COMPARAÇÃO ENTRE HEURÍSTICAS CONSTRUTIVAS E ILS NA SOLUÇÃO DO PROBLEMA DA K-PARTIÇÃO DE NÚMEROS

Alexandre Frias Faria
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas 7675, 30510-000 – Nova Gameleira – Belo Horizonte – MG – Brasil
alexandrefrias1@hotmail.com

Sérgio Rocardo de Souza
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas 7675, 30510-000 – Nova Gameleira – Belo Horizonte – MG – Brasil
sergio@dppg.cefetmg.br

Carlos Alexandre Silva
Instituto Federal de Minas Gerais (IFMG)
Av. Serra da Piedade 299, 34515-640 – Morada da Serra – Sabará – MG – Brasil
carlos.silva@ifmg.edu.br

RESUMO

Este artigo apresenta um algoritmo para a versão de otimização do Problema da k-Partição de Números (k-PPN), que consiste em distribuir os elementos de uma sequência dada em k subconjuntos disjuntos, de modo que as somas dos elementos de cada subconjunto fiquem no menor intervalo possível. A meta-heurística *Iterated Local Search* (ILS), adaptada para a solução do k-PPN, tem desempenho satisfatório para problemas com menos de seis subconjuntos, quando comparada aos métodos construtivos intensificados por uma enumeração inicial de elementos aplicada à heurística construtiva *Longest Processing Time first* (LPT). A principal vantagem desse método é que o desempenho do algoritmo pode ser melhorado quando a solução inicial é mais próxima do ótimo.

PALAVRAS CHAVE. Problema da Partição de Números. *Iterated Local Search*. Heurística de Karmarkar-Karp.

Tópicos: OC - Otimização Combinatória. MH - Metaheuristicas.

ABSTRACT

This paper presents an algorithm for the optimizing version of Multi-Way Number Partitioning Problem (k-PPN). This problem consists in distributing the elements of a given set into k disjoint subsets so that the sum of each subset elements fits in the shortest interval. The metaheuristics Iterated Local Search (ILS) adapted for solving the k-PPN, has satisfactory performance with less than six subsets compared to constructive methods enhanced by an initial list of elements applied to constructive heuristic Longest Processing Time first (LPT). The main advantage of the presented method is that the algorithm performance can be improved when the initial solution is closer to the optimum.

KEYWORDS. Combinatorial Optimization. Number Partitioning Problem. Metaheuristic.

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 é uma partição tendo exatamente k subconjuntos não vazios. Durante o decorrer do texto, os subconjuntos pertencentes à partição são denominados de partes. A primeira generalização do Problema da Partição de Números (PPN) é o Problema da k-Partição de Números (k-PPN), que expande o número de subconjuntos nos quais se pretende distribuir os elementos da sequência V (particionar os índices dos elementos de V). Dada uma sequência numérica V, o objetivo é encontrar uma partição de cardinalidade k para seus índices, 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.

Existe uma ampla literatura sobre a solução do Problema da Partição de Números (PPN) e suas variações. Ele foi listado de modo formal em Karp [1972] e Garey e Johnson [1979] como um dos 6 problemas NP-completos básicos. Ambos demonstram uma série de equivalências entre o PPN e outros problemas NP-completos. O k-PPN aparece explicitamente em um artigo sobre a análise de uma heurística construtiva denominada *Differencing Method*, mais conhecida como Heurística de Karmarkar-Karp (HKK), 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 k-PPN até hoje. Os autores demonstram vários resultados 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 $\frac{4}{3} - \frac{1}{3(k-1)}$ para HKK. Este resultado impõe um intervalo de erro limitado aos valores obtidos com HKK, sendo similar àquele encontrado pelo algoritmo *Longest Processing Time* (LPT), proposto em Graham [1969] e discutido na Seção 4 deste artigo.

Existe, também, um ramo de estudo do PPN 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 PPN e seus problemas relativos, 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 e isso produziria um critério de poda por otimalidade global em um método exato. Gent e Walsh [1995] mostra a existência de transição de fases para identificar as características de uma instância livre de partições perfeitas. Já Mertens [2006] promove um avanço no trabalho de Gent e Walsh [1995], mostrando uma nova expressão para a classificação de instâncias quase certamente livres de partições perfeitas.

Gent e Walsh [1998] e Berretta e Moscato [1999] mostram que o PPN é um problema muito difícil de ser solucionado por meta-heurísticas de uso geral, como Algoritmos Genéticos, *Simulated Annealing* e outros. Em muitos casos, esses métodos perdem em tempo e desempenho para HKK e até mesmo para um algoritmo guloso como LPT. Textos mais recentes, como Kojić [2010] e Pop e Matei [2013], usam várias classes de meta-heurísticas em versões de maior complexidade da família de Problemas da Partição de Números, como o Problema da Partição de Números Multidimensional (PPNM) e o Problema da Múltipla Partição de Números Multidimensional (PMPNM), mas também não mostram resultados muito significativos, exceto pela formulação matemática.

Tanto para o PPN quanto para o k-PPN sempre houve a possibilidade de realizar a conversão para o Problema da Mochila [Horowitz e Sahni, 1974] [Schroeppel e Shamir, 1981], mas o espaço de memória requerido é inviável para problemas de maior cardinalidade. Uma nova abordagem surgiu quando Korf [1998] propôs a construção de novos algoritmos exatos, fazendo um procedimento *Backtrack* em heurísticas construtivas como LPT e HKK, chamadas de *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 k-PPN 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 k-PPN de uma partição de tamanho $k-1$ para k, Moffitt [2013] propõem 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 em [Korf et al., 2013], e no algoritmo *Cached Iterative Weakening*, apresentado em [Schreiber e Korf, 2014]. Uma análise completa e de grande interesse sobre esses algoritmos encontra-se em [Schreiber, 2014].

O objetivo do presente artigo é comparar a solução do Problema da K-Partição de Números (k-PPN) pela aplicação da meta-heurística *Iterated Local Search* (ILS) com a solução do mesmo problema pelo uso de heurísticas construtivas da literatura, intensificadas por uma enumeração parcial da solução. Está, então, organizado como segue. A Seção 2 apresenta um enunciado formal do k-PPN, exemplifica uma pequena instância do k-PPN e mostra o comportamento do espaço de soluções. A Seção 3 mostra uma proposição de formulação matemática para o k-PPN. A Seção 4 apresenta os algoritmos da literatura usados na comparação e suas devidas intensificações. A Seção 5 descreve a aplicação do *Iterated Local Search* (ILS) para a solução do k-PPN, proposta neste artigo. A Seção 6 demonstra todos os resultados computacionais e materiais usados para gerá-los. Por fim, a Seção 7 comenta os resultados e propõe trabalhos futuros.

2. Descrição do k-PPN

Esta seção apresenta a definição do k-PPN e um exemplo desse problema.

Definição 2.1. *Seja $V = \{v_1, v_2, \ldots, v_n\}$ uma sequência de inteiros positivos, 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:*

$$f(\{A_1, A_2, \ldots, A_k\}) = \max_{j'} \left\{ \sum_{i \in A_{j'}} v_i \right\} - \min_{j} \left\{ \sum_{i \in A_j} v_j \right\} \tag{1}$$

O Exemplo 2.1 a seguir mostra uma situação desse problema.

Exemplo 2.1. *O conjunto $S = \{87, 6, 5, 45, 34, 2, 24, 12, 7, 6, 54, 34\}$ pode ser particionado para $k = 3$ em:*

$$\underbrace{\{87, 12, 6\}}_{soma=105}, \underbrace{\{45, 34, 6, 24\}}_{soma=109}, \underbrace{\{5, 7, 2, 54, 34\}}_{soma=102}$$

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

$$\underbrace{\{87, 12, 6\}}_{soma=105}, \underbrace{\{45, 34, 2, 24\}}_{soma=105}, \underbrace{\{5, 7, 6, 54, 34\}}_{soma=106}$$

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

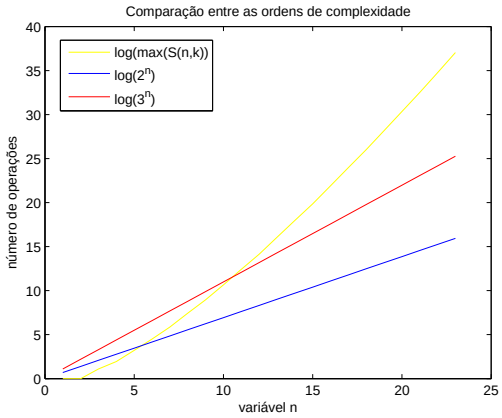
A dificuldade combinatória desse problema, quando resolvido por uma enumeração ingênua das k-partições de V, é dada pelo Número de Stirling $S(n, k)$[1], que conta a quantidade de formas distintas de se repartir um conjunto de tamanho n em k partes. Sua fórmula é definida recursivamente por:

$$S(n, k) = S(n-1, k-1) \; + \; k \, S(n-1, k) \tag{2}$$

[1] Griffiths e Mezo [2010] mostra uma análise detalhada do Número de Stirling.

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, de modo que a ordem de complexidade do algoritmo seria $O\left(n.S(n,k)\right)$.

Figura 1: O Maior dos Números de Stirling vs Funções Exponenciais. Dados retirados de [Sloane, 1991].



A Figura 1 mostra que o Maior Número de Stirling domina assintoticamente funções exponenciais, ou seja, para todo $k > c > 1$, existe n_0 suficientemente grande tal que $S(n, a(n)) > c^n \ : \ \forall n > n_0$.

3. Formulação Matemática para o k-PPN

O modelo matemático de Otimização Linear Inteira proposto neste artigo para o k-PPN usa duas variáveis inteiras para representar o limite inferior t_1 e o limite superior t_2 da soma dos elementos das k partes. As demais variáveis x_{ij} são binárias e informam se um elemento está ou não numa parte. Defini-se $I_m = \{y \in \mathbb{Z} : \quad 1 \leq y \leq m\}$.

$$\min \quad t_2 - t_1 \tag{3}$$

$$\text{suj. a} \quad t_1 \leq \sum_{i=1}^{n} v_i x_{ji} \leq t_2, \quad \forall j \in I_k \tag{4}$$

$$\sum_{j=1}^{k} x_{ji} = 1, \quad \forall i \in I_n \tag{5}$$

$$x_{ji} = 0, \quad \forall (i,j) : i + j \geq n + 2 \tag{6}$$

$$t_1, t_2 \in \mathbb{Z}_+ \tag{7}$$

$$x_{ji} \in \{0,1\}, \quad \forall (j,i) \in I_k \times I_n \tag{8}$$

A expressão (3) mostra o objetivo do problema, ou seja, a minimização do intervalo que contém todas as k somas dos elementos das partes. As k inequações (4) mostram que o modelo garante que cada partição é não vazia, pois $t_1 > 0$, por efeito da expressão (7) e, também, que estão todas contidas no intervalo $[t_1, t_2]$. As n equações indicadas pela expressão (5) garantem que as partições 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 igualdades 6 indicam que a ordem das partes não altera a solução. A relação (7) informa que os limitantes da maior e menor soma dos elementos das partições é sempre positiva. Por fim, a relação de pertinência (8) indica que $x_{ij} = 1$ se o elemento de índice i da sequencia V pertence à parte de índice j e $x_{ij} = 0$, caso contrário.

4. Heurísticas construtivas para a solução do k-PPN

Proposto por Graham [1966, 1969], o algoritmo guloso *Longest Processing Time first* (LPT) foi desenvolvido para o Problema de Escalonamento de Tarefas. Estes problemas se diferenciam do k-PPN, 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 $\max\limits_{1\leq i\leq k}\left\{\sum\limits_{x\in A_i} x\right\}$. Isso significa que, se $OPT(S)$ é o valor ótimo da função objetivo e $LPT(S)$ o valor de função objetivo do LPT para uma instância S, a razão entre estes valores é tal que:

$$\frac{LPT(S)}{OPT(S)} \leq \frac{4}{3} - \frac{1}{3k} \tag{9}$$

Exemplo 4.1. *O resultado ótimo do Problema da k-Partição de Números para a instância:*

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

e $k = 2$ *é*

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

A parte com a maior soma é $max\{8 + 7,\ 6 + 5 + 4\} = 15$. *Substituindo* $k = 2$ *na desigualdade 9 tem-se* $\left(\frac{4}{3} - \frac{1}{6}\right)\ 15 = \frac{7}{6}\ 15 = 17.5$.*Portanto, a resposta do algoritmo (LPT) sempre é uma partição em que a maior soma é menor que* 17.5 *como:*

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

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

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

O Algoritmo LPT está apresentado no Algoritmo 1. Consiste em ordenar a 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 $O(nlog(n))$ para ordenar a entrada e um custo $O(n)$ para alocar os elementos nas k partes. Finalmente, a complexidade do método é $O(nlog(n))$.

Algoritmo 1: *Longest Processing Time* para k máquinas: LPT

Entrada: Conjunto S de números inteiros e inteiro k.
Saída: Partição do conjunto S, $\{A_1, A_2, ..., A_k\}$, com tamanho k
início
  E **para** $j \in \{1, ..., k\}$ **faça**
    $A_j = \emptyset$;
    $L_j = 0$;
  fim
  Ordene S em ordem decrescente;
  para $a \in S$ **faça**
    $l = \arg\min_j L_j$;
    $A_l = A_l \cup a$;
    $L_l = L_l + a$;
  fim
  retorna $\{A_1, A_2, ..., A_k\}$
fim

O algoritmo Lpt_1, mostrado no Algoritmo 2, é uma variação do Algoritmo 1. Consiste em executar o LPT supondo que dois elementos de partes distintas foram pré-fixados na mesma parte. Sua ideia é uma rotina que lista qual é a resposta do LPT quando iniciado com o elemento v_i na posição mesma parte que v_j, ou seja, para cada elemento v_j que não está na parte de v_i na solução do LPT, resolve-se A nova sequência $V' = (v_1, ..., v_i + v_j, ..., v_n)$. Assim, o algoritmo Lpt_1 tem complexidade $O(n^2)$. Esse método garante que a resposta do Lpt_1 é sempre menor ou igual a resposta do LPT.

Algoritmo 2: Lpt_1: guarda as melhores soluções de n iterações do LPT

Entrada: Uma sequência V, k, $f()$
Saída: aux
início
  para $h = 1$ *até* n **faça**
    $sum_1 = v_h$, $a = v_h$ e $v_h = 0$;
    para $i = 1$ *até* n **faça**
      $l = \arg\min_j sum$;
      $A_l = A_l \cup i$;
      $sum_l = sum_l + v_i$;
    fim
    $v_h = a$ e $sum_1 = 0$;
    $ub = f(\{A_1, A_2, ..., A_k\})$;
    se $ub < aux$ **então**
      $aux = ub$;
    fim
  fim
  retorna aux
fim

A Heurística de Karmarkar-Karp, introduzida por Karmarkar e Karp [1982], é um método construtivo para o PPN que pode ser adaptado para o k-PPN. 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. Está apresentado no Algoritmo 3. Para uma implementação de baixo custo computacional, usando uma estrutura de dados `max-heap`, a complexidade desse algoritmo é $O(n.log(n))$. O Exemplo 4.2 mostra uma aplicação dessa heurística.

Algoritmo 3: Heurística de Karmarkar-Karp com $k = 2$ [Karmarkar e Karp, 1982]

Entrada: Um sequência V
Saída: Valor inteiro correspondendo à menor diferença entre as partes
início
  enquanto $|V| \neq 1$ **faça**
    $s_1 = removemax(V)$;
    $s_2 = removemax(V)$;
    $c = s_1 - s_2$;
    $insereordenado(V, c)$;
  fim
  retorna $v[0]$
fim

Exemplo 4.2. *Seja* $V = \{8, 7, 6, 5, 4\}$ *e* $k = 2$. *Então:*

$$\textit{iteração 1: } \{\mathbf{8}, \mathbf{7}, 6, 5, 4\} \Rightarrow \{6, 5, 4, 1\}$$
$$\textit{iteração 2: } \{\mathbf{6}, \mathbf{5}, 4, 1\} \Rightarrow \{4, 1, 1\}$$
$$\textit{iteração 3: } \{\mathbf{4}, \mathbf{1}, 1\} \Rightarrow \{3, 1\}$$
$$\textit{iteração 4: } \{\mathbf{3}, \mathbf{1}\} \Rightarrow \{2\}$$

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.*

A adaptação desse algoritmo para o k-PPN consiste em substituir a operação de diferença entre os dois maiores elementos do conjunto pelo Algoritmo 4 e seguir a mesma lógica de funcionamento do Exemplo 4.2. A diferença é que os elementos são vetores de dimensão k e não escalares. Logo, a estrutura de dados `max-heap` deve operá-los em ordem lexicográfica. O Exemplo 4.3

mostra essa nova situação, apresentando o funcionamento da Heurística de Karmarkar-Karp para $k > 2$, fazendo cada iteração em duas ou 3 fases para demonstrar a generalização desta Heurística. O pseudo-código usado para resolver o Exemplo 4.3 está no Algoritmo 5. A única mudança em relação à Heurística de Karmarkar-Karp original é a troca de $c = s_1 - s_2$ por $c = opera(s_1, s_2, k)$.

Algoritmo 4: $opera(x, y, k)$: Modificação na Heurística de Karmarkar-Karp para a solução do k-PPN.

Entrada: Dois vetores x e y ordenados e um inteiro k
Saída: Um vetor ordenado de tamanho k
início
  para $i = 0$ *até* $n - 1$ **faça**
    $s_i = x_i + y_{n-i}$;
  fim
  ordene s em ordem não crescente;
  para $i = 0$ *até* $n - 1$ **faça**
    $s_i = s_i + s_{n-1}$;
  fim
  retorna s
fim

Algoritmo 5: 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
início
  enquanto $|V| \neq 1$ **faça**
    $s_1 = removemax(V)$;
    $s_2 = removemax(V)$;
    $c = opera(s_1, s_2, k)$;
    $insereordenado(V, c)$;
  fim
  retorna $v[0]$
fim

Exemplo 4.3. *Seja* $V = \{8, 7, 6, 5, 4\}$ *e* $k = 3$. *Então:*

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

5. Aplicação do *Iterated Local Search* (ILS) à solução do k-PPN

Com a pior solução inicial possível para o k-PPN, ou seja, os $k - 1$ menores elementos $v_i \in V$ definindo, cada um, uma parte $A_j = \{v_i\} \quad \forall j \in \{1, 2, ...k - 1\}$ e a parte A_k é a soma dos $n - k + 1$ maiores elementos. O Algoritmo ILS proposto nesse trabalho usa propriedades particulares da função (1) como estratégia de perturbação da solução. O refinamento acontece pela busca de um elemento a ser realocado, de modo que, na parte de maior soma, busca-se um subconjunto de cardinalidade menor ou igual a dois cuja soma seja a mais próximo da metade do valor da função objetivo para a partição da iteração atual. O mesmo se faz na parte de menor soma, porém, buscando um elemento cuja diferença com outro da parte de maior soma se aproxime da metade do valor da função objetivo da iteração atual.

5.1. Movimento e Vizinhança

O movimento de realocação sempre encolhe a parte de maior soma e aumenta a parte de menor soma. 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 da solução s, na forma:

$$N(s) = \left\{ s' \ : \ s' \leftarrow s \oplus m_{i,j}, \ \ \forall i \neq j \in I_k \right\} \tag{10}$$

A dimensão da vizinhança é o produto da cardinalidade das partes $|A_k|.|A_1|$, supondo que as partes estejam ordenadas de forma crescente por suas somas. Essa operação é a perturbação da metaheurística ILS.

5.2. Estratégia de Realocação

A ideia principal do método é encontrar, dentre todos os elementos da parte de maior e menor soma, aqueles que mais reduzem o valor da função objetivo. Para isso, fixa-se b_l como os elementos da parte de menor soma, A_{min}, e a_l como os elementos da parte de maior soma, A_{max}. O Subproblema de Busca consiste em encontrar a solução de:

$$\max_{(i,j,l),(x_1,x_2,x_3)} \quad a_i x_1 + a_j x_2 - b_l x_3 \tag{11}$$

$$a_i x_1 + a_j x_2 - b_l x_3 \leq \frac{1}{2} \left(\sum_{x \in A_{\max}} x - \sum_{x \in A_{\min}} x \right) \tag{12}$$

$$x_1 + x_2 + x_3 \leq 2 \tag{13}$$

$$x_1 + x_2 - x_3 \geq 0 \tag{14}$$

$$x_1, x_2, x_3 \in \{0, 1\} \tag{15}$$

A função (11) retorna a maior redução possível do valor de função objetivo da partição atual. A restrição (12) indica que, mesmo após o movimento, $\sum_{x \in A_{max}} x \geq \sum_{x \in A_{min}} x$. A desigualdade (13) informa que no máximo dois elementos são realocados em um só movimento da metaheurística ILS: um ou dois elementos saem de A_{max} para A_{mim} ou um elemento sai de A_{max} para A_{min} e outro sai de A_{min} para A_{max}. A inequação (14) garante que a função objetivo atual nunca cresce com um movimento encontrado por esse problema. A relação de pertinência (15) indica se o elemento está ou não no movimento. A variável $x_1 = 1$ se a_i faz parte do melhor movimento e $x_1 = 0$ caso contrário, $x_2 = 1$ se a_j faz parte do melhor movimento e $x_2 = 0$ caso contrário, $x_3 = 1$ se b_l faz parte do melhor movimento e $x_3 = 0$ caso contrário. Particularmente, se $(x_1, x_2, x_3) = (0, 0, 0)$, não existe um movimento da metaheurística ILS que encolha a função objetivo: ($x_1 = 1$ e $x_3 = 1$) ou ($x_2 = 1$ e $x_3 = 1$) indicam uma troca entre A_{max} e A_{min}. Por fim, ($x_1 = 1$ e $x_2 = 1$) é a realocação de dois elementos de A_{max} para A_{min}.

A solução desses problemas apontam os elementos que devem ser trocados ou realocados. Essa operação é a busca local da metaheurística ILS.

5.3. Critérios de Parada

Os movimentos continuam enquanto houver possibilidade de melhora da função objetivo, segundo o Subproblema de Busca. Caso contrário, o algoritmo para. Esse critério evita ciclos, impedindo trocas já realizadas.

5.4. Implementação

Suponha, sem perda de generalidade, que $\{A_1, A_2, ..., A_k\}$ seja uma partição com as partes ordenadas em ordem decrescente por suas somas. A metaheurística ILS deve ser usada para fazer movimentos entre A_1 e A_k, já que opera trocas e realocações somente entre a parte de maior e a parte de menor soma. O Algoritmo 6 atualiza suas entradas, realocando um ou dois elementos de A_1 para A_k ou trocando dois elementos entre estas partes. Após reduzir o custo da função objetivo, ele retorna ao Subproblema de Busca para encontrar o novo movimento.

A partição é representada por uma `max heap` de tabelas *hash*. Isso justifica a primeira frase dessa seção. Com esta estrutura de dados, o custo para encontrar a maior e a menor parte após o movimento da metaheurística 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 A_1 e A_k são armazenados numa 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 o conjunto X com todos os seu elementos contidos no intervalo (a, b). Para alocar os elementos de X em uma tabela com tamanho m usa-se a função:

$$hash(x) = \left\lfloor m.\frac{(x-a)}{(b-a)} \right\rfloor \tag{16}$$

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 tenha uma complexidade amortizada de $O(n)$ para resolver o Subproblema de Busca.

Algoritmo 6: Adaptação da metaheurística ILS para a solução do k-PPN.

Entrada: $\{X_1, X_2, ..., X_k\}, obj$
Saída: $\{X_1, X_2, ..., X_k\}, obj$
início
  $obj1 = obj$;
  para $a \in X_1$ **faça**
    $troco = a - obj1/2$;
    se $troco > 0$ **então**
      $s1 =$ *busca-remove*(a, X_1);
      $s2 =$ *busca-remove*$(troco, X_k)$;
      $insere(s1, X_k)$;
      $insere(s2, X_1)$;
      $obj = obj - (s1 - s2)$;
    fim
    senão se $troco < 0$ **então**
      $s1 =$ *busca-remove*(a, X_1);
      $s2 =$ *busca-remove*$(-troco, X_1)$;
      $insere(s1, X_k)$;
      $insere(s2, X_k)$;
      $obj = obj - (s1 + s2)$;
    fim
    senão
      $s1 =$ *busca-remove*(a, X_1);
      $insere(s1, X_k)$;
      $obj = obj - s1$;
    fim
  fim
  retorna $\{X_1, X_2, ..., X_k\}, obj$
fim

6. Experimentos Computacionais

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 16.04 de 32 bits. As instâncias para os experimentos foram gerados 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.

O objetivo dos experimentos computacionais deste artigo é fazer uma comparação entre o resultado encontrado pelas heurísticas construtivas da Seção 4 e a adaptação da meta-heurística

ILS mostrada na Seção 5. A medida para a comparação dos resultados é a função:

$$Gap(A/B) = \frac{z(A) - z(B)}{z(A)}.100\% \quad (17)$$

que mostra o quanto a resposta do algoritmo B é menor que a do algoritmo A, quando $Gap(A/B) > 0$, ou maior, caso contrário, 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. Quanto mais próximo de 1 for o resultado da função $Gap(A/B)$, melhor é o 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 próxima de 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 e hipóteses necessárias para realização do teste t. Os valores de $Gap() = 100\%$ são devidos ao arredondamento de $99,999...\%$, quando existem.

Tabela 1: Comparação entre metaheurística ILS, Heurística de Karmarkar-Karp e Heurística Lpt_1

	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%

A Tabela 1 apresenta duas comparações: comparação entre a Heurística de Karmarkar-Karp (HKK) e a meta-heurística ILS apresentada no Algoritmo 6; e uma comparação entre a heurística Lpt_1 e a mesma adaptação da meta-heurística ILS. Estas comparações servem, por um lado, para justificar que as instâncias geradas aleatoriamente não são fáceis, já que, conforme pode ser avaliado, a heurística HKK falha em se aproximar da solução ótima. 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 HKK teria que reduzir sua solução para se equiparar ao valor encontrado pela meta-heurística ILS. O proposto ILS supera HKK 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 2. 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 $Gap(HKK/ILS) \leq 97\%$ 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 $33,33...$ menores que as respostas de HKK.

Outro teste t com 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 $P(Gap(HKK/ILS) \leq 50\%) = 0,000011475$. Em geral, isso significa que em 95% dos testes realizados teremos as respostas do ILS duas vezes menores que as respostas do Lpt_1.

Tabela 2: Comparação entre quartis do $Gap()$ das instâncias para cada valor de k

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%

O quadro 2 demonstra os quartis inferiores dos resultados de $Gap(HKK/ILS)$ 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.

7. Conclusão

Este artigo apresenta uma proposta de intensificação da heurística LPT voltada para resolução do k-PPN. O método empregado nos algoritmos parte de uma enumeração parcial dos elementos da instância e finaliza o resultado com o procedimento LPT. Os resultados obtidos foram testados com instâncias geradas aleatoriamente, tendo elementos uniformemente distribuídos.

A heurística Lpt_1 consegue superar a solução inicial da heurística LPT sempre que esta não é ótima, ao custo de um aumento de complexidade. O algoritmo Lpt_1 apresentado tem complexidade entre $O(n^2)$. Testes estatísticos confirmam que o ILS é significativamente melhor que HKK e Lpt_1.

Como trabalhos futuros, tem-se a aplicação da intensificação Lpt_1 e da metaheurística ILS propostos dentro de um método de enumeração implícita, bem com a realização de um teste de significância estatística adequados (não descritivos) para verificar hipóteses de eficiência sobre cada um dos algoritmos propostos.

Referências

Berretta, R. e Moscato, P. (1999). The number partitioning problem: An open challenge for evolutionary computation? In Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., e Price, K. V., editors, *New Ideas in Optimization*, p. 261–278. McGraw-Hill Ltd., UK.

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

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

Gent, I. P. e Walsh, T. (1998). Analysis of heuristics for number partitioning. *Computational Intelligence*, 14(3):430–451.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, XLV(9):1563–1581.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429.

Griffiths, M. e Mezo, I. (2010). A generalization of stirling numbers of the second kind via a special multiset. *Journal of Integer Sequences*, 13(2):3.

Horowitz, E. e Sahni, S. (1974). Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292.

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

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E., Thatcher, J. W., e Bohlinger, J. D., editors, *Proceedings of a Symposium on the Complexity of Computer Computations*.

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

Korf, R. E. (1998). A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106(2):181–203.

Korf, R. E. (2009). Multi-way number partitioning. In *IJCAI*, p. 538–543. Citeseer.

Korf, R. E., Schreiber, E. L., e Moffitt, M. D. (2013). Optimal sequential multi-way number partitioning. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM-2014)*.

Korf, R. E. (2010). Objective functions for multi-way number partitioning. In *Third Annual Symposium on Combinatorial Search*.

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

Michiels, W., Korst, J., Aarts, E., et al. (2003). Performance ratios for the karmarkar-karp differencing method. *Electronic Notes in Discrete Mathematics*, 13:71–75.

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

Pedroso, J. P. e Kubo, M. (2010). Heuristics and exact methods for number partitioning. *European Journal of Operational Research*, 202(1):73–81.

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

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

Schreiber, E. L. e Korf, R. E. (2014). Cached iterative weakening for optimal multi-way number partitioning. In *Proceedings of the Twenty-Eighth Annual Conference on Artificial Intelligence (AAAI-14) Quebec City, Canada*.

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

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