta dissertação. De maneira similar aos modelos matemáticos apresentados para os problemas IMSP e BPP, pode-se propor uma formulação matemática para o MWNPP, fiel ao Enunciado 2.1.1, como abaixo:

$$\begin{aligned}
\min \quad & t_2 - t_1 && (2.14)\\
\text{suj. a} \quad & t_1 \leq \sum_{i=1}^{n} v_i x_{ji} \leq t_2, \quad \forall j \in I_k && (2.15)\\
& \sum_{j=1}^{k} x_{ji} = 1, \quad \forall i \in I_n && (2.16)\\
& t_1, t_2 \in \mathbb{Z}_+ && (2.17)\\
& x_{ji} \in \{0,1\}, \quad \forall (j,i) \in I_k \times I_n && (2.18)
\end{aligned}$$

A relação de pertinência (2.18) indica que $x_{ji} = 1$ se o elemento de índice i do conjunto V pertence à parte de índice j; e $x_{ji} = 0$, em caso contrário. A relação (2.17) informa que os limitantes da maior e menor soma dos elementos das partes são sempre positivos. As equações indicadas pela expressão (2.16) garantem que as partes são disjuntas e que todos os elementos de V estão alocados em alguma parte. As inequações (2.15) mostram que o modelo garante que cada parte é não vazia, pois $t_1 > 0$, por efeito de (2.17) e, também, que estão todas contidas no intervalo $[t_1, t_2]$. A expressão (2.14) é o tamanho do intervalo que contém todas as k somas dos elementos das partes e o objetivo de minimização.

Essas formulações têm sobras de variáveis. Verifica-se esse fato por meio de um pequeno exemplo genérico para o IMSP, em que $m = 3$ e $n = 5$. Para qualquer valor que se queira atribuir às tarefas da sequência $(t_1, t_2, t_3, t_4, t_5)$, as matrizes a seguir são soluções factíveis:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Todas essas 6 matrizes representam a mesma solução para o problema. Da forma como está modelado matematicamente, tem-se uma multiplicidade $m!$ para cada solução factível do IMSP. Resolve-se esse problema com a retirada de $\frac{m^2-m}{2}$ variáveis do tipo x_{ji}. Por exemplo, o triângulo inferior à direita. Assim, as soluções mostradas se tornam:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 0 & & \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & \\ 1 & 0 & 1 & & \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & \\ 0 & 0 & 0 & & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \\ 0 & 1 & 0 & & \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & \\ 1 & 0 & 1 & & \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & \\ 0 & 1 & 0 & & \end{pmatrix}$$

Agora, apenas a segunda matriz é factível pelo critério de soma das colunas da igualdade, conforme as expressões (2.11). Isso ocorre porque a ordem das máquinas não altera o argumento ótimo. A restrição capaz de descrever esse fato em uma linha é $x_{ji} = 0, \quad \forall (j,i) : j + i \geq n + 2$. De maneira análoga, aplica-se ao BPP e ao MWNPP esta mesma discussão de multiplicidade de soluções.

Formulação Matemática do MWNPP

O modelo matemático de Otimização Linear Inteira proposto nesse trabalho, como contribuição efetiva, usa duas variáveis inteiras para representar o limite inferior, t_1, e superior, t_2, para a soma dos elementos de todas as k partes. As demais variáveis x_{ij} são binárias e informam se um elemento está ou não em uma parte.

$$\begin{aligned}
\min \quad & t_2 - t_1 && (2.19)\\
\text{suj. a} \quad & t_1 \leq \sum_{i=1}^{n} v_i x_{ji} \leq t_2, \quad \forall j \in I_k && (2.20)\\
& \sum_{j=1}^{k} x_{ji} = 1, \quad \forall i \in I_n && (2.21)\\
& x_{ji} = 0, \quad \forall (i,j) : i + j \geq n + 2 && (2.22)\\
& t_1, t_2 \in \mathbb{Z}_+ && (2.23)\\
& x_{ji} \in \{0,1\}, \quad \forall (j,i) \in I_k \times I_n && (2.24)
\end{aligned}$$

A relação de pertinência (2.24) indica que $x_{ji} = 1$ se o elemento de índice i do conjunto V pertence à parte de índice j, em caso contrário, $x_{ji} = 0$. A relação (2.23) informa que os limitantes da maior e menor soma dos elementos das partes é sempre positiva. As igualdades (2.22) indicam que a ordem dos subconjuntos não altera o argumento. As n equações indicadas pela expressão (2.21) garantem que as partes são disjuntas e que todos os elementos de V estão alocados em alguma parte, já que o problema só está bem definido quando $n > k$. As k inequações (2.20) mostram que o modelo garante que cada parte é não vazia, pois $t_1 > 0$, por efeito de (2.23) e, também, que estão todas contidas no intervalo $[t_1, t_2]$. Por fim, a expressão (2.19) é

o tamanho do intervalo que contém todas as k somas dos elementos das partes e o objetivo de minimização.

A restrição (2.22) é uma forma explícita de explicar uma propriedade do problema em apenas uma linha, mas deve aparecer de forma implícita no modelo, caso o objetivo seja usá-lo em algum algoritmo ou em um "*solver*" comercial de otimização.

$$\min \quad t_2 - t_1 \tag{2.25}$$

$$\text{suj. a} \quad t_1 \leq \sum_{i=1}^{n-j+1} v_i x_{ji} \leq t_2, \quad \forall j \in I_k \tag{2.26}$$

$$\sum_{j=1}^{\min\{n-i+1,k\}} x_{ji} = 1, \quad \forall i \in I_n \tag{2.27}$$

$$t_1, t_2 \in \mathbb{Z}_+ \tag{2.28}$$

$$x_{ji} \in \{0,1\}, \quad \forall (j,i) : i + j \leq n + 1 \tag{2.29}$$

Este modelo é equivalente ao anterior, mas define apenas as variáveis de decisão fora do triângulo inferior à direita.

2.3 Solução via Algoritmo Ingênuo

Uma abordagem mais algorítmica do problema sugere a construção de um algoritmo polinomial para o MWNPP, usando um algoritmo hipotético que retorna sim (1) ou não (0) para uma pergunta específica sobre a entrada. Esse algoritmo é denominado Oráculo e é denotado por $Or(V, V')$, sendo V e V' duas instâncias de tamanho n e $n-1$, respectivamente, do MWNPP.

A pergunta em questão é:

Pergunta 2.3.1 *As entradas V e V' têm o mesmo valor ótimo de função objetivo ?*

O Exemplo 2.3.1 mostra uma situação associada ao uso desse Oráculo.

Exemplo 2.3.1 *Suponha que $k = 3$, $V = \{1,2,3,5,6\}$ e $V' = \{1,2,3,4\}$. O resultado de:*

$$Or(V, V') = Or(\{1,2,3,5,6\}, \{1,2,3,4\}) = 1$$

é (SIM), pois a 3-partição ótima de V pode ser $\{1,2,3\}, \{5\}, \{6\}$ ou $\{2,3\}, \{1,5\}, \{6\}$, ambas com valor de função objetivo igual a 1, assim como V', que tem 3-partição ótima $\{1,2\}, \{3\}, \{4\}$, com valor de função objetivo igual a 1. □

A ideia central dessa técnica é decidir se dois elementos distintos da sequência V pertencem ou não à mesma parte, com a troca de dois elementos da sequência pela soma dos mesmos. Sempre que essa troca não mudar o valor ótimo, significa que os dois valores substituídos pertencem à mesma parte.

Exemplo 2.3.2 *Considere que $V = \{1,2,3,5,6\}$ e $k = 3$ e considere a notação $V^k = (V, k)$, indicando uma instância do problema MWNPP. Considere, então, uma*

nova instância $V^{k'}$, na forma $V' = \{1+2, 3, 5, 6\}$ e $k' = 3$. Fazendo a chamada do Oráculo sobre V^k e $V^{k'}$, encontra-se que 1 e 2 estão contidos na mesma parte, quando $Or(V^k, V^{k'}) = 1$, e que estão em partes distintas, em caso contrário. Nesse exemplo, é claro que $Or((\{1,2,3,5,6\},3),\ (\{3,3,5,6\},3)) = 1$ (veja no Exemplo 2.3.1).

Algoritmo 2.1: Algoritmo exato hipotético

```
Entrada: Or() e um instância V^k
Saída: {A_1, A_2, ..., A_k}
início
    j = 1;
    h = 1;
    enquanto V ≠ ∅ faça
        insere(A_j, v_h);
        para v_i ∈ V − {v_h} faça
            V^{k'} = V^k;
            remove(V', v_i);
            insere(V', v_i + v_h);
            se Or(V^k, V^{k'}) = 1 então
                insere(A_j, v_i);
                V^k = V^{k'};
            fim
        fim
        j = j + 1;
        h = frente(V^k);
    fim
    retorna {A_1, A_2, ..., A_k};
fim
```

Observe que, no Algoritmo 1, V^k contém a sequência V. O funcionamento do algoritmo é simples e direto. Primeiro, o elemento v_1 é retirado de V e vai para a parte A_1, sendo o elemento líder da primeira parte. Em seguida, para cada $v_i \in V - \{v_1\}$, o elemento v_i é trocado por $v_1 + v_i$, formando uma nova instância $V^{k'} = (V_i, k)$ sendo $V_i = V \cup \{v_i + v_1\} - \{v_1, v_i\}$. Executa-se $Or(V^k, V^{k'})$ e, sempre que a resposta do Oráculo for positiva, retira-se v_i de V, armazenando-o em A_1 e V é trocado por V_i. Repete-se o procedimento até $V = \varnothing$, $k-1$ vezes.

No pior caso, este algoritmo faz $O(n^2)$ chamadas ao Oráculo, pois compara o elemento líder de A_j com cada um dos restantes em V.

2.4 Revisão Bibliográfica

Nesta Seção apresentam-se trabalhos correlatos à discussão da família de Problemas de Partição de Números disponíveis na literatura. Prioriza-se um conjunto de artigos e livros mais associados às ideias discutidas no Capítulo 5. Na Subseção 2.4.1, descrevem-se as publicações relacionadas ao Problema da k-Partição de Números (MWNPP) desde sua formulação inicial até trabalhos mais recentes. Na Subseção 2.4.2 apresenta-se um panorama temporal, classificando as técnicas adotadas nos trabalhos correlatos por autor e ano.

2.4.1 Uma Visão Histórica

Existe uma ampla literatura sobre a solução do Problema da Partição de Números (TWNPP) e suas variações. Ele foi listado de modo formal no trabalho de Karp (1972) como um dos 6 problemas NP-completos básicos e, mais tarde, por Garey e Johnson (1979). Ambos demonstram uma série de equivalências entre o TWNPP e outros problemas NP-completos.

O MWNPP aparece explicitamente em um artigo sobre a análise de uma heurística construtiva denominada *Differencing Method*, mais conhecida como Heurística de Karmarkar-Karp, proposta por Karmarkar e Karp (1982). Esta heurística está focada na ideia de dividir os maiores números em partes distintas, inserindo as diferenças entre os elementos retirados do conjunto dos não-alocados. Este é o artigo mais citado sobre o MWNPP até hoje. Os autores demonstram resultados teóricos e práticos sobre a eficiência da heurística proposta e sobre a dificuldade das instâncias do problema.

Mais tarde, Michiels et al. (2003) demonstram uma razão de aproximação menor ou igual a $\frac{4}{3} - \frac{1}{3(k-1)}$, similar aquela encontrada pelo algoritmo *Longest Processing Time* (LPT), descrito na Seção 4.1, para a Heurística de Karmarkar-Karp. Este resultado impõe um intervalo de erro limitado aos resultados obtidos com a Heurística de Karmarkar-Karp.

Existe, também, um ramo de estudo do TWNPP no escopo do interesse da Física Estatística. Muitos destes trabalhos são motivados pela análise do problema feita por Karmarkar e Karp (1982). Esses trabalhos pesquisam uma forma de classificar as instâncias do TWNPP e seus problemas associados, buscando uma metodologia que garanta um conjunto livre de partições perfeitas. Partições perfeitas são aquelas que têm um valor ótimo de função objetivo igual a 0 ou 1. Percebe-se que uma instância com grande variedade de partições perfeitas poderia ser resolvida até mesmo por uma enumeração ingênua, pois não existem valores de função objetivo menores para se encontrar. Os trabalhos mais importantes que apresentam essa linha de interesse são:

- Gent e Walsh (1995), mostrando a transição de fases para a identificar as características de uma instância livre de partições perfeitas;
- Mertens (2006), que promove um avanço no trabalho de Gent e Walsh (1995), mostrando uma nova fórmula que classifica instâncias quase certamente livres de partições perfeitas; e
- Ferreira (2001), que apresenta uma nova equação para o valor esperado da função objetivo do TWNPP.

Trabalhos como Gent e Walsh (1998) e Berretta e Moscato (1999) mostram que o TWNPP é um problema muito difícil para metaheurísticas de uso geral, como Algoritmos Genéticos, *Simulated Annealing* e outros. Em muitos casos, esses métodos perdem em tempo e desempenho para a Heurística de Karmarkar-Karp e até mesmo para um algoritmo guloso como LPT. Textos mais recentes, como Kojić (2010) e Pop e Matei (2013), usam várias classes de metaheurísticas em versões de maior complexidade da família de Problemas da Partição de Números, como o MDTWNPP

e o MDMWNPP, mas também não mostram resultados muito significativos, exceto pela formulação matemática.

Tanto para o TWNPP quanto para o MWNPP, sempre houve a possibilidade de usar os algoritmos de Horowitz e Sahni (1974) e Schroeppel e Shamir (1981), fazendo a conversão para o Problema da Mochila, mas o espaço de memória requerido é inviável para problemas de maior cardinalidade. Uma nova abordagem surgiu quando Korf (1998) propôs um algoritmo exato, fazendo um procedimento *Backtrack* em heurísticas construtivas, como o LPT e a Heurística de Karmarkar-Karp, na forma *Complete Greedy Algorithm* e *Complete Karmarkar-Karp Algorithm*, respectivamente.

A primeira melhoria desses trabalhos acontece com o algoritmo *Recursive Number Partitioning*, proposto por Korf (2009). A segunda melhoria é a contribuição de Pedroso e Kubo (2010), em que é proposta uma nova estrutura de dados aplicada ao trabalho de Korf (2009), agilizando a busca na Árvore de Karmarkar-Karp. A discussão sobre a função objetivo do MWNPP empreendida por Korf em Korf (2010) mostra as diferenças dos objetivos de seus artigos e do trabalho seminal de Narendra Karmarkar e Richard M. Karp em 1982 (Karmarkar e Karp, 1982). Por meio de sucessivas conversões do MWNPP de uma partição de tamanho $k-1$ para k, Moffitt (2013) propõe um algoritmo baseado em programação dinâmica.

Atualmente, o estado da arte para a solução desta família de problemas está posto no algoritmo *Sequential Number Partitioning*, apresentado no artigo Korf et al. (2014), e no algoritmo *Cached Iterative Weakening*, apresentado em Schreiber e Korf (2014). Uma análise completa e de grande interesse sobre esses algoritmos, com todos os seus detalhes, encontra-se em Schreiber (2014).

2.4.2 Panorama das Publicações mais Relevantes

A Tabela 2.1 apresenta a lista, em ordem cronológica, das publicações mais relevantes sobre o MWNPP. Nesta tabela, tem-se que:

- Autor: indica a publicação;
- Problema: problema tratado na publicação;
- Objetivos: são tratados os seguintes casos:
  - MmaxP - minimizar a parte com maior soma;
  - MminP - maximizar a parte com menor soma;
  - MdianP - minimizar o diâmetro das somas das partes;
  - Mpitem - maximizar o valor dos itens na mochila.
- Método: A - aproximado, E - exato, H - heurístico, MH - metaheurístico, Est - estatístico;
- Complexidade: $O(2^{\frac{n}{2}})$, $O(n.log(n))$, $O(2^n)$, $O(k^n)$, $O(\frac{k^n}{k!})$ sendo ou não amortizadas;
- Gerais: # - nulo, ? - incerto, ! - variado.

	Autor	Problema	Objetivo	Método	Complexidade
1	(Karp, 1972)	Coletânea	#	#	#
2	(Horowitz e Sahni, 1974)	Mochila	Mpitem	E	$O(2^{\frac{n}{2}})$
3	(Garey e Johnson, 1979)	Coletânea	#	#	#
4	(Schroeppel e Shamir, 1981)	Mochila	Mpitem	E	$O(2^{\frac{n}{2}})$
5	(Karmarkar e Karp, 1982)	MWNPP	MdiaP	H	$O(n.log(n))$
6	(Gent e Walsh, 1995)	TWNPP	MmaxP, MminP, MdiaP	Est	#
7	(Gent e Walsh, 1998)	TWNPP	MmaxP, MminP, MdiaP	Est, MH, H	(!)
8	(Korf, 1998)	TWNPP	MmaxP, MminP, MdiaP	E	$O(2^n)$, amortizado
9	(Berretta e Moscato, 1999)	TWNPP	MmaxP, MminP, MdiaP	MH, H	(!)
10	(Ferreira, 2001)	TWNPP	MmaxP, MminP, MdiaP	Est	#
11	(Hayes, 2002)	TWNPP	MmaxP, MminP, MdiaP	Est	#
12	(Mertens, 2006)	TWNPP	MmaxP, MminP, MdiaP	Est	#
13	(Michiels et al., 2003)	MWNPP	MmaxP	A	$O(n.log(n))$
14	(Korf, 2009)	MWNPP	MmaxP	E	$O(k^n)$, amortizado
15	(Pedroso e Kubo, 2010)	TWNPP	MmaxP, MminP, MdiaP	E	$O(2^n)$, amortizado
16	(Korf, 2010)	MWNPP	MmaxP, MminP, MdiaP	#	#
17	(Kojić, 2010)	MDTWNPP	MdianP	MH	#
18	(Pop e Matei, 2013)	MDMWNPP	MdianP (?)	MH	#
19	(Moffitt, 2013)	MWNPP	MmaxP	E	$O(k^n)$, amortizado
20	(Korf et al., 2014)	MWNPP	MmaxP	E, H	$O(\frac{k^n}{k!})$, amortizado
21	(Schreiber e Korf, 2014)	MWNPP	MmaxP	E, H	$O(\frac{k^n}{k!})$, amortizado
22	(Schreiber, 2014)	MWNPP	MminP	E, H	(!)

Tabela 2.1: Quadro das publicações mais relevantes.

Os artigos em que o item "Problema" está classificado como "Coletânea" são abordagens teóricas sobre problemas NP-completos. Nestes casos, os itens "Objetivo", "Método" e "Complexidade" estão marcados com "#".

Nos artigos em que o item "Método" está marcado com apenas "Est", não há nenhum algoritmo proposto, mas, sim, uma análise da dificuldade das instâncias.

O termo "amortizado" significa que o algoritmo faz cortes em suas ramificações e quase nunca recai no pior caso.

Capítulo 3

Análise de Complexidade do MWNPP

Este Capítulo tem a finalidade de cobrir uma lacuna teórica acerca do MWNPP, apresentando uma demonstração de sua NP-completude. Para esta análise, considera-se o MWNPP em sua versão de decisão mostrada no Enunciado 3.2.2. Esta restrição não trará grandes perdas de generalidade. Na Seção 3.1, apresentam-se conceitos preliminares, como definições, proposições e teoremas relacionados ao estudo das classes de complexidade. A Seção 3.2 usa a aplicação dos conceitos discutidos na Seção 3.1 ao MWNPP para classificá-lo como um problema NP-completo. A Seção 3.3 apresenta generalizações dentro da família de Problemas de Partição de Números e uma discussão acerca dos resultados encontrados.

3.1 Conceitos Preliminares

Todas as proposições, definições, teoremas e corolários dessa seção estão presentes em Ausiello et al. (2003b), Sipser (2006) e Korte et al. (2012).

Seja $\mathcal{A}$ um problema computacional. Denota-se $\mathcal{I}(\mathcal{A})$ como o conjunto de todas as instâncias do problema $\mathcal{A}$.

Definição 3.1.1 *Um problema $\mathcal{A}$ não é mais difícil que um problema $\mathcal{B}$ se uma solução para $\mathcal{B}$ implica em uma solução para $\mathcal{A}$ com perda de desempenho polinomial relativa à eficiência da solução de $\mathcal{B}$.*

Definição 3.1.2 *Sejam $\mathcal{A}$ e $\mathcal{B}$ problemas computacionais de decisão. Uma redução em tempo polinomial de $\mathcal{A}$ para $\mathcal{B}$ é um algoritmo M que computa uma função $M : \mathcal{I}(\mathcal{A}) \to \mathcal{I}(\mathcal{B})$ tal que, para toda instância $I \in \mathcal{I}(\mathcal{A})$ de $\mathcal{A}$, a instância $M(I) \in \mathcal{I}(\mathcal{B})$ do problema $\mathcal{B}$ tem a mesma resposta.*

A notação para descrever a relação da Definição 3.1.2 é $\mathcal{A} \leq_p \mathcal{B}$. Esta é uma relação de ordem e deve ser lida na forma "$\mathcal{A}$ se reduz a $\mathcal{B}$".

Se há uma redução de $\mathcal{A}$ para $\mathcal{B}$ e $M_{\mathcal{B}}$ é um algoritmo que resolve $\mathcal{B}$, então o Algoritmo 3.1 resolve $\mathcal{A}$.

Se $M_{\mathcal{B}}$ é um algoritmo de tempo $p(n)$ e M é um algoritmo de tempo $q(n)$, então o Aloritmo $M_{\mathcal{A}}$, mostrado no Algoritmo 3.1, tem consumo de tempo $p(q(n))$. Nota-se

Algoritmo 3.1: Algoritmo $M_{\mathcal{A}}(I)$

Entrada: Uma Instância I para o problema $\mathcal{A}$ e uma função M
Saída: Solução da instância I

```
1 início
2 |  retorna M_B(M(I));
3 fim
```

que $|M(I)| = O(q(|I|))$, ou seja, a saída de M tem tamanho polinomial na dimensão de I. Assim, tem-se as seguintes definições:

Definição 3.1.3 *Um problema $\mathcal{A} \in P$ se pode ser resolvido por um algoritmo eficiente (com complexidade polinomial), sendo P o conjunto dos problemas de decisão que podem ser resolvidos em tempo polinomial.*

Definição 3.1.4 *Um problema $\mathcal{A} \in NP$ se sua solução pode ser verificada por um algoritmo eficiente, sendo NP o conjunto dos problemas de decisão que são verificáveis em tempo polinomial.*

Teorema 3.1.1 *Se $\mathcal{A} \leq_p \mathcal{B}$ e $\mathcal{B} \in P$ então $\mathcal{A} \in P$.*

Prova: Seja $M(I)$ uma função computável em tempo polinomial, que leva uma instância I do problema $\mathcal{A}$ para uma instância do problema $\mathcal{B}$. Então, o Algoritmo 3.1 resolve o problema $\mathcal{A}$ em tempo polinomial. ■

Definição 3.1.5 *Um problema $\mathcal{A}$ é NP-difícil se todos os problemas em NP não são mais difíceis do que $\mathcal{A}$ no seguinte sentido:*

$$\mathcal{B} \leq_p \mathcal{A}, \quad \forall \mathcal{B} \in NP. \tag{3.1}$$

Suponha que o problema $\mathcal{A}$ tenha um algoritmo $M_{\mathcal{A}}$ que o resolve em tempo $O(n^d)$ e seja M a redução de $\mathcal{B} \in NP$ para $\mathcal{A}$, que consome tempo $O(n^b)$ para algum natural b. Então, se I é uma instância de tamanho n para $\mathcal{B}$, $M(I)$ é uma instância de tamanho $O(n^k)$ para $\mathcal{A}$, cuja resposta $M_{\mathcal{A}}(M(I))$ é computada em tempo $O((n^d)^b) \sim O(n^{d+b})$. Portanto, tem-se o seguinte teorema:

Teorema 3.1.2 *Se algum problema NP-difícil tem algoritmo eficiente, então todos os problemas em NP também têm, isto é, P=NP.*

Prova: Considere que $\mathcal{A}$ é NP-difícil. Para cada $\mathcal{B} \in NP$, existe uma função computável em tempo polinomial $M_{\mathcal{B}\to\mathcal{A}}(I)$ que transforma uma instância I do problema $\mathcal{B}$ para o problema $\mathcal{A}$. ■

Os Teoremas 3.1.1 e 3.1.2 têm demonstrações muito similares, pois afirmam duas questões muito parecidas sobre as implicações da redução de um problema ao outro.

Definição 3.1.6 *Um problema $\mathcal{A}$ é NP-completo se*

(i) $\mathcal{A} \in NP$;

(ii) $\mathcal{A}$ *é NP-difícil.*

Uma consequência da Definição 3.1.6 é que todos os problemas NP-completos se reduzem uns nos outros, ou seja, são todos equivalentes por uma simples transformação polinomial. Um corolário mais completo desse fato segue abaixo.

Corolário 3.1.1 *Se algum problema NP-completo tem algoritmo eficiente, então* $P = NP$.

Prova: Sabe-se que $P \subseteq NP$ pois todo problema polinomial é verificável em tempo polinomial. Resta fazer a inclusão $NP \subseteq P$, supondo que algum problema NP-completo admite um algoritmo polinomial. Como todo problema em NP se reduz a um problema NP-completo pela Definição 3.1.6, um algoritmo polinomial de algum problema desse grupo produz um algoritmo eficiente para todos os problemas NP. ■

A Proposição 3.1.1 pede uma suposição de que $P \neq NP$ de forma implícita.

Proposição 3.1.1 *Se algum problema NP-completo não tem algoritmo eficiente, então nenhum outro problema NP-completo tem, isto é* $P \cap$ *NP-completo* $= \emptyset$.

Prova: Todos os problemas NP-completos são equivalentes. Se um deles, $\mathcal{A} \in$ NP-completo, não tem um algoritmo eficiente, então nenhum problema $\mathcal{C}$ tal que $\mathcal{C} \leq_p \mathcal{A}$ pode ter, pois esse único algoritmo eficiente $M_{\mathcal{C}}$ serviria para resolver $\mathcal{A}$. Logo, se existe $\mathcal{A} \in NP-completo$ e $\mathcal{A} \notin P$, implica que $P \cap NP-completo = \emptyset$. ■

A proposição abaixo é muito útil para classificar um problema como NP-completo. Seu uso na Seção 3.2 será explícito com o problema $\mathcal{A}$ sendo o TWNPP.

Proposição 3.1.2 *Se* $\mathcal{A}$ *é NP-completo e* $\mathcal{A} \leq_p \mathcal{B}$, *então* $\mathcal{B}$ *é NP-difícil.*

Prova: Sendo $\mathcal{A}$ um problema NP-completo, então todo problema $\mathcal{C}$ em NP se reduz a $\mathcal{A}$. Se $\mathcal{A} \leq_p \mathcal{B}$, todo problema NP se reduz a $\mathcal{B}$ pela transitividade da relação de redução. Portanto, $\mathcal{B}$ é NP-difícil segundo à Definição 3.1.5. ■

3.2 TWNPP se reduz ao MWNPP

Os dois enunciados abaixo definem o MWNPP-*decisão*. O segundo enunciado é mais geral que o primeiro.

Enunciado 3.2.1 *(MWNPP1_ d) - Dada uma sequência numérica* V. *Existe uma partição dos índices de* V, *na forma* $\{A_1, A_2, ..., A_k\}$ *tal que a função objetivo (1.2), mostrada na Seção 1.3, tenha valor nulo, ou seja:*

$$f(\{A_1, A_2, ..., A_k\}) = \max_{j'} \left\{ \sum_{i \in A_{j'}} v_i \right\} - \min_{j} \left\{ \sum_{i \in A_j} v_i \right\} = 0 \ ?$$

Enunciado 3.2.2 *(MWNPP2_ d) - Dada uma sequência numérica V e uma constante C. Existe uma partição dos índices de V, na forma $\{A_1, A_2, ..., A_k\}$ tal que a função objetivo 1.2 seja menor que C, ou seja:*

$$f(\{A_1, A_2, ..., A_k\}) = \max_{j'} \left\{ \sum_{i \in A_{j'}} v_i \right\} - \min_{j} \left\{ \sum_{x \in A_j} v_i \right\} \leq C \ ?$$

Seja TWNPP_d a versão de decisão do problema TWNPP. Então, o seguinte resultado relaciona TWNPP_d e MWNPP2_d.

Teorema 3.2.1 *$TWNPP\_d \leq_p MWNPP2\_d$*

Prova: Seja I^2 uma instância para o TWNPP_d e $Q = \sum_{i \in I^2} v_i$. Considera-se a função de redução M:

$$M(I^2) = I^2 \cup \underbrace{\left\{ \frac{Q}{2}, ..., \frac{Q}{2} \right\}}_{k-2 \ vezes} = I^k. \tag{3.2}$$

Seja I^2 uma sequência cujos índices podem ser 2-particionados em $\{A_1, A_2\}$. Tem-se uma k-partição, dada por $\{A'_1, A'_2, ..., A'_k\}$, para os índices da sequência associada à instância $M(I^2)$, sendo:

$$A'_1 = A_1 \tag{3.3}$$
$$A'_2 = A_2 \tag{3.4}$$
$$A'_j = \{j\}, \quad \forall j \in \{3, ..., k\} \tag{3.5}$$

Para provar que a redução funciona, deve-se mostrar que uma resposta SIM para o problema de decisão MWNPP2_d ocorre se, e somente se, a resposta para o problema de decisão TWNPP_d é SIM. Em outras palavras:

$$f(\{A'_1, A'_2, ..., A'_k\}) \leq C \Leftrightarrow f(\{A_1, A_2\}) \leq C. \tag{3.6}$$

Para isso, prova-se que $f(\{A'_1, A'_2, ..., A'_k\}) = f(\{A_1, A_2\})$. Se a função M tem tal propriedade, pode-se construir um algoritmo similar ao Algoritmo 3.1, sendo $\mathcal{B} = MWNPP2\_d$ e $\mathcal{A} = TWNPP\_d$. Sem perda de generalidade, considera-se

$$\sum_{i \in A_1} v_i \geq \sum_{i \in A_2} v_i$$

Sabe-se que

$$\begin{aligned} Q &= \sum_{i \in I^2} v_i \\ &= \sum_{i \in A_1 \cup A_2} v_i \\ &= \sum_{i \in A_1} v_i + \sum_{i \in A_2} v_i \end{aligned}$$

Portanto, conclui-se que:

$$\sum_{i \in A_2} v_i \leq \frac{Q}{2} \leq \sum_{i \in A_1} v_i \tag{3.7}$$

pois a média de dois números sempre está no intervalo entre esses dois números. Nesse caso, a parte de maior soma de $\{A'_1, A'_2, ..., A'_k\}$ é a mesma parte de maior soma de $\{A_1, A_2\}$ e a parte de menor soma de $\{A'_1, A'_2, ..., A'_k\}$ é a mesma parte de menor soma de $\{A_1, A_2\}$, ou seja, $A'_1 = A_1$ e $A'_k = A_2$. Portanto, ambas as partições têm o mesmo valor de função objetivo. ■

Essa redução tem custo $O(n)$ para calcular a soma Q dos elementos da sequência da instância I^2. A saída da função é uma instância I^k, cuja sequência de inteiros associada tem dimensão $n + k - 2$. Sendo $k < n$, pode-se afirmar, também, que a redução é polinomial no tamanho da entrada. Assim, tem-se o seguinte corolário.

Corolário 3.2.1 *O MWNPP2_d é NP-difícil.*

Teorema 3.2.2 *MWNPP2_d é NP-completo.*

Prova: O Teorema 3.2.1 prova que o problema de decisão MWNPP2_d é um problema NP-difícil. Resta mostrar que MWNPP2_d também está em NP, exibindo uma verificação em tempo polinomial como na Definição 3.1.4 e incluí-lo na classe de problemas NP-completos, de acordo com a Definição 3.1.6. Para este fim, considera-se uma k-partição $\{A_1, ..., A_k\}$ dos índices de I^k. Avaliar a função objetivo consiste em: (i) calcular um vetor de dimensão k, no qual cada elemento j é dado pela soma dos elementos da parte A_j; (ii) encontrar o maior e o menor elemento deste vetor; e, finalmente, (iii) computar a diferença destes dois valores. Este procedimento tem custo $O(n)$. O cálculo permite decidir se o resultado é menor ou igual a um valor C dado. ■

A redução $TWNPP\_d \leq_p (MWNPP1\_d)$ é análoga à demonstração acima. Por outro lado, ao dissociar o valor k da instância do MWNPP1 e considerar problemas específicos com k fixo como o Problema da K_1-Partição, nomeado como B_1, e Problema K_2-Partição, denominado B_2, mostra-se que $B_1 \leq_p B_2$ sempre que $K_1 \leq K_2$ com a seguinte função

$$M : \mathcal{I}(B_1) \rightarrow \mathcal{I}(B_2) \tag{3.8}$$

sendo:

$$M(I) = I \ \cup \underbrace{\left\{ \frac{Q}{K_1}, ..., \frac{Q}{K_1} \right\}}_{K_2 - K_1 \ \ vezes} \tag{3.9}$$

3.3 Análise da versão de Otimização do MWNPP

Nesta Seção discutem-se os aspectos essenciais que mostram que a redução de problemas de decisão não é uma grande restrição teórica, já que, na prática, a maioria dos problemas de interesse são de busca e otimização.

Algoritmo 3.2: Busca pelo valor mínimo usando o MWNPP2_d

Entrada: I^k, $decide()$
Saída: obj

```
1  início
2      c_1 = (1/k) Σ_{i∈I^k} v_i;
3      c_2 = 0;
4      enquanto (c_1 − c_2)/2 > 1 faça
5          obj = (c_1 + c_2)/2;
6          se decide(I^k, obj) então
7              c_1 = obj;
8          fim
9          senão
10             c_2 = obj;
11         fim
12     fim
13     retorna obj;
14 fim
```

A função de redução M da Seção 3.2 também pode ser utlizada para o MWNPP em sua versão de otimização. Esse fato tem relação direta com a possibilidade de aplicação de um algoritmo para o MWNPP2_d à versão de otimização do MWNPP. Pode-se aplicar uma busca binária no MWNPP2_d, apresentado no Enunciado 3.2.2, e encontrar o valor mínimo da função objetivo (1.2).

Suponha que, no Algoritmo 3.2, a função $decide(I^k, C)$ retorna 1 se existe uma partição com valor de função objetivo menor que C e 0, em caso contrário, tendo um custo computacional $O(1)$.

Usando o Algoritmo 3.2, com custo computacional $O(log(C))$, determina-se o valor mínimo da função objetivo, mas não o argumento ótimo associado à este. Para descobrir a partição associada ao valor mínimo retornado pelo Algoritmo 3.2, deve-se usar o Algoritmo 1, trocando o Oráculo $O(I^k, I^{k'})$ por um outro, na forma:

$$Or(I^k, obj^*) = decide(I^k, obj^*)$$

sendo obj^* a saída do Algoritmo 3.2.

A definição de um problema de otimização da classe NP pode ser encontrada em (Ausiello et al., 1995) e (Berretta e Moscato, 1999) e é apresentada a seguir.

Definição 3.3.1 *Um problema de otimização $\mathcal{A}$ é NP se:*

(i) $I \in \mathcal{I}(\mathcal{A})$ *é verificável em tempo polinomial;*

(ii) Uma solução factível para $\mathcal{A}$ é verificável em tempo polinomial;

(iii) A função objetivo é computável em tempo polinomial;

(iv) A meta do problema é a minimização ou maximização da função objetivo.

O MWNPP se encaixa nessa definição, pois suas instâncias podem ser identificadas em tempo polinomial, como exige o item (i), uma vez que são sequências de inteiros positivos; esse procedimento de verificação tem custo $O(n)$, sendo n o tamanho da sequência. As soluções factíveis são verificáveis em tempo polinomial, como exige o item (ii), pois são k-partições de índices dos elementos da sequência dada, e esta verificação tem custo $O(n^2)$. Conforme conclui-se a partir do Teorema 3.2.2, a função objetivo dada pela Expressão (1.2) é computável em $O(n)$. Por fim, a meta do problema é a minimização dessa função objetivo.

Portanto, conclui-se que o MWNPP, em sua versão de otimização, é um problema NP-completo segundo a Definição 3.3.1.

Capítulo 4

Fundamentação Teórica

Este Capítulo apresenta os fundamentos básicos relacionados ao tema desta dissertação no sentido de explorar técnicas de solução do problema abordado. A Seção 4.1 apresenta o Algoritmo (LPT). A Seção 4.2 mostra o Algoritmo 0/1-*Interchange*, ambos desenvolvidos para Problemas de Escalonamento de Máquinas. A Seção 4.3 discute a relação do Problema da k-Partição de Números com o Problema da Mochila. A Seção 4.4 apresenta a versão geral da metaheurística *Iterated Local Search* (ILS). A Seção 4.5 introduz a Heurística de Karmarkar-Karp e sua adaptação ao Problema da k-Partição de Números. Por fim, a Seção 4.6 exibe o funcionamento do método exatos *Branch-and-Bound* e sua aplicação ao MWNPP.

4.1 Algoritmo *Longest Processing Time first* (LPT)

Proposto por Graham (1966, 1969), o algoritmo guloso *Longest Processing Time first* (LPT) foi desenvolvido para o Problema de Partição Mínima e, também, para o Problema de Escalonamento de Tarefas. Estes problemas se difereciam do Problema da k-Partição de Números (MWNPP) pela função objetivo, como comenta Korf (2010).

Este algoritmo garante um resultado com razão de aproximação menor que $\frac{4}{3} - \frac{1}{3k}$ do ótimo global em relação à função objetivo $\max\limits_{1 \leq i \leq k} \left\{ \sum\limits_{i \in A_i} v_i \right\}$. Isso significa que, se $OPT(I^k)$ é o valor ótimo da função objetivo do problema e $LPT(I^k)$ o valor de função objetivo encontrado pelo algoritmo LPT para uma instância I^k, a razão entre estes valores é tal que:

$$\frac{LPT(I^k)}{OPT(I^k)} \leq \frac{4}{3} - \frac{1}{3k} \tag{4.1}$$

A demonstração desta razão de aproximação se encontra em Ausiello et al. (2003a).

Exemplo 4.1.1 *O resultado ótimo do Problema da k-Partição de Números para a instância:*

$$S = \{8, 7, 6, 5, 4\}$$

e $k = 2$ *é*

$$\{8, 7\} \qquad e \qquad \{6, 5, 4\}$$

A parte com a maior soma é $\max\{8+7,\ 6+5+4\} = 15$. *Substituindo* $k = 2$ *na desigualdade (4.1), tem-se:*

$$\left(\frac{4}{3} - \frac{1}{6}\right)\ 15 = \frac{7}{6}\ 15 = \frac{35}{2}$$

Portanto, a resposta do algoritmo (LPT) sempre é uma partição em que a maior soma é menor que $35/2$, *como, por exemplo:*

$$\{8,5,4\} \qquad e \qquad \{7,6\}$$

cuja maior soma é 17, *resultado encontrado pelo (LPT), mas nunca um valor* 18, *como a partição abaixo:*

$$\{8,4\} \qquad e \qquad \{7,6,5\}$$

□

O algoritmo consiste em ordenar os elementos da sequência V em ordem não crescente e alocar os números, sempre inserindo o novo elemento na parte de menor soma e, em caso de empate, desempata-se arbitrariamente. Este algoritmo tem um custo computacional $O(n \log n)$, para ordenar a entrada, e um custo $O(n)$, para alocar os elementos nas k partes. Finalmente, a complexidade do método é $O(n \log n)$.

O Algoritmo *Longest Processing Time first* (LPT) está apresentado no Algoritmo 4.1.

Algoritmo 4.1: *Longest Processing Time* para k máquinas Graham (1966, 1969)

Entrada: Uma Sequência V de números inteiros e um inteiro k.
Saída: Partição dos índices V, $\{A_1, A_2, ..., A_k\}$, com tamanho k.

```
início
    para j ∈ {1, ..., k} faça
        A_j = ∅;
        L_j = 0;
    fim
    Ordene V em ordem decrescente;
    para i ∈ I^k faça
        l = arg min_j L;
        A_l = A_l ∪ i;
        L_l = L_l + v_i;
    fim
    retorna {A_1, A_2, ..., A_k}
fim
```

4.2 Algoritmo 0/1-*Interchange*

O Algoritmo 0/1-*Interchange*, proposto por Finn e Horowitz (1979), também é um método para o Problema de Escalonamento em k máquinas idênticas. Partindo de uma distribuição inicial das tarefas, o algoritmo ordena os processadores em ordem não decrescente.

Considere A_j o conjunto de tarefas alocadas no processador j. Considere também que $L(A_j)$ seja o instante de finalização das tarefas alocadas no processador j. Assim, defina $d_i = L(A_1) - L(A_k)$ como a diferença entre o instante de finalização do processador mais carregado e o instante de finalização do processador menos carregado na iteração i. Enquanto existir uma tarefa $p \in A_1 \ : \ p < d_i$, esta tarefa será realocada para o processador k, pois este é o de menor carregamento. Este algoritmo tem uma complexidade de tempo computacional da ordem $O(n.log(k))$, como demonstrado por Langston (1982)

O Algoritmo 0/1-*Interchange* está apresentado no Algoritmo 4.2.

Algoritmo 4.2: Algoritmo 0/1-*Interchange* (Finn e Horowitz, 1979)

Entrada: k-Partição $\{A_1, A_2, ..., A_k\}$ do conjunto V
Saída: k-Partição do conjunto V, na forma $\{A_1, A_2, ..., A_k\}$

```
início
    Ordene os tempos de cada máquina L(A_1) ≥ ... ≥ L(A_k);
    d = L(A_1) − L(A_k);
    enquanto ∃q ∈ A_1 : L(q) < d faça
        remove(A_1, q);
        insere(A_k, q);
        L(A_1) = L(A_1) − L(q);
        L(A_k) = L(A_k) + L(q);
        d = L(A_1) − L(A_k);
    fim
    retorna {A_1, A_2, ..., A_k};
fim
```

4.3 Problema da Mochila e Soma dos Subconjuntos

O Problema da Mochila é um dos mais estudados em Otimização Combinatória e costuma ser um subproblema muito comum em vários trabalhos. Todas as variações do Problema da Mochila são NP-completos, mas admitem uma solução pseudo-polinomial, como pode ser encontrado em Garey e Johnson (1979) e Ausiello et al. (2003a).

Definição 4.3.1 *Seja $W \in \mathbb{N}$ e um conjunto de itens $I = \{1, 2, ..., n\}$, tal que cada elemento de I indexe um peso w_i e um valor p_i. Encontre um subconjunto de itens que maximize o valor da mochila, respeitando sua capacidade W de peso.* □

A formulação matemática para esse problema é:

$$\max \quad \sum_{j=1}^{n} p_j x_j \tag{4.2}$$

$$\text{s.a} \quad \sum_{j=1}^{n} w_i x_j \leq W \tag{4.3}$$

$$x_j \in \{0, 1\} \tag{4.4}$$

Algoritmo 4.3: *subsetsum*(): algoritmo exato para Soma de Subconjuntos (Ausiello et al., 2003a)

Entrada: Vetor v de tamanho n e uma capacidade W.
Saída: Subconjunto V de índices de v.

```
1  início
2      Inicialize uma matriz M ∈ ℜ^{(n+1)×(W+1)} = 0;
3      para i=n até 1 faça
4          para w=0 até W faça
5              se i > n ou w = 0 então
6                  M(i, w) = 0;
7              fim
8              se w_i > w então
9                  M(i, w) = M(i + 1, w);
10             fim
11             se max{M(i + 1, w), M(i + 1, w − w_i) + w_i} então
12                 M(i, w) = 0;
13             fim
14         fim
15     fim
16     retorna M(1, W)
17 fim
```

O Problema da Soma dos Subconjuntos é um caso particular do Problema da Mochila, caracterizado por $\forall i \in I \ : \ w_i = p_i$. O algoritmo pseudo-polinomial para resolvê-lo tem custo $O(nW)$ e, por isso, costuma ser o mais indicado quando o conjunto de itens I possui grande dimensão, mas o número W tem poucos algarismos. Caso contrário, os algoritmos exatos mais eficientes são os propostos por Horowitz e Sahni (1974) e Schroeppel e Shamir (1981). A ideia da solução, neste caso é a seguinte:

- Caso o item i faça parte da solução ótima, considerando os itens $\{i, ..., n\}$, tem-se uma solução com valor p_i a mais do que a solução ótima para os itens $\{i+1, ..., n\}$, com capacidade restante $W - w_i$. Esta ideia corresponde a fazer $x_i = 1$, gerando o subproblema:

$$p_i + \max \left\{ \sum_{j=i+1}^{n} p_j x_j \mid \sum_{j=i+1}^{n} w_j x_j \leq w - w_i, \quad x_j \in \{0, 1\} \right\} \tag{4.5}$$

- Caso contrário, tem-se um valor correspondente à solução ótima para itens $\{i + 1, ..., n\}$, com capacidade W. Esta ideia corresponde a fazer $x_i = 0$, gerando o subproblema:

$$\max \left\{ \sum_{j=i+1}^{n} p_j x_j \mid \sum_{j=i+1}^{n} w_j x_j \leq w, \quad x_j \in \{0, 1\} \right\} \tag{4.6}$$

Assim, seja $M(i, w)$ o valor da solução ótima para os itens $\{i, ..., n\}$ e capacidade W, na forma:

$$M(i, w) = \max\{M(i + 1, w), M(i + 1, w - w_i) + p_i\} \tag{4.7}$$

e o valor ótimo do problema será $M(1, W)$.

O Algoritmo 4.3 formaliza o procedimento. O Algoritmo 4.3 cria uma matriz M de dimensão $n \times W$, no qual os elementos $M_{i,j}$ são os valores ótimos do Problema da Mochila com capacidade j usando os i primeiros itens de V.

4.4 *Iterated Local Search* (ILS)

A meta-heurística *Iterated Local Search* (ILS) parte do pressuposto que os ótimos locais podem ser totalmente percorridos por um tipo de movimento. Assim como todas as permutações de n números podem ser expressas como um produto de transposições, o movimento definido para o ILS deve conseguir gerar todo o espaço de busca do problema abordado.

O ILS, proposto em Lourenço et al. (2003) funciona, genericamente, com alguns passos fundamentais, conforme mostra o Algoritmo 4.4.

Gera-se uma solução inicial para o problema. Faz-se uma busca local da melhor solução na vizinhança. A vizinhança de x é tudo que pode ser alcançado partindo de x com um único movimento. Perturba-se o espaço de soluções atuais quando este já estiver estagnado. A perturbação deve levar a próxima solução para fora da vizinhança da solução atual. Aceita-se ou não a solução atual de acordo com um critério, geralmente baseado no valor da função objetivo. O procedimento termina com o número máximo de iterações.

O Algoritmo 4.4 apresenta a estrutura típica do ILS.

Algoritmo 4.4: Algoritmo ILS.

Entrada: Gerador de solução inicial.
Saída: Solução do espaço de busca.

```
1  início
2      s_0 = SolucaoInicial ;
3      s = BuscaLocal(S_0);
4      i = 0;
5      enquanto i < Itermax faça
6          i = i + 1;
7          s' = Perturba(s);
8          s'' = BuscaLocal(s');
9          s = CriteriodeAceitacao(s, s', s'');
10     fim
11     retorna s
12 fim
```

Na Seção 5.2 demonstram-se as principais funções do ILS de forma específica para o MWNPP.

4.5 Heurística de Karmarkar-Karp para $k \geq 2$

A Heurística de Karmarkar-Karp, introduzida por Karmarkar e Karp (1982), é um método construtivo para o TWNPP que pode ser adaptado para o MWNPP. Esse algoritmo escolhe alocar os dois maiores elementos de uma sequência V em

partes distintas e, em seguida, adiciona a diferença entre os dois maiores elementos no conjunto como um novo elemento, descontando o erro produzido pela alocação.

Algoritmo 4.5: Heurística de Karmarkar-Karp com $k = 2$ (Karmarkar e Karp, 1982)

Entrada: Sequência V.
Saída: Valor inteiro correspondendo à menor diferença entre as partes.

```
1 início
2     enquanto |V| ≠ 1 faça
3         s1 = removemax(V);
4         s2 = removemax(V);
5         c = s1 − s2;
6         insereordenado(V, c);
7     fim
8     retorna v[0]
9 fim
```

O pseudo-código da Heurística de Karmarkar-Karp está no Algoritmo 4.5. O Exemplo 4.5.1 mostra uma aplicação dessa heurística.

Exemplo 4.5.1 *Seja* $V = \{8, 7, 6, 5, 4\}$ *e* $k = 2$. *Então:*

Iteração 1: $\{\mathbf{8}, \mathbf{7}, 6, 5, 4\} \Rightarrow \{6, 5, 4, 1\}$

Iteração 2: $\{\mathbf{6}, \mathbf{5}, 4, 1\} \Rightarrow \{4, 1, 1\}$

Iteração 3: $\{\mathbf{4}, \mathbf{1}, 1\} \Rightarrow \{3, 1\}$

Iteração 4: $\{\mathbf{3}, \mathbf{1}\} \Rightarrow \{2\}$

O procedimento retorna o valor de função objetivo igual a 2 *e a partição* $\{A_1, A_2\} = \{(8, 6), (7, 5, 4)\}$. *É importante observar que este resultado não se constitui na solução ótima do problema para esta instância.*

Para uma implementação de baixo custo computacional, usando uma estrutura de dados `max-heap`, a complexidade desse algoritmo é $O(n \log n)$.

A adaptação desse algoritmo para o MWNPP consiste em substituir a operação de diferença entre os dois maiores elementos do conjunto pelo Algoritmo 4.6 e seguir a mesma lógica de funcionamento do Exemplo 4.5.1. A diferença é que os elementos são vetores de dimensão k e não escalares, como no caso do Exemplo 4.5.1. Logo, a estrutura de dados `max-heap` deve operá-los em ordem lexicográfica.

Algoritmo 4.6: $opera(x, y, k)$; Modificação na Heurística de Karmarkar-Karp para a solução do MWNPP.

Entrada: Dois vetores x e y ordenados e um inteiro k.
Saída: Um vetor ordenado de tamanho k.

```
1  início
2      para i = 0 até n − 1 faça
3          s_i = x_i + y_{n−i};
4      fim
5      ordene s em ordem não crescente;
6      para i = 0 até n − 1 faça
7          s_i = s_i + s_{n−1};
8      fim
9      retorna s
10 fim
```

Algoritmo 4.7: Heurística de Karmarkar-Karp para $k > 2$

Entrada: Um sequência V e um inteiro k.
Saída: Valor inteiro correspondendo à menor diferença entre as partes.

```
1  início
2      enquanto |V| ≠ 1 faça
3          s_1 = removemax(V);
4          s_2 = removemax(V);
5          c = opera(s_1, s_2, k);
6          insereordenado(V, c);
7      fim
8      retorna v[0]
9  fim
```

O pseudo-código usado para resolver o Exemplo 4.5.2 está no Algoritmo 4.7. A única mudança em relação à Heurística de Karmarkar-Karp original é a troca de $c = s_1 - s_2$ por $c = opera(s_1, s_2, k)$.

O Exemplo 4.5.2 mostra o funcionamento da Heurística de Karmarkar-Karp para $k > 2$, fazendo cada iteração em duas ou 3 fases para demonstrar a generalização desta Heurística.

Exemplo 4.5.2 *Seja* $V = \{8, 7, 6, 5, 4\}$ *e* $k = 3$. *Então:*

Iteração 1: $\{(\mathbf{8}, \mathbf{0}, \mathbf{0}), (\mathbf{7}, \mathbf{0}, \mathbf{0}), (6, 0, 0), (5, 0, 0), (4, 0, 0)\} \Rightarrow$
$\{(8, 7, 0), (6, 0, 0), (5, 0, 0), (4, 0, 0)\}$

Iteração 2: $\{(\mathbf{8}, \mathbf{7}, \mathbf{0}), (\mathbf{6}, \mathbf{0}, \mathbf{0}), (5, 0, 0), (4, 0, 0)\} \Rightarrow \{(8, 7, 6), (5, 0, 0), (4, 0, 0)\} \Rightarrow$
$\{(5, 0, 0), (4, 0, 0), (2, 1, 0)\}$

Iteração 3: $\{(\mathbf{5}, \mathbf{0}, \mathbf{0}), (\mathbf{4}, \mathbf{0}, \mathbf{0}), (2, 1, 0)\} \Rightarrow \{(5, 4, 0), (2, 1, 0)\}$

Iteração 4: $\{(\mathbf{5}, \mathbf{4}, \mathbf{0}), (\mathbf{2}, \mathbf{1}, \mathbf{0})\}\} \Rightarrow \{(5, 5, 3)\} \Rightarrow \{(3, 3, 0)\}$

O procedimento retorna um valor de função objetivo igual a 3 *e uma partição* $\{8\}, \{7, 4\}, \{6, 5\}$

4.6 Método de *Branch and Bound*

O método de *Branch-and-Bound* (Método de Ramificação e Poda), proposto por Land e Doig (1960), é um algoritmo que consiste no desenvolvimento de uma enumeração inteligente dos pontos candidatos à solução de um problema de otimização. A ramificação consiste em particionar o espaço de soluções, gerando subproblemas associados ao original, e a poda avalia a otimalidade da solução, usando limitantes calculados em cada subproblema gerado na enumeração dos candidatos, os quais também permitem a eliminação de candidatos não promissores.

Considere o problema

$$z = \min\{f(x) \quad : \quad x \in D\} \tag{4.8}$$

Seja $D = \{D_1, D_2, ..., D_k\}$ uma decomposição de D em k conjuntos menores e seja

$$z_j = \min\{f(x) \quad : \quad x \in D_j\}, \quad \forall j \in I_k = \{1, 2, \ldots, k\} \tag{4.9}$$

Então, a solução ótima do problema (4.8) é:

$$z = \min_j z_j \ , \quad j \in I_k \tag{4.10}$$

A representação dessa decomposição é feita através de uma árvore de enumeração. Cada nó representa um subconjunto do espaço de soluções. A raiz da árvore representa todo o conjunto D, ou seja, todo o espaço de soluções do problema.

Sejam, então, UB_j e LB_j, respectivamente, um limite superior e um limite inferior para o subproblema z_j . Então $UB = \min_j UB_j$ é um limite superior em z. Primeiramente, encontram-se os limitantes superior e inferior da raiz da árvore de enumeração. Caso a solução ótima da relaxação seja factível quanto à integralidade, o algoritmo termina, pois foi encontrada a solução ótima z. Caso contrário, realiza-se uma operação de ramificação. Decompõe-se o espaço de soluções D em dois ou mais subconjuntos, na forma $\{D_1, D_2, ..., D_k\}$ que originam novos nós da árvore de enumeração.

A técnica de *Branch-and-Bound* evita a criação de toda a árvore de enumeração, usando em sua ramificação critérios de poda. A poda de um nó evita ramificações não promissoras. Em um problema de minimização, um nó pode ser podado por:

(a) Critério (i): infactibilidade,

(b) Critério (ii): limitante inferior do subproblema associado ao nó supera o ínfimo dos limitantes superiores,

(c) Critério (iii): otimalidade local ou global.

O Critério (ii) merece uma atenção especial. Pode-se podar um nó j tal que $LB_j > UB$, porque a solução associada ao valor UB, ínfimo dos limites superiores, sempre é factível quanto à integralidade. Mais geralmente, a solução associada à qualquer UB_j é sempre factível, mas LB_j é uma relaxação e, possivelmente, só terá soluções factíveis nas últimas ramificações da árvore de enumeração ou no Critério de poda (iii), quando $UB_j = LB_j$.

Nós não podados e não ramificados recebem o nome de nós ativos. Os nós ativos são os únicos que podem gerar um limitante superior menor para o problema. A otimalidade é detectada pelo algoritmo quando não há mais nós ativos. Sendo assim, podemos inferir que, quanto melhor forem os limitantes, melhor será o desempenho de um algoritmo *Branch-and-Bound*. O Algoritmo 4.8 apresenta um pseudo-código do método *Branch-and-Bound*.

Algoritmo 4.8: Método de *Branch-and-Bound*.

```
   Entrada: lower(), upper(), D, f()
   Saída: z*
1  início
2      Criar uma fila Q;
3      UB = upper(f, D);
4      LB = lower(f, D);
5      No = [LB, UB];
6      OPT = true;
7      enquanto |Q| > 0 ou OPT == true faça
8          No_i = remove(Q);
9          se LB_i > UB então
10             continue;
11         fim
12         se LB_i == UB então
13             se UB_i > UB então
14                 UB = UB_i;
15             fim
16             continue;
17         fim
18         senão
19             {D_1, D_2, ..., D_k} = branch(D);
20             para j = 1 até k faça
21                 UB_j = upper(f, D_j);
22                 LB_j = lower(f, D_j);
23                 No_j = [LB_j, UB_j];
24                 insere(Q, No_j, prioridade(No_j));
25                 se UB_j > UB então
26                     UB = UB_j;
27                 fim
28             fim
29         fim
30         se tempo > limite então
31             OPT = false;
32             |Q| = 0;
33         fim
34     fim
35     retorna UB
36 fim
```

O passo da ramificação da função *prioridade* mostrado na linha 24 diz respeito ao processo de escolha da ordem dos nós a serem ramificados. Existem diversas estratégias de seleção:

(i) Busca em largura;

(ii) Busca em profundidade;

(iii) Busca prioritária pelo nó com maior valor de *prioridade*(),

dentre outras.

A Busca em largura pode exigir mais memória do que o esperado, porém, é a mais usada para problemas com árvore de busca k-ária. A Busca em profundidade se mostra viável, pois sempre exige uma memória limitada polinomialmente. Outra vantagem é que, a cada nível de profundidade atingido pelos nós resolução da relaxação, se aproxima mais da solução inteira. A Busca prioritária tem, como objetivo, explorar o nó que possui o melhor limitante primeiro.

Estas variações do Método *Branch-and-Bound* geram algoritmos diversos, como o mostrado na Seção seguinte.

4.7 *Complete Greedy Algorithm* (CGA)

Um algoritmo do tipo *Branch-and-Bound* aplicado ao MWNPP é o *Complete Greedy Algorithm* (CGA), proposto em (Korf, 1998). Este algoritmo usa uma enumeração clássica de k-partições, baseada em uma construção similar à recorrência dos Números de Stirling. A Figura 4.1 mostra um exemplo de uma enumeração de partições aplicada ao CGA.

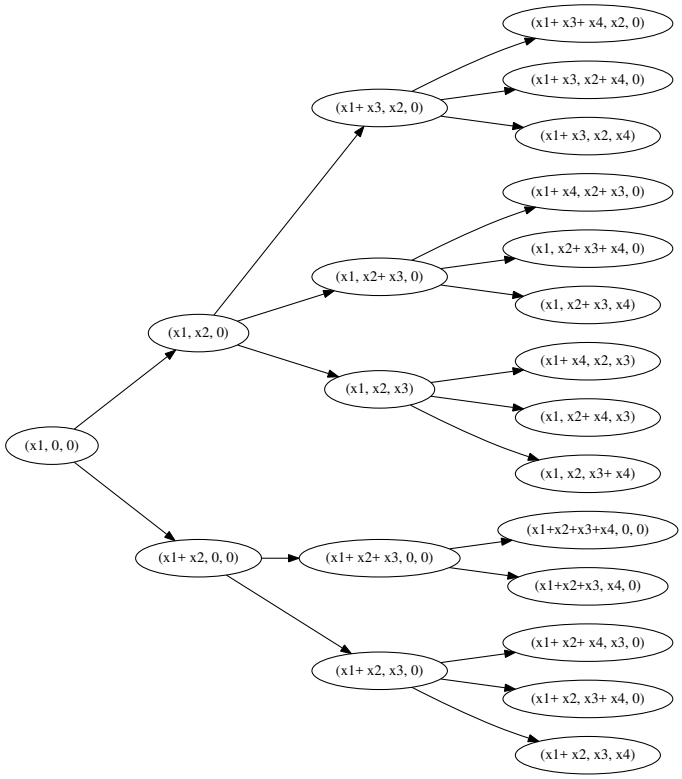

Figura 4.1: Exemplo de árvore de busca completa para o MWNPP: $|V| = 4$ e $k = 3$.

Observe que, pela Figura 4.1, o método parece impraticável, pois há 22 nós para uma instância de dimensão 4 e considerando $k = 3$, ou seja, uma instância de pequena dimensão. Mas algumas boas ideias podem melhorar seu desempenho, conforme discutido a seguir.

Existem alguns nós folhas infactíveis e isso induz a inserir uma regra de poda. Na prática, esta poda é suprimida pelo corte por limitante, equivalente ao Critério

(ii) no caso do Método de *Branch-and-Bound*. A melhor ideia, nesse caso, é impor um critério para que nós com h partes vazias no nível $n-h$ da árvore só façam um único filho, alocando o elemento v_{n-h+1} sempre em uma parte vazia. Quando um nó entra nesse critério, seu valor de limitante superior UB não se altera mais.

A Figura 4.2 mostra um exemplo de uma árvore de busca completa, enumerando todas as k-partições de uma sequência com nós livres de infactibilidade. Nota-se que o número de folhas dessa árvore é exatamente $S(4,3) = 6$.

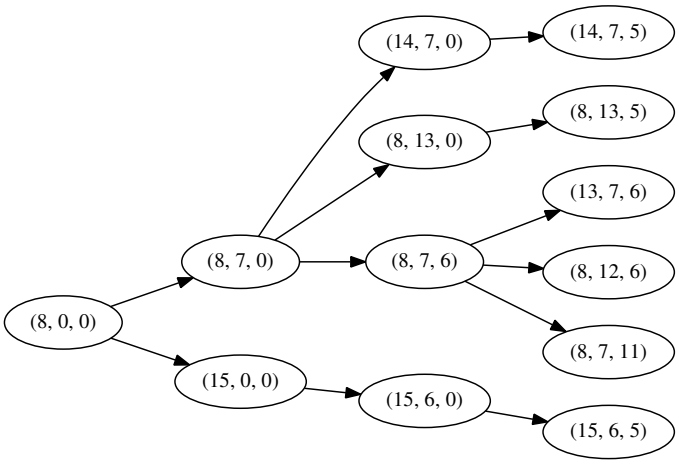

Figura 4.2: Exemplo de árvore de busca sem partes vazias para o MWNPP: $V = \{8,7,6,5\}$ e $k = 3$

As entradas do vetor *sum* presente em cada um desses nós variam de acordo com o nó que este representa. Como se vê na Figura 4.2, cada nó tem um vetor com 3 entradas. Calculando a diferença entre vetores sum^{filho} e sum^{pai}, identifica-se o último elemento alocado. Por exemplo, no caso $(15,6,5) - (15,6,0) = (0,0,5)$ significa que o elemento 5 foi adicionado na parte 3. Observa-se essa propriedade no Algoritmo 4.11.

Para construir o Algoritmo CGA, usa-se um limite inferior proposto em Korf (1998), baseado no princípio de que, se a parte com soma maior supera $P = \frac{1}{k}\sum_{i \in A_{\max}} v_i$, uma $(k-1)$-partição perfeita dos demais elementos é a melhor solução que se pode encontrar. Guardada a soma dos elementos da sequência na inicialização do algoritmo, o custo da função $lower()$, calculada conforme o Algoritmo 4.9, é $O(k)$ em qualquer nível da árvore de busca.

De fato, se este limite inferior e se uma solução factível de um nó tiverem o mesmo valor de função objetivo que o valor LB retornado na função $lower()$, este nó será um ótimo local.

Deve-se ressaltar que este limite inferior não é uma solução relaxada do MWNPP, pois não está associado a qualquer modelo matemático desse problema, como, por exemplo, o mostrado pelas expressões (2.25)-(2.29). Devido a isso, não é um bom limitante, à medida que os níveis da árvore de busca se afastam da raiz. Isto pode

Algoritmo 4.10: *upper*(): limite superior do Algoritmo CGA baseado no Algoritmo 4.1

Entrada: Uma sequência V, um vetor sum, k, nível h,$f()$
Saída: UB

```
1 início
2     para i = h + 1 até n faça
3         l = arg min_j sum;
4         A_l = A_l ∪ i;
5         sum_l = sum_l + v_i;
6     fim
7     UB = f({A_1, A_2, ..., A_k});
8     retorna UB
9 fim
```

prejudicar as podas do algoritmo.

Algoritmo 4.9: *lower*(): limite inferior do Algoritmo CGA.

Entrada: Uma sequência V, um vetor sum, k, nível h
Saída: LB

```
1  início
2      LB = 0;
3      r_1 = Σ_{i=h+1}^{n} v_i;
4      r_2 = Σ_{i=1}^{k} sum_i;
5      l = arg max_j sum;
6      se sum_l > (r_1+r_2)/k então
7          LB = sum_l − (r_1+r_2−sum_l)/(k−1);
8      fim
9      retorna LB
10 fim
```

O limite superior do Algoritmo CGA é o próprio LPT, com um vetor soma dos elementos das partes, quando os elementos de v_1 até v_h já estão pré-alocados. Com os elementos já em ordem não crescente na inicialização do método, uma execução desse algoritmo em um nível h da árvore de busca tem custo $O(n-h)$.

A função $ramifica()$ do Algoritmo CGA, mostrada no Algoritmo 4.11, apenas cria nós e os insere em uma lista F. O primeiro desvio condicional da função impede a ramificação de nós infactíveis. Essa operação não é uma poda, pois só faz o algoritmo ramificar certo, em lugar de podar uma ramificação infactível, como mostram as Figuras 4.1 e 4.2.

Claro que, na prática, a implementação desta função não deve listar nós filhos e colocá-los em uma lista para serem percorridos novamente no crivo de cortes do algoritmo. As opções para corrigir tal problema são:

(i) incorporar a função $ramifica()$ ao corpo do código do *Branch and Bound*; ou

(ii) incorporar os cortes dentro da função $ramifica()$, trocando a lista F pela própria fila Q.

Algoritmo 4.11: $ramifica()$: função de ramificação do Algoritmo CGA.

Entrada: Uma sequência V, k, $upper()$, $lower()$, No.

início
  $sum = No.sum$;
  $h = No.h$;
  $i = 1$;
  se $h \geq n - k$ *e* $sum_{n-h} == 0$ **então**
    $ramo = false$;
    $sum_{n-h} = sum_{n-h} + v_{h+1}$;
    $LB = lower(V, sum, k, h + 1)$;
    $UB = upper(V, sum, k, h + 1, f())$;
    $No_i = [LB, UB, sum, h + 1]$;
    $insere(No_i, F)$;
  fim
  enquanto $ramo == true$ **faça**
    $sum_i = sum_i + v_{h+1}$;
    $LB = lower(V, sum, k, h + 1)$;
    $UB = upper(V, sum, k, h + 1, f())$;
    $No_i = [LB, UB, sum, h + 1]$;
    $insere(No_i, F)$;
    $sum_i = sum_i - v_{h+1}$;
    $i = i + 1$;
    se $sum_i == 0$ *ou* $i == k$ **então**
      $ramo = false$;
      $sum_i = sum_i + v_{h+1}$;
      $LB = lower(V, sum, k, h)$;
      $UB = upper(V, sum, k, h, f())$;
      $No_i = [LB, UB, sum, h + 1]$;
      $insere(No_i, F)$;
      $sum_i = sum_i - v_{h+1}$;
    fim
  fim
  retorna F
fim

A estrutura completa do Algoritmo CGA é mostrada no Algoritmo 4.12. Ele funciona de maneira similar ao Algoritmo 4.8, mas usa as funções auxiliares dos Algoritmos 4.9, 4.10 e 4.11. O nó raiz é inicializado com um limite inferior $LB = 0$, a menos que o elemento v_1 seja maior que $\frac{1}{k}\sum_{i \in V} v_i$. Existe a possibilidade de iniciar este algoritmo com uma solução diferente, dada pelo retorno da função $upper()$, denotada por $SolInit()$, pois qualquer *Branch and Bound* é sensível à qualidade da solução inicial. O algoritmo finaliza ou por limite de tempo ou por critério de otimalidade global.

O funcionamento completo do Algoritmo CGA é melhor descrito pelas Figuras

4.3 e 4.4.

Algoritmo 4.12: Algoritmo CGA

Entrada: $SolInit(), lower(), upper(), ramifica(), V, k$
Saída: UB

```
1  início
2      Criar uma fila Q;
3      h = 1;
4      sum_1 = v_1;
5      ub = SolInit();
6      lb = lower(V, sum, k, h + 1);
7      No = [lb, ub, sum, h];
8      insere(Q, No);
9      UB = ∞;
10     LB = 0;
11     i = 1;
12     OPT = true;
13     enquanto |Q| > 0 ou OPT == true faça
14         No_i = remove(Q);
15         F = ramifica(V, k, upper(), loower(), No_i);
16         para No_j ∈ F faça
17             se No_j.lb > UB então
18                 continue;
19             fim
20             se No_j.ub == No_j.lb então
21                 se No_j.ub > UB então
22                     UB = No_j.ub;
23                 fim
24                 continue;
25             fim
26             senão
27                 insere(Q, No_j);
28                 se No_j.ub > UB então
29                     UB = No_j.ub;
30                 fim
31             fim
32         fim
33         se tempo ≥ limite então
34             OPT = false;
35             |Q| = 0;
36         fim
37     fim
38     retorna UB
39 fim
```

A Figura 4.3 mostra a árvore de busca do Algoritmo 4.12 aplicado a um exemplo de pequena dimensão. A ordem de exploração dos nós é de cima para baixo e da esquerda para a direita. As marcações nas arestas são $j : v_i$, indicando que, naquele

nó, insere-se o elemento v_i na parte j.

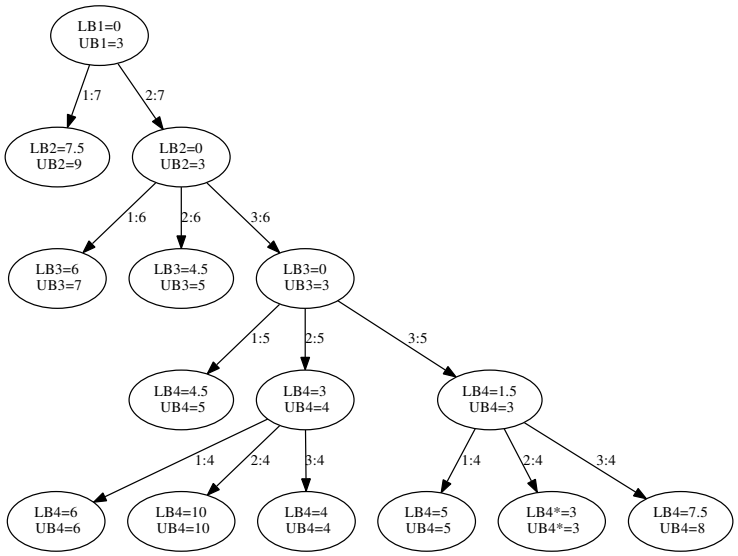

Figura 4.3: Exemplo de árvore de busca com podas $V = \{8, 7, 6, 5, 4\}$ e $k = 3$.

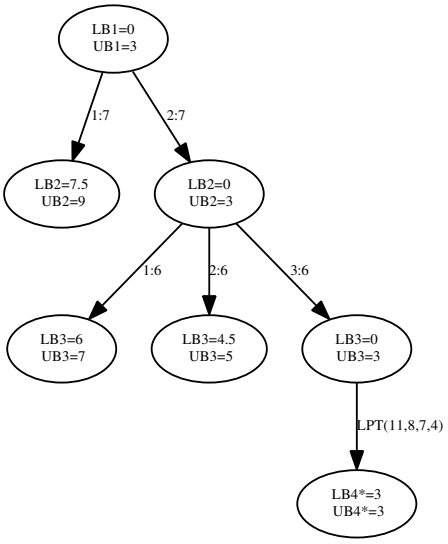

Figura 4.4: Exemplo de árvore de busca com podas e terminada com o algoritmo (LPT) a partir do nível $n - k - 2$.

Inicialmente, conforme a Figura 4.3, a raiz tem $LB_1 = 0$ e $UB_1 = 3$, o nó correspondente ao elemento v_1 fixado na parte 1. A exploração dos nós segue até a certificação de otimalidade global, ou seja, quando $LB_i = UB_i = 0$ ou $LB_i = UB_i$ no nível n da árvore de busca. Nesta fase não existem mais nós ativos.

Em Korf (2011) mostra-se que o LPT é exato quando $n \leq k + 2$. Este resultado facilita o uso do Algoritmo CGA, pois não existe a necessidade de explorar nós no nível $n - k - 2$ da árvore de busca e basta usar o LPT em cada nó ativo e o menor limite superior será o valor ótimo do problema.

Capítulo 5

Algoritmos Propostos para o Problema da k-Partição de Números

Este Capítulo apresenta algoritmos heurísticos e exatos propostos nesta dissertação para a solução do Problema da k-Partição de Números (MWNPP). O Capítulo é iniciado com a Seção 5.1, em que uma proposta de algoritmo geral para a solução do MWNPP é introduzida. Em seguida, a Seção 5.2 mostra uma proposta de adaptação da meta-heurística *Iterated Local Search* (ILS) para a solução do problema em análise. A Seção 5.4 mostra um algoritmo pseudo-polinomial para o Problema de Decisão 3.2.1. Por fim, a Seção 5.5 apresenta uma proposta de algoritmo exato baseado no CGA.

5.1 Algoritmo Genérico para MWNPP

Seja V uma sequência de números. Considere a existência de um algoritmo, denominado $part_2(V, X)$, que aproxima a solução do MWNPP dada uma partição inicial, e um algoritmo $part_1(V, k)$, que encontra uma solução inicial para o MWNPP. É possível, então, propor um algoritmo, inspirado no Algoritmo 0/1-*Interchange*, mostrado no Algoritmo 4.2 da Seção 4.2, que solucione o Problema da k-Partição de Números (MWNPP).

O Algoritmo 5.1 contempla esta proposição, sendo uma proposta de solução geral

para o Problema da k-Partição de Números.

Algoritmo 5.1: *iterails*(): Algoritmo de solução Geral para o MWNPP.

Entrada: Um sequência V, $part_1()$, $part_2()$, k, $f()$.
Saída: k-partição dos índices de V.

```
1  início
2      {A_1, ..., A_k} = part_1(V, k);
3      repita
4          obj = f(A_1, A_2, ..., A_k);
5          j' = arg max_{j'} { Σ_{i∈A_{j'}} v_i };
6          j = arg min_{j} { Σ_{i∈A_j} v_i };
7          X = A;
8          troca(X, 1, j');
9          troca(X, k, j);
10         {A_1, A_2, ..., A_k} = part_2(V, X);
11     até obj == f(A_1, A_2, ..., A_k);
12     retorna {A_1, A_2, ..., A_k};
13 fim
```

Após a linha 2, a comparação entre obj e a função objetivo do Algoritmo 5.1 descrito acima, pode-se listar dois casos:

(i) a parte de maior soma fica menor e a parte de menor soma fica maior; ou

(ii) ambas as partes permanecem inalteradas. Enquanto o primeiro caso ocorre, a função objetivo do problema decresce.

Portanto, o ótimo ainda não foi encontrado.

Observe que o Algoritmo 5.1 é um algoritmo geral para a solução do MWNPP. O Algoritmo $part_2()$ é parte desse algoritmo geral, assim como o Algoritmo $part_1()$, que retorna uma solução inicial. O Algoritmo 5.1 cumpre a função de gerar a partição do conjunto que leve a uma solução viável do MWNPP. Qualquer método que execute as funções dos Algoritmos $part_1$ e $part_2$ pode vir a ser implementado, seja um método exato, uma heurística ou uma meta-heurística.

A Seção 5.2 apresenta uma adaptação da meta-heurística ILS, mostrada em sua versão geral no Algoritmo 4.4 da Seção 4.4, para atuar com a função do Algoritmo $part_2$.

5.2 Adaptação do ILS para a Solução do Problema da k-Partição de Números

Com uma solução inicial já bem aproximada para o problema, basta encontrar uma boa estratégia para rearranjar elementos e buscar uma solução ainda melhor. O refinamento acontece pela busca de um elemento a ser realocado, pertencente à

parte de maior soma. Se esse elemento não superar a metade do valor da função objetivo para partição atual, busca-se um segundo elemento na mesma parte para completar essa diferença. Caso contrário, busca-se um segundo elemento na parte de menor soma para retirar o excesso. Caso não exista forma de se fazer uma troca ou realocação de dois elementos sem a piora da função objetivo, tenta-se a realocação de um único elemento. O algoritmo para quando não existirem mais trocas de melhora.

As subseções a seguir detalham esta estratégia.

5.2.1 Movimento e Vizinhança

O movimento de realocação sempre aproxima a parte de maior soma, encolhendo-a, da parte de menor soma, aumentando-a. O movimento $m_{i,j}$ significa que um elemento sai da parte de índice i e vai para parte de índice j. Assim, gera-se a vizinhança $N(s)$ da solução s, na forma:

$$N(s) = \{s' \ : \ s' \leftarrow s \oplus m_{i,j}, \quad \forall i \in A\} \tag{5.1}$$

sendo A o conjunto de índices das partes. A dimensão da vizinhança é $|A|$. Essa operação é a perturbação do ILS descrito no Algoritmo 4.4 na Seção 4.4 do Capítulo 4.

5.2.2 Estratégia de Realocação

Os movimentos entre elementos das partes são definidos na forma de problemas de otimização. A seguir, são descritos os movimentos de Realocação de um elemento, Realocação de dois elementos e Troca de dois elementos. A solução desses problemas aponta os elementos que devem ser trocados ou realocados. Essa operação é a busca local do ILS descrito no Algoritmo 4.4 na Seção 4.4 do Capítulo 4.

Realocação de um elemento

A ideia principal do método é determinar, dentre todos os elementos da parte de maior soma, aquele que mais reduz o valor da função objetivo, e realocá-lo na parte de menor soma. Essa busca gera um subproblema, que consiste em encontrar:

$$\max_{a_i \in A_{\max}} \quad \{a_i\} \tag{5.2}$$

$$\text{s.a} \quad a_i \leq \frac{1}{2}\left(\sum_{i \in A_{\max}} v_i - \sum_{i \in A_{\min}} v_i\right) \tag{5.3}$$

Realocação de dois elementos

Pode-se, também, realocar dois elementos de uma só vez, passando-os da parte de maior soma para a parte de menor soma, caso este seja o movimento que reduz o valor da função objetivo. O subproblema associado a essa busca é:

$$\max_{a_i \in A_{\max}, b_j \in A_{min}} \quad \{a_i + b_j\} \tag{5.4}$$

$$\text{s.a} \quad a_i + b_j \leq \frac{1}{2}\left(\sum_{i \in A_{\max}} v_i - \sum_{i \in A_{\min}} v_i\right) \tag{5.5}$$

Troca de dois elementos

A ideia secundária é determinar, dentre todos os elementos a_i da parte de maior soma, e b_j, da parte de menor soma, aqueles que mais aumentam a função objetivo. Para isso, troca-se os dois elementos cuja diferença seja a mais próxima possível da metade do valor da função objetivo. Essa busca gera um subproblema, que consiste em encontrar:

$$\min_{a_i \in A_{\max}, b_j \in A_{min}} \quad \{a_i - b_j\} \tag{5.6}$$

$$\text{s.a} \quad a_i - b_j \leq \frac{1}{2}\left(\sum_{i \in A_{\max}} v_i - \sum_{i \in A_{\min}} v_i\right) \tag{5.7}$$

5.2.3 Critérios de Parada

Os movimentos continuam enquanto houver possibilidade de melhora da função objetivo pelo movimento ou um limite de iterações não chegar ao fim. Caso contrário, o algoritmo para. Esse critério evita ciclos, mesmo sem reconhecer trocas já realizadas anteriormente.

5.2.4 Implementação

O ILS deve ser usado no lugar da função $part_2()$ dentro do Algoritmo 5.1, já que este opera trocas e realocações até o critério de parada. O Algoritmo 5.2 atualiza suas entradas, realocando um ou dois elementos de X_1 para X_k ou trocando dois

elementos entre essas partes.

Algoritmo 5.2: $ils()$: Adaptação do ILS para a solução do Problema da k-Partição de Números.

Entrada: $\{X_1, X_2, ..., X_k\}, obj$
Saída: $\{X_1, X_2, ..., X_k\}, obj$

```
1  início
2      obj1 = obj;
3      para a ∈ X_1 faça
4          troco = a − obj1/2;
5          se troco > 0 então
6              s1 = busca-remove(a, X_1);
7              s2 = busca-remove(troco, X_k);
8              insere(s1, X_k);
9              insere(s2, X_1);
10             obj = obj − (s1 − s2);
11         fim
12         senão se troco < 0 então
13             s1 = busca-remove(a, X_1);
14             s2 = busca-remove(−troco, X_1);
15             insere(s1, X_k);
16             insere(s2, X_k);
17             obj = obj − (s1 + s2);
18         fim
19         senão
20             s1 = busca-remove(a, X_1);
21             insere(s1, X_k);
22             obj = obj − s1;
23         fim
24     fim
25     retorna {X_1, X_2, ..., X_k}, obj
26 fim
```

Suponha, sem perda de generalidade, que $\{X_1, X_2, ..., X_k\}$ seja uma partição com as partes ordenadas em ordem decrescente por suas somas. Assim, X_1 representa a parte de maior soma e X_k representa a parte de menor soma.

Usa-se o ILS para fazer movimentos entre X_1 e X_k, já que este opera trocas e realocações somente entre as partes de maior e menor soma. O Algoritmo 5.2 atualiza suas entradas, realocando um ou dois elementos de X_1 para X_k ou trocando dois elementos entre essas partes. Após reduzir o custo da função objetivo, ele retorna ao Algoritmo 5.1 para decidir se deve buscar um novo movimento.

Representa-se a partição por uma min max `heap` de tabelas *hash*. Com esta estrutura de dados, o custo para encontrar a maior e a menor parte após o movimento do ILS tem complexidade $O(log(n))$. A função *busca-remove*(a, S) procura o maior elemento menor ou igual ao valor a em uma lista encadeada. Por isso, os dados de X_1 e X_k são armazenados em uma tabela *hash*, afim de amortizar o custo dessa busca. A função *hash* usada na tabela tem as propriedades de um histograma, mantendo elementos de um determinado intervalo dentro da mesma posição.

Seja a sequência V com todos os seu elementos contidos no intervalo (a, b). Para alocar os elementos de V em uma tabela com tamanho m usa-se a função:

$$hash(x) = \left\lfloor m.\frac{(x-a)}{(b-a)} \right\rfloor \tag{5.8}$$

Como as instâncias têm distribuição uniforme, espera-se que as listas em cada posição da tabela contenham $\frac{n}{m}$ elementos. Essa propriedade acelera a busca pela variável `troca`, permitindo que o Algoritmo 5.2 tenha uma complexidade amortizada de $O(n)$.

5.3 Intensificação do LPT

Uma modificação trivial do Algoritmo LPT, mostrado no Algoritmo 4.1, que é capaz de intensificar sua resposta é apresentada no Algoritmo 5.3, denominado LPT_1. Sua ideia é uma rotina que lista qual é a resposta do LPT quando iniciado com o elemento v_i na posição 1.

Algoritmo 5.3: Lpt_1: guarda as melhores soluções de n iterações do LPT

Entrada: Uma sequência V, k, $f()$.
Saída: aux.

```
1  início
2      para h = 1 até n faça
3          sum_1 = v_h, a = v_h e v_h = 0;
4          para i = 1 até n faça
5              l = arg min_j sum;
6              A_l = A_l ∪ i;
7              sum_l = sum_l + v_i;
8          fim
9          v_h = a e sum_1 = 0;
10         ub = f({A_1, A_2, ..., A_k});
11         se ub < aux então
12             aux = ub;
13         fim
14     fim
15     retorna aux
16 fim
```

Para cada possibilidade, guarda-se o menor valor em uma variável auxiliar aux. Assim, o Algoritmo Lpt_1 tem complexidade $O(n^2)$. Esse método garante que a resposta do Algoritmo Lpt_1 seja sempre menor ou igual à resposta do Algoritmo LPT.

5.4 Algoritmos Exatos

Uma ideia ingênua para resolver o problema MWNPP com um algoritmo exato para o Problema da Soma de Subconjuntos, já descrito no Algoritmo 4.3, é obter

uma parte com a soma mais próxima de $\frac{1}{k}\sum_{i\in I_n} v_i$, retirá-la de V, decrementar k e fazer isso até $V = \emptyset$. Esta ideia está posta no Algoritmo 5.4.

Algoritmo 5.4: $kpart()$: o algoritmo retorna parte a parte, usando, k vezes, um algoritmo exato para o Problema da Soma de Subconjuntos.

Entrada: Uma sequência V, inteiro k, $subsetsum()$
Saída: k-Partição do conjunto V, $\{A_1, A_2, ..., A_k\}$

```
início
    se V = ∅ então
        break;
    fim
    A_k = subsetsum(V, (1/k) Σ_{i∈I_n} v_i);
    V = V \ A_k;
    {A_1, A_2, ..., A_{k-1}} = kpart(V, k − 1);
    retorna {A_1, A_2, ..., A_k};
fim
```

Deve-se lembrar que a função $subsetsum()$ é dada pelo Algoritmo 5.3.

Em muitos casos, a aplicação do Algoritmo 5.4 é inviável, pois utiliza um algoritmo exponencial k vezes. Além disso, o Algoritmo 5.4 pode falhar, caso a instância seja livre de partições perfeitas, com valor de função objetivo igual a 0, ou quando um subconjunto retirado tem dois elementos pertencentes a partes distintas na partição ótima.

Um método correto de Programação Dinâmica é considerar a função $f(i, sum_j)$ como a solução ótima do MWNPP, usando os i primeiros elementos de V com a soma da parte j igual a sum_j para cada $j \in \{1, ..., k-1\}$. Pode-se escrever esta função da seguinte maneira:

$$f(i, sum_j) = \min_{1\leq l\leq k-1}\{f(i-1, sum_{j\neq l} \cup sum_l - a_i), f(i-1, sum_j)\} \tag{5.9}$$

Esta Equação de recorrência 5.9 informa que $f(i, sum_j)$ é o menor valor de todos os subproblemas $f(i-1, sum_{j\neq l} \cup sum_l - a_i)$, supondo que o elemento a_i está na parte l para cada $l \in \{1, ..., k-1\}$, ou o subproblema $f(i-1, sum_j)$ que indica que elemento a_i está na parte k. A solução procurada será $f(n, sum_j)$. Esse método é inviável caso os elementos de V tenham muitos algarismos.

5.5 Algoritmo Híbrido CGA+ILS

A maior vantagem das heurísticas simples, com baixa complexidade, é que elas são uma parte importante dos métodos exatos de enumeração implícita. Considerando o Algoritmo CGA, mostrado no Algoritmo 4.12 da Seção 4.6 do Capítulo 4, uma heurística construtiva pode ser aplicada dentro de um algoritmo exato para gerar limites superiores, enquanto uma relaxação serve para encontrar limites inferiores.

Seja $H(I)$ uma heurística, $R(I)$ uma relaxação e $Opt(I)$ o ótimo global para o problema de minimização I. Então, tem-se que:

$$R(I) \leq Opt(I) \leq H(I) \tag{5.10}$$

Logo, uma melhoria em uma relaxação ou em uma heurística usada pode gerar cortes em uma árvore de busca e acelerar o desempenho de um método de enumeração implícita.

Para melhorar o desempenho do CGA, propõe-se uma série de modificações nos seus principais métodos, com exceção da função *ramifica*().

5.5.1 Limite Inferior para a Relaxação

Esta Subseção apresenta um procedimento para a determinação de um limite inferior para a relaxação do problema MWNPP.

Primeiro, troca-se o limite inferior de um subproblema associado a um nó por um resultado mais próximo da solução ótima do problema relaxado apresentado no modelo dado pelas Expressões (2.25)-(2.29).

Agora, o objetivo é supor uma distribuição ideal dos elementos não alocados. Dado que uma ou mais partes superam $\frac{1}{k}\sum_{i=1}^{n} v_i$, resta aumentar ao máximo o tamanho da menor parte distribuindo elementos de modo fracionário.

O Algoritmo 5.5 formaliza o procedimento de cálculo do limite inferior da relaxação.

Algoritmo 5.5: *lower*(): limite inferior do CGA baseado na relaxação do modelo matemático (2.25)-(2.29).

Entrada: Uma sequência V, um vetor sum, k, nível h.
Saída: LB.

```
1  início
2      r = Σ_{i=1}^{n} v_i,;
3      x = k;
4      s = r;
5      para l = 1 até k faça
6          se sum_l > r/k então
7              s = s − sum_l;
8              x = x − 1;
9          fim
10     fim
11     l = arg max_j sum;
12     LB = sum_l − s/x;
13     retorna LB
14 fim
```

5.5.2 Limite Superior para a Relaxação

Esta Subseção apresenta um procedimento para a determinação de um limite superior para a relaxação do problema MWNPP.

Assim, neste caso, usa-se o Algoritmo 5.2, ou seja, a adaptação da metaheurística ILS proposta para a solução do MWNPP, para que o limite superior decresça mais rapidamente. O Algoritmo 5.6 formaliza esta estrutura de cálculo.

Para o uso do Algoritmo 5.2 como parte integrante da função *upper()*, mostrada no Algoritmo 5.6, sugere-se um pequeno número máximo de iterações, para evitar problemas com a diferença de complexidade na exploração de 2 nós distintos. Dessa maneira, limita-se o Algoritmo 5.2 em, no máximo, 10 iterações. Aplicam-se os movimentos apenas em elementos v_i com $i > h$, sendo h a profundidade do nó na árvore de busca, para não descaracterizar a solução parcial do nó explorado. Com esta mudança, espera-se uma convergência mais rápida do algoritmo.

Como cada iteração do Algoritmo 5.2 tem a mesma complexidade assintótica do LPT, pode-se dizer que cada nó explorado com essa modificação fica 11 vezes mais caro.

Algoritmo 5.6: Limite superior refinado pelo ILS baseado nos algoritmos 5.1 e 5.2, $upper()$

Entrada: Uma sequência V, um vetor sum, k, nível h,$f()$.
Saída: UB.

```
1  início
2      para i = h + 1 até n faça
3          l = arg min_j sum;
4          A_l = A_l ∪ i;
5          sum_l = sum_l + v_i;
6      fim
7      {A_1, A_2, ..., A_k} = iterails({A_1, A_2, ..., A_k});
8      UB = f({A_1, A_2, ..., A_k});
9      retorna UB
10 fim
```

5.5.3 Prioridade de Busca na Árvore

Esta Subseção apresenta um procedimento para determinar a ordem de exploração dos nós na árvore de busca.

O CGA original usa uma fila comum para fazer sua busca em largura. Este é um método clássico e sua principal vantagem é o baixo custo da retirada e inserção de nós. Usando uma fila de prioridades implementada em uma `min-heap`, pode-se impor uma ordem de exploração, de modo que os nós mais promissores serão explorados primeiramente. Os nós mais promissores são aqueles com limites superiores mais próximos do seus limites inferiores e, ao mesmo tempo, do menor limite superior histórico.

O Algoritmo 5.7 retorna uma medida da adequação do nó a estas propriedades. Uma penalidade, com um fator proporcional ao seu nível, impede que se priorize apenas nós em níveis mais baixos.

Algoritmo 5.7: Prioridade de busca do CGA+ILS baseado no critério de otimalidade local, *prioridade*()

Entrada: Menor limite superior histórico UB, No.
Saída: p.

```
1 início
2     se UB > No.lb então
3         p = (No.ub − No.lb)/(UB − No.lb) + 1/No.h;
4     fim
5     senão
6         p = ∞;
7     fim
8     retorna p
9 fim
```

A fila de prioridades consome $O(log(n))$ para inserir ou retirar nós, sendo n o tamanho da fila. Isso pode ser trabalhoso, como mostra a Figura 2.1. Este método só é viável na prática devido ao volume de cortes produzidos pelos novos limitantes.

5.5.4 Estrutura Geral do Algoritmo CGA+ILS

A estrutura geral do Algoritmo CGA+ILS é mostrada no Algoritmo 5.8. Ela inclui as alterações do Algoritmo CGA discutidas nas subseções precedentes, de modo a melhorar tanto o cálculo de limitantes inferiores quanto de limitantes superiores para a relaxação do problema MWNPP, como também inclui a melhor análise de prioridade de busca da árvore.

O primeiro desvio condicional dentro do laço de repetição `enquanto` é a propriedade da Figura 4.4, que mostra um método de fuga da exploração dos dois últimos níveis da árvore de busca. Esta ideia está na versão original do CGA, apesar de ser um resultado encontrado em Korf (2011). Nela, aplica-se o LPT, considerando $V = \{sum_1, ..., sum_k, v_{n-1}, v_n\}$. A otimalidade global dessa solução está garantida, pois $|V| = n - k - 2$.

Algoritmo 5.8: Algoritmo Híbrido CGA+ILS

Entrada: $lower(), upper(), ramifica(), V, k, LPT()$.
Saída: UB.

```
1  início
2      Criar uma fila Q;
3      h = 1;
4      sum_1 = v_1;
5      ub = SolInit(V, sum, k, h + 1);
6      lb = lower(V, sum, k, h + 1);
7      No = [lb, ub, sum, h];
8      insere(Q, No, prioridade(No, ub));
9      UB = ub;
10     LB = 0;
11     i = 1;
12     OPT = true;
13     enquanto |Q| > 0 ou OPT == true faça
14         No_i = remove(Q);
15         se No_i.h == n − k − 2 então
16             X = sum ∪ {v_{n−1}, v_n};
17             UB_i = LPT(X, k);
18         fim
19         F = ramifica(V, k, upper(), lower(), No_i);
20         para No_j ∈ F faça
21             se No_j.lb > UB então
22                 continue;
23             fim
24             se No_j.ub == No_j.lb então
25                 se No_j.ub > UB então
26                     UB = No_j.ub;
27                 fim
28                 continue;
29             fim
30             senão
31                 se No_j.ub > UB então
32                     UB = No_j.ub;
33                 fim
34                 insere(Q, No_j, prioridade(No_j, UB));
35             fim
36         fim
37         se tempo ≥ limite então
38             OPT = false;
39             |Q| = 0;
40         fim
41         i = i + 1;
42     fim
43     retorna UB
44 fim
```

Capítulo 6

Resultados Computacionais

Este Capítulo apresenta os resultados da aplicação dos algoritmos estudados e desenvolvidos nesta dissertação. Está organizado da seguinte forma: a Seção 6.1 apresenta uma comparação entre heurísticas construtivas e o ILS proposto. A Seção 6.2 mostra uma comparação entre o CGA, implementado como na literatura, e o CGA+ILS com prioridade de busca e limitantes modificados.

Os algoritmos foram implementados em linguagem `C++`. Os testes computacionais foram realizados em um computador com processador Intel Core i3-M330, 2.13 GHz, 3GB de RAM e sistema operacional Ubuntu 14.04 de 32 bits.

Obtêm-se resultados dos algoritmos propostos com instâncias geradas aleatoriamente. Os conjuntos gerados têm dimensão entre $n \in \{100, 200, 400, 800\}$ e os elementos são inteiros, variando de 0 até 999999999999, com 12 algarismos, tendo distribuição uniforme. A dimensão n da sequência e o número de partes k são parâmetros de entrada do gerador de instâncias. Um número entre 0 e 9, inclusive, é gerado aleatoriamente por 12 vezes consecutivas. Esse processo é repetido n vezes para a geração de uma instância de teste.

6.1 Algoritmos Heurísticos

O objetivo dos experimentos computacionais deste trabalho é analisar os algoritmos heurísticos apresentados nos Capítulos 4 e 5, quais sejam, Algoritmos 4.1, 4.5, 5.3 e 5.1, de modo a realizar uma comparação entre o melhor resultado das heurísticas construtivas e o ILS.

Para as Tabelas 6.1, 6.2, 6.3 e 6.4, adota-se a seguinte notação: a variável inst[i] indica a i-ésima instância: de inst1 até inst5, n=100; de inst6 até inst10, n=200; de inst11 até inst15, n=400 e de inst16 até inst20, n=800. O número $k \in \{3, 4, 5, 6\}$ complementa a instância, indicando o número de subconjuntos da partição. A primeira coluna indica a instância do experimento. Da segunda até a quarta coluna tem-se os algoritmos Lpt_0, Lpt_1, ILS, KK e suas respectivas soluções encontradas. Da sexta até a oitava coluna tem-se a medida de erro relativo, calculada na forma:

$$Gap(A/B) = \frac{z(A) - z(B)}{z(A)}.100\% \tag{6.1}$$

que mostra o quanto a resposta do algoritmo B é menor que a do algoritmo A, quando $Gap(A/B) > 0$, ou maior, caso $Gap(A/B) < 0$, sendo $z(A)$ e $z(B)$ seus respectivos valores de função objetivo. Se, por exemplo, $Gap(A/B) = 90\%$ significa que a resposta do Algoritmo A teria que ser dividida por 10 para se igualar à resposta do algoritmo B. Não se compara o tempo de solução ou número de operações elementares desses algoritmos apresentados, pois todos têm complexidades entre $O(n)$ com multiplicidade menor que 10, $O(n.log(n))$ e $O(n^2)$. A ordem dos algoritmos na entrada da função $Gap()$ foi cuidadosamente escolhida para manter o valor positivo e facilitar a conferência dos dados. Os casos em que $Gap() = 100\%$ são causados pelo arredondamento de $99,999...\%$.

Apesar dos trabalhos Gent e Walsh (1998) e Berretta e Moscato (1999) descreverem que a Heurística de Karmarkar-Karp supera várias heurísticas e meta-heurísticas em uma grande bateria de testes, ela se apresenta superada pelo Lpt_1 e pelo ILS para todas as instâncias deste experimento, conforme mostrado nas Tabelas 6.1, 6.2, 6.3 e 6.4, para todos os valores de k considerados.

Na Tabela 6.5 tem-se uma comparação da Heurística de Karmarkar-Karp (KK) e Lpt_1 com o ILS apresentado no Algoritmo 5.2. Esta comparação serve, por um lado, para justificar que as instâncias geradas aleatoriamente não são fáceis, já que, conforme pode ser avaliado, a heurística KK falha em se aproximar da solução ótima e, por outro lado, justifica o uso do ILS dentro da função $upper()$ a fim de otimizar o CGA. Mostra-se isso com o Gap mínimo dessa Tabela, dado por $70,52\%$ na instância *inst4* para $k = 3$. Esta porcentagem informa o quanto a heurística KK teria que reduzir sua solução para se equiparar ao valor encontrado pela meta-heurística ILS. O algoritmo ILS proposto supera a heurística KK com um $Gap()$ médio de mais de 99% e supera, também, Lpt_1 com um $Gap()$ médio de mais de 88% em todas as instâncias testadas.

É importante lembrar que, como a heurística LPT é, por definição, dominada pela heurística Lpt_1, ela não é incluída nas comparações apresentadas. Pode-se notar que a heurística Lpt_1 possui um menor $Gap()$ na instância *inst13* para $k = 5$, sendo o valor do $Gap()$ igual a $4,02\%$, uma diferença pouco significativa. Mas é importante ressaltar que, mesmo nesse caso, o valor de $Gap()$ das amostras é alto, conforme pode ser observado na Tabela 6.6. De fato, a meta-heurística ILS supera ambas as heurísticas construtivas na análise dos valores de $Gap()$.

Um teste t com 95% de confiança rejeita a hipótese nula de que o $Gap(KK/ILS) \leq 97\%$ sendo o resultado expressado pelo p-valor igual a $0,0001470751$, representando que a probabilidade $P(Gap(HKK/ILS) \leq 97\%) = 0,0001470751$. De modo informal, pode-se dizer que as respostas do ILS são $33,33...$ vezes menores que as respostas de KK.

Outro teste t com realizado 95% de confiança rejeita a hipótese nula de que o $Gap(Lpt\_1/ILS) \leq 50\%$ sendo o resultado expressado pelo p-valor igual a $0,000011475$, representando que a probabilidade $P(Gap(Lpt\_1/ILS) \leq 50\%) = 0,000011475$. Em geral, isso significa que em 95% dos testes realizados teremos as respostas do ILS duas vezes menores que as respostas do Lpt_1.

O quadro 6.6 demonstra os quartis inferiores dos resultados de $Gap(KK/ILS)$ comparados com os quartis superiores dos resultados de $Gap(Lpt\_1/ILS)$. Esse quadro comprova que o Algoritmo Lpt_1 se sai melhor contra o ILS do que o Algoritmo HKK sem a necessidade de um teste estatístico porque o menor quartil

inferior de HKK vs ILS supera em $2,95\%$ o maior quartil superior de Lpt_1 vs ILS.

K=3							
Inst/Alg	LPT	Lpt_1	ILS	KK	Gap 0/1	Gap 1/ILS	Gap KK/1
inst1	277667604	5937037	1687512	41346336	97,86%	71,58%	85,64%
inst2	33266420	15480040	1044716	1604884656	53,47%	93,25%	99,04%
inst3	130950041	39364611	4079508	2946161439	69,94%	89,64%	98,66%
inst4	42031699	35502732	17237659	58479988	15,53%	51,45%	39,29%
inst5	25191261	7118067	842232	6318269369	71,74%	88,17%	99,89%
inst6	218772576	9603978	3748156	636232165	95,61%	60,97%	98,49%
inst7	129206619	16126140	971246	103818313	87,52%	93,98%	84,47%
inst8	124586508	118159745	4266521	2328744777	5,16%	96,39%	94,93%
inst9	94160686	4227340	986676	74741269	95,51%	76,66%	94,34%
inst10	148223495	119895842	3903241	5698061694	19,11%	96,74%	97,90%
inst11	41115509	1947877	195622	3489497722	95,26%	89,96%	99,94%
inst12	78985609	48162349	19697495	1930925747	39,02%	59,10%	97,51%
inst13	78687824	24772373	605450	6020635525	68,52%	97,56%	99,59%
inst14	53432822	23953655	498615	378777978	55,17%	97,92%	93,68%
inst15	67315512	3972386	501916	16033000	94,10%	87,36%	75,22%
inst16	16378823	7254715	12989	160225313	55,71%	99,82%	95,47%
inst17	25902922	14815589	1012403	3149419117	42,80%	93,17%	99,53%
inst18	88537526	34748915	1781184	5407329606	60,75%	94,87%	99,36%
inst19	78236976	15483485	839650	1750977565	80,21%	94,58%	99,12%
inst20	7879334	634938	44351	3163580778	91,94%	93,01%	99,98%

Tabela 6.1: Comparação entre ILS e heurísticas construtivas com $k = 3$.

K=4							
Inst/Alg	LPT	Lpt_1	ILS	KK	Gap 0/1	Gap 1/ILS	Gap KK/1
inst1	122406112	71137211	27098191	1284570205	41,88%	61,91%	94,46%
inst2	447131534	267558610	103757256	2546849966	40,16%	61,22%	89,49%
inst3	87670981	25383442	3529043	4174645695	71,05%	86,10%	99,39%
inst4	255279548	223112333	76068954	603181331	12,60%	65,91%	63,01%
inst5	68540827	28873113	5495161	9416668966	57,87%	80,97%	99,69%
inst6	163368284	131740030	85445183	2645091384	19,36%	35,14%	95,02%
inst7	108654410	20649653	2191613	1825896268	81,00%	89,39%	98,87%
inst8	37173373	11387722	4860900	6791194503	69,37%	57,31%	99,83%
inst9	63897598	21408945	6779057	2384730430	66,49%	68,34%	99,10%
inst10	51519147	46310499	26276183	4364461985	10,11%	43,26%	98,94%
inst11	32449998	16007361	154687	6663918656	50,67%	99,03%	99,76%
inst12	148352240	55565684	35865428	5667277649	62,54%	35,45%	99,02%
inst13	98619586	69540563	13420539	7414857611	29,49%	80,70%	99,06%
inst14	91039351	76185564	28074947	461236599	16,32%	63,15%	83,48%
inst15	63619258	28409584	24758230	2628417671	55,34%	12,85%	98,92%
inst16	23733845	13964928	1236509	2737419471	41,16%	91,15%	99,49%
inst17	75575159	57203876	26758439	5197880087	24,31%	53,22%	98,90%
inst18	56084420	31967817	12018266	6212768244	43,00%	62,41%	99,49%
inst19	45002148	26991187	16414242	4265433595	40,02%	39,19%	99,37%
inst20	21909411	6501610	187551	6475984152	70,33%	97,12%	99,90%

Tabela 6.2: Comparação entre ILS e heurísticas construtivas com $k = 4$.

K=5							
Inst/Alg	LPT	Lpt_1	ILS	KK	Gap 0/1	Gap 1/ILS	Gap KK/1
inst1	206448413	107820802	42456197	1339796923	47,77%	60,62%	91,95%
inst2	475570355	374614686	119154711	3412666863	21,23%	68,19%	89,02%
inst3	229590226	37937749	15730554	4336821085	83,48%	58,54%	99,13%
inst4	388564340	170727925	142781164	3122772011	56,06%	16,37%	94,53%
inst5	120270467	64320832	18155495	8643611068	46,52%	71,77%	99,26%
inst6	197777765	145917465	65858801	6430680846	26,22%	54,87%	97,73%
inst7	118818882	54510235	6682085	954421112	54,12%	87,74%	94,29%
inst8	178729820	90690347	72495658	7067196783	49,26%	20,06%	98,72%
inst9	107509840	32264710	18531124	6521820777	69,99%	42,57%	99,51%
inst10	183140948	170472298	49237473	5625413088	6,92%	71,12%	96,97%
inst11	62620330	26391490	11365285	7508244798	57,85%	56,94%	99,65%
inst12	174329379	156454992	95654464	6263240002	10,25%	38,86%	97,50%
inst13	73922250	64052640	61480216	7133551983	13,35%	4,02%	99,10%
inst14	107866991	85294733	62867537	1937028486	20,93%	26,29%	95,60%
inst15	47100982	34905809	21268395	2729821671	25,89%	39,07%	98,72%
inst16	43791832	21086571	3100909	6588625650	51,85%	85,29%	99,68%
inst17	79153253	66484390	37416121	4463460829	16,01%	43,72%	98,51%
inst18	59590511	35161862	28291419	8431894886	40,99%	19,54%	99,58%
inst19	69433290	27982741	2748595	5869626230	59,70%	90,18%	99,52%
inst20	25383590	16796368	4980795	4895060872	33,83%	70,35%	99,66%

Tabela 6.3: Comparação entre ILS e heurísticas construtivas com $k = 5$.

K=6							
Inst/Alg	LPT	Lpt_1	ILS	KK	Gap 0/1	Gap 1/ILS	Gap KK/1
inst1	330130131	149128749	106800464	3408735057	54,83%	28,38%	95,63%
inst2	484471029	285078347	148665500	3153114876	41,16%	47,85%	90,96%
inst3	151272212	61719896	36886563	5342568635	59,20%	40,24%	98,84%
inst4	435992281	218937314	89874741	4805673494	49,78%	58,95%	95,44%
inst5	255194773	180000607	57831137	7665955101	29,47%	67,87%	97,65%
inst6	199294002	25538362	16717269	7348838908	87,19%	34,54%	99,65%
inst7	166844782	53207785	44781556	1710733944	68,11%	15,84%	96,89%
inst8	202027224	119249485	65082322	5968101693	40,97%	45,42%	98,00%
inst9	184125469	139023597	15798486	6522418685	24,50%	88,64%	97,87%
inst10	179668093	134896753	13840334	7106364514	24,92%	89,74%	98,10%
inst11	62264721	26486948	20656301	7428669456	57,46%	22,01%	99,64%
inst12	259426183	102492483	79138836	6798570491	60,49%	22,79%	98,49%
inst13	213261381	171954956	100335139	6671863756	19,37%	41,65%	97,42%
inst14	190792175	179858672	75172828	4217015720	5,73%	58,20%	95,73%
inst15	159264611	69743775	26307641	1063440884	56,21%	62,28%	93,44%
inst16	33700662	15527663	12828067	5719228493	53,92%	17,39%	99,73%
inst17	27833906	21804875	18812342	5725885831	21,66%	13,72%	99,62%
inst18	80307181	35332179	9745616	9482468974	56,00%	72,42%	99,63%
inst19	87966387	55419045	24435532	6291892596	37,00%	55,91%	99,12%
inst20	53276421	31323019	4980669	5173001101	41,21%	84,10%	99,39%

Tabela 6.4: Comparação entre ILS e heurísticas construtivas com $k = 6$.

	KK × ILS				Lpt_1 × ILS			
Inst/ k	k=3	k=4	k=5	k=6	k=3	k=4	k=5	k=6
inst1	95,92%	97,89%	96,83%	96,87%	71,58%	61,91%	60,62%	28,38%
inst2	99,93%	95,93%	96,51%	95,29%	93,25%	61,22%	68,19%	47,85%
inst3	99,86%	99,92%	99,64%	99,31%	89,64%	86,10%	58,54%	40,24%
inst4	**70,52%**	87,39%	95,43%	98,13%	51,45%	65,91%	16,37%	58,95%
inst5	99,99%	99,94%	99,79%	99,25%	88,17%	80,97%	71,77%	67,87%
inst6	99,41%	96,77%	98,98%	99,77%	60,97%	35,14%	54,87%	34,54%
inst7	99,06%	99,88%	99,30%	97,38%	93,98%	89,39%	87,74%	15,84%
inst8	99,82%	99,93%	98,97%	98,91%	96,39%	57,31%	20,06%	45,42%
inst9	98,68%	99,72%	99,72%	99,76%	76,66%	68,34%	42,57%	88,64%
inst10	99,93%	99,40%	99,12%	99,81%	96,74%	43,26%	71,12%	89,74%
inst11	99,99%	100,00%	99,85%	99,72%	89,96%	99,03%	56,94%	22,01%
inst12	98,98%	99,37%	98,47%	98,84%	59,10%	35,45%	38,86%	22,79%
inst13	99,99%	99,82%	99,14%	98,50%	97,56%	80,70%	**4,02%**	41,65%
inst14	99,87%	93,91%	96,75%	98,22%	97,92%	63,15%	26,29%	58,20%
inst15	96,87%	99,06%	99,22%	97,53%	87,36%	12,85%	39,07%	62,28%
inst16	99,99%	99,95%	99,95%	99,78%	99,82%	91,15%	85,29%	17,39%
inst17	99,97%	99,49%	99,16%	99,67%	93,17%	53,22%	43,72%	13,72%
inst18	99,97%	99,81%	99,66%	99,90%	94,87%	62,41%	19,54%	72,42%
inst19	99,95%	99,62%	99,95%	99,61%	94,58%	39,19%	90,18%	55,91%
inst20	100,00%	100,00%	99,90%	99,90%	93,01%	97,12%	70,35%	84,10%

Tabela 6.5: Comparação entre metaheurística ILS, Heurística de Karmarkar-Karp e Heurística Lpt_1.

HKK vs ILS (Quartil inferior)				Lpt_1 vs ILS (Quartil superior)			
K=3	K=4	K=5	K=6	K=3	K=4	K=5	K=6
99,04%	98,77%	98,85%	98,20%	95,25%	82,25%	70,54%	63,68%

Tabela 6.6: Comparação entre quartis do $Gap()$ das instâncias para cada valor de k.

6.2 Algoritmos Exatos

Esta Seção mostra a comparação entre o desempenho do Algoritmo Híbrido CGA+ILS, descrito pelo Algoritmo 5.8, e o desempenho do Algoritmo CGA, descrito pelo Algoritmo 4.12, para a solução do problema MWNPP.

As Tabelas 6.7, 6.8 e 6.9 seguem a mesma lógica das anteriores. A variável inst[i] indica a i-ésima instância: de inst1 até inst5, n=100; de inst6 até inst10, n=200; de inst11 até inst15, n=400 e de inst16 até inst20, n=800. O número $k \in \{3, 4, 5, 6\}$ indica a quantidade de partes da partição. A primeira coluna contém as instâncias do experimento. Executou-se cada instância com um tempo limite de uma hora (3600 segundos).

As medidas de avaliação de desempenho apresentadas são:

- # nós : número de nós explorados até a finalização do algoritmo ou dentro de uma hora de execução;
- z^* :menor limite superior encontrado pelo algoritmo.

Nas soluções destas tabelas, tanto o CGA quanto o CGA+ILS usam a solução do ILS nas Tabelas 6.1, 6.2, 6.3 e 6.4 como solução inicial.

Tanto o CGA quanto o CGA+ILS resolveram todas as instâncias de tamanho 100 dentro do tempo limite. Nas demais instâncias, os algoritmos não convergiram para o ótimo pois o tempo limite se esgotou. O CGA+ILS se mostra melhor, no sentido de explorar cerca de 100 vezes menos nós em todas as instâncias do que justificariam as suas modificações em relação à versão original e obter um limite superior com o valor cerca de 4 vezes menor com o mesmo tempo de execução que o algoritmo concorrente no presente estudo. Isto significa que o CGA+ILS tem nós computacionalmente mais caros, mas possui uma velocidade de convergência maior. Em linhas gerais, quando o tempo de 3600 segundos é suficiente para resolver o problema, o CGA+ILS entrega a resposta mais rapidamente que o CGA. Quando o tempo limite se esgota antes do ótimo ser encontrado, o limite superior do CGA+ILS é menor que o do CGA.

A medida que o valor de k aumenta, o problema fica, naturalmente mais difícil para ambos os métodos em proporções muito parecidas como discutido na Seção 2.1 no Capítulo 2. Nesse aspecto, não existe diferença significativa entre os algoritmos quando o valor de k varia entre 3 e 6.

A Tabela 6.7 mostra em quais instâncias e em quanto os limites superiores do CGA+ILS são menores que os do CGA . A Tabela 6.8 apresenta o número de nós explorados e limitantes encontrados pelo CGA.

CGA vs CGA+ILS				
Inst/k	K=3	K=4	K=5	K=6
inst1	0,00%	0,00%	0,00%	0,00%
inst2	0,00%	0,00%	0,00%	0,00%
inst3	0,00%	0,00%	0,00%	0,00%
inst4	0,00%	0,00%	0,00%	0,00%
inst5	0,00%	0,00%	0,00%	0,00%
inst6	59,95%	60,00%	60,00%	60,00%
inst7	39,92%	39,96%	40,00%	40,00%
inst8	49,96%	50,00%	50,00%	50,00%
inst9	49,83%	49,99%	50,00%	50,00%
inst10	39,96%	40,00%	40,00%	40,00%
inst11	0,00%	9,80%	10,00%	10,00%
inst12	29,99%	30,00%	30,00%	30,00%
inst13	19,84%	20,00%	20,00%	20,00%
inst14	49,87%	50,00%	50,00%	50,00%
inst15	29,79%	29,99%	30,00%	30,00%
inst16	0,00%	9,91%	9,98%	10,00%
inst17	39,85%	39,99%	40,00%	40,00%
inst18	59,91%	59,90%	60,00%	60,00%
inst19	19,57%	19,99%	19,99%	20,00%
inst20	0,00%	9,58%	9,99%	9,99%

Tabela 6.7: Tabela com o $Gap(CGA/CGA+ILS)$ mostrando o CGA+ILS retorna respostas estritamente menores que as do CGA quando o método não converge antes de 3600 segundos.

CGA								
Inst/ k	K=3		K=4		K=5		K=6	
	z*	# nós	z*	# nós	z*	# nós	z*	# nós
inst1	29654	52341140	54701	64212878	101297	87196118	87221	98236577
inst2	12678	43188174	49283	65193460	212794	86706868	90341	98858106
inst3	6785	42873090	11204	74956403	175301	86412524	68774	101324634
inst4	11206	42603548	60315	76127913	163406	86600724	73982	100017977
inst5	35743	53680891	55497	64937019	200567	86519874	56033	99306585
inst6	193028	3268807365	2487907	3140003786	31304646	3711721837	5201972	3647926264
inst7	129455	3137376254	255080	3629980566	2207790	3699049586	18154991	3757154875
inst8	269345	3513097823	2298292	3820969827	14192063	3143551316	29655509	3639531813
inst9	61049	3929630096	1277764	3338837476	6166209	3841882373	6988464	3662224462
inst10	312385	3362539838	7062017	3785225439	16893493	3198371770	7820211	3124283497
inst11	9005	1888867514	52857	1473551100	2455587	1734909781	7966808	1079362164
inst12	972503	1852911928	3420533	1591325936	77276597	1295532516	54558844	1514165541
inst13	69072	1577157032	4177078	1322120335	3353472	1731897859	38725972	1764595235
inst14	84548	1856650184	5936640	1491866368	5185834	1872287281	13319079	1060782653
inst15	47001	1239338626	1285865	1600100617	6823108	1648566975	4352568	1373907032
inst16	625	847465743	116347	789589120	478041	879540720	4382256	553609095
inst17	68567	849951062	1142435	734054363	15756977	654964077	15264867	606544857
inst18	118072	792279709	104538	747749867	18145485	867242417	4112745	568813368
inst19	23785	799959238	821852	800164170	787553	959824883	11213751	680916084
inst20	1356	856990876	27162	801295462	2217848	797684814	1832947	604089616

Tabela 6.8: Menor limite superior histórico e número de nós explorados pelo CGA.

CGA+ILS								
Inst/ k	K=3		K=4		K=5		K=6	
	z*	# nós	z*	# nós	z*	# nós	z*	# nós
inst1	29654	513148	54701	629538	101297	854863	87221	963103
inst2	12678	415270	49283	626860	212794	833719	90341	950558
inst3	6785	389755	11204	681421	175301	785568	68774	921133
inst4	11206	405748	60315	725027	163406	824768	73982	952552
inst5	35743	521173	55497	630456	200567	839998	56033	964141
inst6	77312	32364429	995263	31089146	12521959	36749721	2080889	36118081
inst7	77778	29879773	153153	34571243	1324779	35229043	10893099	35782427
inst8	134778	33142432	1149252	36046885	7096137	29656144	14827860	34335205
inst9	30631	36725514	638989	31204088	3083211	35905442	3494339	34226396
inst10	187542	30293151	4237321	34101130	10136206	28814160	4692237	28146698
inst11	9005	17989214	47679	14033820	2210136	16522950	7170235	10279639
inst12	680861	16999192	2394482	14599320	54093726	11885619	38191299	13891426
inst13	55366	14469330	3341771	12129544	2682886	15888971	30980886	16188947
inst14	42382	17191205	2968428	13813577	2593025	17335993	6659647	9822061
inst15	33001	12270679	900206	15842580	4776276	16322445	3046898	13603039
inst16	625	8227822	104821	7665913	430345	8539230	3944139	5374845
inst17	41246	8018406	685567	6925041	9454292	6178906	9159026	5722121
inst18	47333	7545521	41920	7121427	7258299	8259451	1645203	5417270
inst19	19130	7842737	657583	7844746	630144	9410047	8971102	6675647
inst20	1356	8009260	24561	7488742	1996179	7454998	1649768	5645697

Tabela 6.9: Menor limite superior histórico e número de nós explorados pelo CGA+ILS.

Capítulo 7

Conclusão

Esta dissertação de mestrado apresentou um estudo da família de Problemas de Partição de Números, com foco principal no Problema da k-Partição de Números (MWNPP).

Propôs-se uma formulação matemática do MWNPP baseada em dois problemas semelhantes da literatura, quais seja, o Problema do Empacotamento (BPP) e o Problema de Escalonamento em Máquinas Idênticas (IMSP). Eliminando a multiplicidade de soluções do modelo de programação inteira, possibilitou-se um maior entendimento da ramificação utilizada nos métodos exatos e a identificação de um melhor limitante inferior que o presente na literatura, apresentado em Korf (1998). O limite inferior de varredura da árvore de busca descrito na presente dissertação cresce mais rapidamente a cada nível descido na árvore de busca, pois considera os elementos não alocados com variáveis reais e as somas dos alocados em cada parte com variáveis binárias, tendo uma solução do problema relaxado em $O(k)$.

Provou-se a NP-completude do problema estudado com uma extensão da ideia apresentada em Karp (1972). A ideia da demonstração é simples, mesmo que sua formalização pareça um pouco mais complicada. Explorou-se a relação entre problemas de decisão e problemas de otimização, para exibir algoritmos polinomiais teóricos, desde que exista um algoritmo de decisão $O(1)$ para o mesmo.

Estudou-se algoritmos heurísticos, aproximados e exatos da literatura, aplicáveis com o intuito de combinar ideias de diferentes paradigmas de solução de problemas. O produto inicial deste trabalho foi uma adaptação da meta-heurística ILS ao MWNPP, de modo que o algoritmo fosse rápido e significativamente melhor que os demais métodos usados para obtenção de limites superiores em um algoritmo *Branch-and-Bound* presente na literatura. Os critérios de escolha de parada e movimentos usam ideias do Algoritmo 4.2, proposto em Finn e Horowitz (1979), e estratégias com estrutura de dados propostas em Schroeppel e Shamir (1981). Graças a essa fundamentação, o ILS se mostrou superior a todas as heurísticas construtivas da literatura, encontrando respostas, no pior caso, 3 vezes menores que a Heurística de Karmarkar-Karp (KK).

Propôs-se uma série de melhorias no *Complete Greedy Algorithm* (CGA), apresentado no Algoritmo 4.12 e proposto em Korf (1998). As modificações que refinam os limitantes do método, usando a relaxação do modelo matemático mostrado nas Expressões (2.25)-(2.29), e o *Iterated Local Search* (ILS), na forma proposta, aumentaram a complexidade de exploração de cada nó do Algoritmo CGA+ILS em relação

ao CGA, mas reduziram a necessidade de exploração de nós para convergência do método. O critério de busca do Algoritmo 5.7 permitiu a antecipação de cortes em troca de um maior custo de inserção e retirada. Falseou-se a hipótese de exatidão da solução do ILS por meio de CGA e CGA+ILS em todas as instâncias testadas. Por fim, concluiu-se que o CGA+ILS convergiu para resposta melhores, não exatas, em todas as instâncias, com exceção daquelas de tamanho 100, na qual ambos alcançaram a otimalidade. Porém, o CGA+ILS conseguiu este resultado explorando menor número de nós em relação ao aumento de complexidade das melhorias inseridas.

7.1 Considerações Finais

A proposta de uso de meta-heurísticas em métodos exatos se mostra promissora, no sentido de acelerar a busca de soluções exatas. A análise do tempo de execução não consta nos resultados, pois a contagem de operações fundamentais do algoritmo é uma metodologia que torna a comparação futura desse trabalho com outros independente do *hardware* adotado. Uma contrapartida do uso de uma fila de prioridades na busca dos métodos exatos pode ser o aumento do tamanho da fila, mesmo com a antecipação de cortes. Para um planejamento de experimentos mais adequado com análise estatística, regressão e correlação, necessita-se de um maior volume de dados colhidos, maior tempo de computação, separação de cada modificação realizada para se verificar o efeito individual das melhorias sobre a solução encontrada, uso de memória e complexidade do algoritmo.

7.2 Trabalhos Futuros

Como trabalhos futuros, tem-se a formulação de uma função objetivo que mantenha o problema equivalente e, em cada subproblema associado a um nó, retorne uma sequência de limites inferiores que se aproximem mais rapidamente dos seus respectivos limitantes superiores.

A coleta e análise de dados sobre a execução do algoritmo em cada instância teste ainda pode incluir dados como:

- Tamanho máximo atingido pela fila para encontrar a solução exata de uma instância. Este dado ajuda a decidir se o método deve usar fila comum, fila de prioridades ou uma pilha;
- Tamanho da fila em função do número de nós explorados. Este dado descreve o comportamento do número de nós ativos, diretamente ligados à convergência do método, e serve como medida de uso de memória em função do tempo;
- Menor limite superior encontrado e menor limite inferior ativo em função do número de nós explorados. Este dado mostra uma curva decrescente de limites superiores e uma curva crescente de limites inferiores se aproximando à medida que os nós da árvore de busca crescem;
- O número médio de nós explorados para uma instância de tamanho n e um k fixo. Estes dados auxiliam na estimativa da complexidade do algoritmo;

- O número médio do tamanho máximo da fila para uma instância de tamanho n e um k fixo. Estes dados auxiliam na estimativa do custo de memória do algoritmo;

- Regressão exponencial aplicada aos dados acima para estimar o custo de memória e a complexidade de tempo. Um método *Branch and Bound*, implementado com busca em largura ou busca prioritária, ainda tem complexidade exponencial de tempo e memória. Mesmo que seja impossível descobrir sua complexidade exata, estes dados auxiliam na estimativa da base da função exponencial que descreve a complexidade do algoritmo.

Ainda não existe a demonstração de que o algoritmo 5.5 retorna a solução exata de qualquer subproblema relaxado associado ao nó segundo o modelo matemático (2.25)-(2.29). Este é um trabalho em andamento.

Outro possível trabalho futuro seria obter a volta direta da redução apresentada $MWNPP \leq_p (TWNPP)$.

Referências Bibliográficas

Ausiello, G.; Crescenzi, P.; Kann, V.; Gambosi, G.; Spaccamela, A. M. e Protasi, M. (2003)a. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Capítulo Sequential Algorithms for Partitioning Problems, p. 50–60. Springer-Verlag New York, Inc., 1st edição.

Ausiello, G.; Crescenzi, P. e Protasi, M. (1995). Approximate solution of NP optimization problems. *Theoretical Computer Science*, v. 150, n. 1, p. 1–55.

Ausiello, Giorgio; Crescenzi, Pierluigi; Gambosi, Giorgio; Kann, Viggo; Marchetti-Spaccamela, Alberto e Protasi, Marco. (2003)b. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, corrected edition edição.

Berretta, Regina e Moscato, Pablo. (1999). The number partitioning problem: An open challenge for evolutionary computation? Corne, David; Dorigo, Marco; Glover, Fred; Dasgupta, Dipankar; Moscato, Pablo; Poli, Riccardo e Price, Kenneth V., editors, *New Ideas in Optimization*, p. 261–278. McGraw-Hill, Maidenhead, England, UK.

Crescenzi, Pierluigi; Kann, Viggo; Halldórsson, Magnús; Karpinski, Marek e Woeginger, Gerhard. A compendium of NP optimization problems. URL: https://www.nada.kth.se/ viggo/problemlist/compendium.html, July(1997).

Ferreira, Fernando Fagundes. *Análise estatística do problema da partição numérica*. Tese de Doutorado, Universidade de São Paulo, (2001).

Finn, Greg e Horowitz, Ellis. (1979). A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics*, v. 19, n. 3, p. 312–320.

Garey, Michael R. e Johnson, David S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Gent, Ian P. e Walsh, Toby. (1995). The number partition phase transition. Technical Report RR-95-185, Department of Computer Science, University of Strathclyde, Glasgow, Scotland.

Gent, Ian P. e Walsh, Toby. (1998). Analysis of heuristics for number partitioning. *Computational Intelligence*, v. 14, n. 3, p. 430–451.

Graham, Ronald L. (1966). Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, v. XLV, n. 9, p. 1563–1581.

Graham, Ronald L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, v. 17, n. 2, p. 416–429.

Griffiths, Martin e Mezo, István. (2010). A generalization of Stirling numbers of the second kind via a special multiset. *Journal of Integer Sequences*, v. 13, n. 2, p. 3.

Hayes, Brian. (2002). Computing science: The easiest hard problem. *American Scientist*, v. 90, p. 113–117.

Horowitz, Ellis e Sahni, Sartaj. (1974). Computing partitions with applications to the knapsack problem. *Journal of the ACM*, v. 21, n. 2, p. 277–292.

Joosten, Sebastiaan J. C. e Zantema, Hans. (2013). Relaxation of 3-partition instances. Cornelissen, Kamiel; Hoeksma, Ruben; Hurink, Johann e Manthey, Bodo, editors, *12th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, Enschede, Netherlands, May 21-23, 2013*, volume WP 13-01 of *CTIT Workshop Proceedings*, p. 133–136, (2013).

Karmarkar, Narendra e Karp, Richard M. (1982). The differencing method of set partition. Report UCB/CSD 81/113, Computer Science Division, University of California, Berkeley, CA.

Karp, Richard M. (1972). Reducibility among combinatorial problems. Miller, Raymond E.; Thatcher, James W. e Bohlinger, Jean D., editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972*, p. 85–103, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA. Springer US.

Kojić, Jelena. (2010). Integer linear programming model for multidimensional two-way number partitioning problem. *Computers & Mathematics with Applications*, v. 60, n. 8, p. 2302–2308.

Korf, Richard E. (1998). A complete anytime algorithm for number partitioning. *Artificial Intelligence*, v. 106, n. 2, p. 181–203.

Korf, Richard E. (2009). Multi-way number partitioning. *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, p. 538–543, (2009).

Korf, Richard E. (2010). Objective functions for multi-way number partitioning. *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS 2010)*, (2010).

Korf, Richard E. (2011). A hybrid recursive multi-way number partitioning algorithm. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, p. 591–596, (2011).

Korf, Richard E.; Schreiber, Ethan L. e Moffitt, Michael D. (2014). Optimal sequential multi-way number partitioning. *International Symposium on Artificial Intelligence and Mathematics (ISAIM-2014)*, (2014).

Korte, Bernhard; Vygen, Jens; Korte, B e Vygen, J. (2012). *Combinatorial optimization*, volume 2. Springer.

Land, A. H. e Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, v. 28, n. 3, p. 497–520.

Langston, Michael A. (1982). Improved 0/1-interchange scheduling. *BIT Numerical Mathematics*, v. 22, n. 3, p. 282–290.

Lima, Elon Lages. (2004). *Análise real*. IMPA, Rio de Janeiro, Brasil.

Lourenço, Helena R.; Martin, Olivier C. e Stützle, Thomas. (2003). Iterated local search. Glover, Fred e Kochenberger, Gary A., editors, *Handbook of Metaheuristics*, p. 320–353. Springer US, Boston, MA.

Mertens, Stephan. (2006). The easiest hard problem: Number partitioning. Percus, A.G.; Istrate, G. e Moore, C., editors, *Computational Complexity and Statistical Physics*, p. 125–139, New York. Oxford University Press.

Michiels, Wil; Korst, Jan; Aarts, Emile e others,. (2003). Performance ratios for the Karmarkar-Karp differencing method. *Electronic Notes in Discrete Mathematics*, v. 13, p. 71–75.

Moffitt, Michael D. (2013). Search strategies for optimal multi-way number partitioning. *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, p. 623–629. AAAI Press, (2013).

Pedroso, João Pedro e Kubo, Mikio. (2010). Heuristics and exact methods for number partitioning. *European Journal of Operational Research*, v. 202, n. 1, p. 73–81.

Pop, Petrică C e Matei, Oliviu. (2013). A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem. *Applied Mathematical Modelling*, v. 37, n. 22, p. 9191–9202.

Schreiber, Ethan L. *Optimal Multi-Way Number Partitioning*. PhD thesis, University of California Los Angeles, (2014).

Schreiber, Ethan L. e Korf, Richard E. (2014). Cached iterative weakening for optimal multi-way number partitioning. *Proceedings of the Twenty-Eighth Annual Conference on Artificial Intelligence (AAAI-14)*, p. 2738–2745, (2014).

Schroeppel, Richard e Shamir, Adi. (1981). A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM journal on Computing*, v. 10, n. 3, p. 456–464.

Sipser, Michael. (2006). *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston.

Sloane, NJA. On-Line Encyclopedia of Integer Sequences. https://oeis.org/, (1991).

Stanley, Richard P. (1997). *Enumerative Combinatorics. Vol. 1, vol. 49 of Cambridge Studies in Advanced Mathematics*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2nd edição.