Trends in Computational and Applied Mathematics, **22**, N. 3 (2021), 1-21
Sociedade Brasileira de Matemática Aplicada e Computacional
Online version ISSN 2676-0029
www.scielo.br/tcam
doi: 00001

Algoritmo GVNS Híbrido Aplicado ao Problema das *p*-Medianas Capacitado

A. M. VASCONCELOS[1], M. J. F. SOUZA[2] e S. R. DE SOUZA[3*]

Recebido em 22 de fevereiro de 2020 / Aceito em 1 de março de 2021

RESUMO. O Problema das *p*-Medianas Capacitado (PPMC) consiste em localizar *p* facilidades em uma rede composta por *n* vértices e decidir qual facilidade atenderá cada vértice, a fim de minimizar a soma de todas as distâncias de cada facilidade para cada vértice e atender às restrições de capacidade máxima de cada facilidade. Neste trabalho, dado que o PPMC é da classe NP-difícil, quatro variantes da metaheurística *General Variable Neighborhood Search* (GVNS) são implementadas para resolver o PPMC: G-VND, G-RVND, GG-VND e GG-RVND. Elas diferem entre si com relação ao método usado para construir uma solução inicial e o usado para a busca local. Nas duas primeiras variantes, a solução inicial é gerada de forma aleatória e o método de busca local é feita via *Variable Neighborhood Descent* (VND) ou *Random Variable Neighborhood Descent* (RVND), respectivamente. Por sua vez, nas duas últimas, a solução inicial é feita via a fase de construção do método *Greedy Randomized Adaptive Search Procedure* (GRASP). Usando instâncias padrões de teste da literatura, mostramos, inicialmente, que a variante GG-VND teve melhor desempenho quando comparada com as demais. Em seguida, mostramos que, quando comparada com os algoritmos da literatura, esta variante possui desempenho equivalente ou superior.

Palavras-chave: Problema das *p*-Medianas Capacitado, *General Variable Neighborhood Search*, GRASP, metaheurísticas.

1 INTRODUÇÃO

O problema das *p*-Medianas (PPM), também conhecido na literatura como problema de localização de facilidades [21] ou problema de agrupamento (clusterização) [16], possui seus primeiros registros na década de 1960 [6]. Uma variação de grande importância do PMP é o problema das *p*-Medianas Capacitado (PPMC). O PPMC consiste em localizar *p* facilidades em

*Autor correspondente: Sérgio Ricardo de Souza – E-mail: sergio@dppg.cefetmg.br
[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Av. Amazonas, 7675, Nova Gameleira, 30510-000, Belo Horizonte, MG, Brasil – E-mail: andersonmv@cefetmg.br https://orcid.org/0000-0002-0870-4243
[2]Universidade Federal de Ouro Preto (UFOP), Campus Universitário, Morro do Cruzeiro, 35400-000, Ouro Preto, MG, Brasil – E-mail: marcone@ufop.edu.br https://orcid.org/0000-0002-7141-357X
[3]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Av. Amazonas, 7675, Nova Gameleira, 30510-000, Belo Horizonte, MG, Brasil – E-mail: sergio@dppg.cefetmg.br https://orcid.org/0000-0001-7831-6740

uma rede composta por n vértices e decidir qual facilidade atende cada vértice, de forma a minimizar a soma de todas as distâncias de cada vértice a cada facilidade, respeitando-se as restrições de capacidade máxima de atendimento.

Ao longo do tempo, diferentes métodos têm sido aplicados para a solução do PPMC, sejam métodos de programação matemática ou métodos heurísticos e metaheurísticos. Conforme [10], o PPMC é um problema da classe NP-difícil e a utilização de metaheurísticas para sua solução, portanto, se justifica. Assim, em [3], três algoritmos heurísticos são propostos, todos usando a fase de construção da metaheurística *Greedy Randomized Adaptative Search Procedure* (GRASP) para gerar a solução inicial: (i) *Scatter Search*; (ii) *Path Relinking*; (iii) metaheurística híbrida envolvendo *Path Relinking* e, após, *Scatter Search*. Segundo os autores, a metaheurística híbrida na forma descrita alcançou o melhor desempenho tanto em termos da qualidade da solução quanto no custo computacional. Já em [11], os autores utilizam o *Grouping Genetic Algorithm* (GGA) e o *Grouping Harmony Search* (GHS) para a solução deste problema. Os dois algoritmos compartilham o mesmo procedimento de codificação; no entanto, são baseados em conceitos muito diferentes no que diz respeito aos seus procedimentos operacionais. Segundo os autores, GGA é uma classe de algoritmos evolutivos, especialmente projetados para lidar com problemas de agrupamento, isto é, problemas nos quais um número de itens deve ser atribuído a um conjunto de grupos pré-definidos, enquanto GHS é um algoritmo de otimização iterativo, baseado na imitação da coordenação dos músicos em uma orquestra quando buscam conjuntamente a melhor harmonia sob uma métrica estética. Por sua vez, em [8] estuda-se a aplicação de algoritmos genéticos, propondo três variações para a solução do PPMC.

Em [5], um algoritmo baseado na metaheurística *Variable Neighborhood Search* (VNS) [7] foi aplicado para a solução do PPMC. Esse algoritmo é caracterizado por usar limites inferiores para verificar se é necessário ou não avaliar todas as soluções dentro da vizinhança. As soluções de interesse são, então, avaliadas por um algoritmo exato de um subproblema gerado. Em [23], um algoritmo matheurístico, combinando uma estrutura de vizinhança de plano de corte e uma metaheurística Busca Tabu é introduzido para a solução do PPMC. O algoritmo proposto é testado em vários conjuntos de instâncias e a análise estatística mostra a eficiência e eficácia do algoritmo híbrido. Já em [22], os mesmos autores anteriores apresentam uma matheurística envolvendo *Local Branching* e *Relaxation Induced Neighborhood Search* para a solução do PPMC. Por sua vez, [9] propõem uma matheurística integrando algoritmo genético, segundo uma variação da implementação apresentada em [2] também para a solução de PPMC, e um *solver* de programação matemática. Uma solução envolvendo mais diretamente métodos exatos de programação matemática é apresentada em [18]. É introduzido o procedimento *Iterated Reduction Matheuristic Algorithm* (IRMA), que é uma heurística híbrida baseada em decomposição, que utiliza um método de otimização local integrado a um *solver* de programação matemática. Este procedimento se constitui no estado da arte para a solução de instâncias do PPMC, conforme aponta [19].

O presente artigo propõe o estudo da solução do PPMC, buscando soluções de boa qualidade em tempo computacional reduzido, em especial no tocante a instâncias de grande dimensão. Para

isso, aplica-se a metaheurística *General Variable Neighborhood Search* (GVNS), que é uma variação da metaheurística *Variable Neighborhood Search* (VNS) [7], na qual a busca local é feita pelo método *Variable Neighborhood Descent* (VND). No GVNS clássico, a solução inicial é também definida de forma aleatória. Este artigo propõe uma alteração nesses procedimentos, de modo a gerar a solução inicial usando a fase de construção do método *Greedy Randomized Adaptive Search Procedure* (GRASP) [4], enquanto, para a busca local, usa-se ou o próprio VND ou sua variante *Random Variable Neighborhood Descent* (RVND) [17, 20].

O restante deste trabalho está organizado da seguinte forma. Na Seção 2 são apresentadas a definição e uma formulação do PPMC. A Seção 3 apresenta os algoritmos heurísticos utilizados para resolver o PPMC. Na Seção 4 são mostrados os testes realizados com os algoritmos desenvolvidos, bem como os resultados dos experimentos computacionais realizados. Na Seção 5 conclui-se o artigo.

2 DESCRIÇÃO DO PPMC

Esta seção descreve o Problema de p-Medianas Capacitado (PPMC). De acordo com [23], o PPMC é definido a partir de um grafo $\mathscr{G}$ não direcionado $\mathscr{G} = (\mathscr{V}, \mathscr{A})$, em que $\mathscr{V}$ é o conjunto dos vértices e $\mathscr{A}$ é o conjunto das arestas do grafo. Seja então $\mathscr{N}$ o conjunto de instalações e $\mathscr{M}$ o conjunto de clientes, de modo que $\mathscr{N} \subseteq \mathscr{V}$ e $\mathscr{M} \subseteq \mathscr{V}$, sendo $n = |\mathscr{N}|$ o número máximo de possíveis instalações e $m = |\mathscr{M}|$ o número de clientes a serem atendidos. É interessante salientar que as formulações em que $\mathscr{N} \subseteq \mathscr{M}$ ou, no limite, $\mathscr{N} = \mathscr{M}$, são situações particulares da formulação adotada. A cada cliente $j \in \mathscr{M}$ está associada uma demanda q_j e a cada instalação $i \in \mathscr{N}$ está associada uma capacidade máxima de atendimento Q_i. O custo do transporte de uma unidade de um produto de uma determinada instalação $i \in \mathscr{N}$ para um cliente $j \in \mathscr{M}$ é proporcional à distância d_{ij} entre as localizações i e j, sendo $d_{ij} > 0$ e $d_{jj} = 0$. O valor $p \leq n$ define o número de instalações que serão denominadas como medianas.

O PPMC consiste, assim, em determinar um conjunto $\mathscr{N}_p \subseteq \mathscr{N}$, que representa o conjunto de p-Medianas escolhidas dentre o conjunto de instalações $\mathscr{N}$, de modo a minimizar a distância total entre os clientes e suas respectivas instalações, asseguradas as capacidades das instalações. Seja y_i a variável de decisão binária que representa a decisão de seleção da instalação i como mediana, de modo que, se $y_i = 1$, então a instalação i foi definida como mediana; e $y_i = 0$, caso contrário. Seja x_{ij} a variável de decisão binária que assume valor $x_{ij} = 1$ se o cliente j é atendido pela instalação i; e valor $x_{ij} = 0$, caso contrário. A formulação matemática do PPMC é dada por:

$$\min \quad \sum_{i\in\mathscr{N}}\sum_{j\in\mathscr{M}} d_{ij}x_{ij} \tag{2.1}$$

$$\text{suj. a} \quad \sum_{i\in\mathscr{N}} x_{ij} = 1, \quad \forall\, j \in \mathscr{M} \tag{2.2}$$

$$\sum_{i\in\mathscr{N}} y_i = p \tag{2.3}$$

$$\sum_{j\in\mathscr{M}} q_j x_{ij} \leq Q_i y_i, \quad \forall\, i \in \mathscr{N} \tag{2.4}$$

$$x_{ij} \in \{0,1\}, \quad \forall\, i \in \mathscr{N},\ \forall\, j \in \mathscr{M} \tag{2.5}$$

$$y_i \in \{0,1\}, \quad \forall\ i \in \mathscr{N} \tag{2.6}$$

Uma solução ótima corresponde a um *cluster* viável, dado pelas instalações em $\mathscr{N}_p$ com menor distância total entre todas as instalações e seus respectivos clientes. A função objetivo (2.1), a ser minimizada, representa a soma dos custos de todos os clientes j a todas as instalações i. As restrições (2.2) garantem que cada cliente é atendido por apenas uma instalação, enquanto que a restrição (2.3) garante que a instalação i só pode ser escolhida se ela estiver ativa. As restrições (2.4) impõem que a capacidade total das instalações deve ser respeitada. As restrições (2.5) e (2.6) definem o domínio das variáveis de decisão.

3 ALGORITMOS PROPOSTOS

Para tratar o PPMC, este artigo propõe algoritmos heurísticos baseados na metaheurística GVNS. Esses algoritmos diferem entre si com relação ao método usado para construir uma solução inicial e o usado para a busca local. Esta seção está organizada como segue. Na subseção 3.1 mostramos como uma solução do PPMC é representada e avaliada, respectivamente. Os dois métodos de construção de uma solução são apresentados na Subseção 3.2. As estruturas de vizinhanças utilizadas para explorar o espaço de soluções do problema são apresentadas na Subseção 3.3. Os métodos de busca local e perturbação são apresentados nas subseções 3.4 e 3.5, respectivamente. Por fim, na Subseção 3.6, apresentamos o algoritmo base do GVNS para o PPMC.

3.1 Representação e Avaliação da Solução

Uma solução do PPMC é representada por dois vetores: o primeiro vetor representa o conjunto de instalações selecionadas como medianas, cujos elementos, portanto, estão contidos em $\mathscr{N}_p$; o segundo vetor representa as alocações das instalações i aos respectivos clientes j. Esta estrutura de representação foi proposta em [1]. Um exemplo de representação é apresentado na Figura 1. A Figura 1a mostra o vetor de instalações selecionadas como medianas, de modo que $\mathscr{N}_p = \{4,7,9\}$. A Figura 1b mostra a alocação dos $m = 10$ clientes às $p = 3$ instalações-medianas. Note que, à instalação $i = 4$, estão associados os clientes 3, 4 e 5; à instalação $i = 7$, são associados os clientes 2, 7 e 10; e à instalação $i = 9$, estão associados os clientes 1, 6, 8 e 9.

4	7	9

(a) Vetor de medianas.

Clientes	1	2	3	4	5	6	7	8	9	10
Medianas	9	7	4	4	4	9	7	9	9	7

(b) Vetor de alocação de clientes a instalações.

Figura 1: Exemplo de representação de uma solução.

A função de avaliação é a própria função objetivo do problema, dada pela Expressão (2.1), que representa a soma das distâncias entre as instalações i e seus clientes j.

3.2 Construção de uma solução inicial

Uma solução inicial é construída de duas formas distintas. Na primeira, detalhada na Subseção 3.2.1, mostramos como gerar uma solução inicial de forma randômica para o PPMC. Na segunda, detalhada na Subseção 3.2.2, mostramos como gerá-la pela fase de construção da metaheurística GRASP.

3.2.1 Construção randômica da solução inicial

O Algoritmo 1 mostra o pseudocódigo do procedimento de construção de uma solução inicial $s^0 = (vm^0, vc^0)$ de forma randômica. Os dados de entrada para este algoritmo são os conjuntos de clientes $\mathscr{M}$, de instalações-candidatas $\mathscr{N}$, o vetor Q contendo a capacidade de cada instalação-candidata, o vetor q de demanda de cada cliente, além do número p de medianas desejadas, do número m de clientes, do número n de instalações-candidatas e do número máximo *MaxTentativas* de tentativas para encontrar uma solução viável. Esse último parâmetro foi fixado em p tentativas. O procedimento inicia com a solução inicial vazia na linha 3; número de tentativas para encontrar uma solução viável igual a zero na linha 4; uma cópia $\mathscr{M}'$ do conjunto de clientes $\mathscr{M}$ na linha 5; uma cópia $\mathscr{N}'$ do conjunto de instalações-candidatas na linha 6; o conjunto vazio de medianas, isto é, das instalações $\mathscr{N}_p$ a serem abertas na linha 7; e o contador do número de medianas com valor zero, na linha 8.

Em seguida, o algoritmo entra em um laço formado pelas linhas 9-15 para escolher as medianas dentre o conjunto de instalações-candidatas. Inicialmente, o contador do número de medianas é acrescido em uma unidade na linha 10. A seguir, na linha 11, uma instalação-candidata $r \in \mathscr{N}'$ é aleatoriamente escolhida como mediana. Nas linhas 12-14, essa mediana r é inserida na δ-ésima posição do vetor de medianas vm^0, incluída no conjunto $\mathscr{N}_p$ de medianas e excluída do conjunto $\mathscr{N}'$ de instalações, respectivamente. Na linha 10, o contador do número de medianas é acrescido em uma unidade. Escolhidas as medianas dentre as instalações-candidatas, inicia-se um laço, entre as linhas 16-30, para atribuir cada cliente a uma mediana. Para tanto, na linha 17, um cliente é aleatoriamente escolhido. Em seguida, verificamos se o conjunto $\mathscr{N}_p^v \subseteq \mathscr{N}_p$ de medianas com

Algoritmo 1: Solução Inicial construída de forma randômica.

```
1  Entrada: M, N, Q, m, n, p, q, MaxTentativas
2  Saída: Solução inicial s^0 = (vm^0, vc^0), Conjunto N_p de medianas
3  s^0 ← ∅
4  Tentativas ← 0
5  M' ← M                                   /* Cópia do conjunto de clientes */ ;
6  N' ← N                                   /* Cópia do conj. de instalações-candidatas */
7  N_p ← ∅                                  /* Conjunto de medianas */
8  δ ← 0                                    /* Contador do número de medianas criadas */
9  enquanto (δ ≤ p) faça
10 |  δ ← δ + 1
11 |  Selecione aleatoriamente uma instalação r ∈ N' para ser mediana
12 |  vm^0_δ ← r                            /* Adicione a mediana r à δ-ésima posição do vetor de medianas vm^0 */
13 |  N_p ← N_p ∪ {r}                       /* Atualize o conjunto N_p */
14 |  N' ← N' \ {r}                         /* Atualize o conjunto N' */
15 fim
16 enquanto (M' ≠ ∅) faça
17 |  Selecione aleatoriamente um cliente j ∈ M'
18 |  se o conjunto N_p^v ⊆ N_p de medianas com capacidade para atender a demanda q_j do cliente j é não-vazio então
19 |  |  Selecione aleatoriamente uma mediana r ∈ N_p^v
20 |  |  vc^0_j ← r                         /* Insira a mediana r na j-ésima posição do vetor de clientes vc^0 */
21 |  |  M' ← M' \ {j}                      /* Atualize o conjunto de clientes */
22 |  senão
23 |  |  se Tentativas ≤ MaxTentativas então
24 |  |  |  Reinicialize a construção, retornando à linha 3
25 |  |  |  Tentativas ← Tentativas + 1
26 |  |  senão
27 |  |  |  Retorne s_0 = ∅, ou seja, não foi encontrada uma solução viável
28 |  |  fim
29 |  fim
30 fim
31 Retorne s^0 = (vm^0, vc^0), N_p
```

capacidade para atender a demanda q_j do cliente j é não-vazio. Havendo medianas disponíveis para alocação, uma delas, digamos $r \in \mathscr{N}_p^v$, é aleatoriamente escolhida para atender ao cliente j. Em seguida, nas linhas 20 e 21, respectivamente, são feitas a inserção dessa mediana na j-ésima posição do vetor de clientes vc^0 e atualização do conjunto de clientes. Não havendo uma mediana disponível para atender ao cliente j, o procedimento de construção é reinicializado na linha 24 e o número de tentativas de encontrar uma solução viável é incrementado na linha 25. Caso o número máximo de tentativas de construção tenha sido atingido, o procedimento construtivo retorna uma solução inicial vazia na linha 27. Sendo capaz de encontrar uma solução inicial viável, o algoritmo retorna a solução s^0 e o conjunto de medianas $\mathscr{N}_p$ na linha 31.

3.2.2 Solução inicial por meio da fase de construção GRASP

O próximo algoritmo desenvolvido utiliza o procedimento de construção da metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) [4] para gerar uma solução inicial. Nesse

Algoritmo 2: Solução Inicial construída via fase de construção GRASP.

1 **Entrada:** $\mathscr{M}$, $\mathscr{N}$, Q, m, n, p, q, g, α, *MaxTentativas*
2 **Saída:** Solução inicial $s^0 = (vm^0, vc^0)$, Conjunto $\mathscr{N}_p$ de medianas
3 $s^0 \leftarrow \emptyset$
4 *Tentativas* $\leftarrow 0$
5 $\mathscr{M}' \leftarrow \mathscr{M}$ /* Cópia do conjunto de clientes */
6 $\mathscr{N}' \leftarrow \mathscr{N}$ /* Cópia do conj. de instalações-candidatas */
7 $\mathscr{N}_p \leftarrow \emptyset$ /* Conjunto de medianas */
8 $\delta \leftarrow 0$ /* Contador do número de medianas criadas */
9 **enquanto** *(*$\delta \leq p$*)* **faça**
10 $\delta \leftarrow \delta + 1$
11 $LRC \leftarrow \{i, i \in \mathscr{N}' \mid g(i) \leq g_{\min} + \alpha \times [g_{\max} - g_{\min}]\}$
12 Selecione aleatoriamente uma instalação $r \in LRC$ para ser mediana
13 $vm^0_\delta \leftarrow r$ /* Adicione a mediana r à δ-ésima posição do vetor de medianas vm^0 */
14 $\mathscr{N}_p \leftarrow \mathscr{N}_p \cup \{r\}$ /* Atualize o conjunto $\mathscr{N}_p$ */
15 $\mathscr{N}' \leftarrow \mathscr{N}' \setminus \{r\}$ /* Atualize o conjunto $\mathscr{N}'$ */
16 **fim**
17 **enquanto** $(\mathscr{M}' \neq \emptyset)$ **faça**
18 Selecione aleatoriamente um cliente $j \in \mathscr{M}'$
19 **se** *o conjunto* $\mathscr{N}_p^v \subseteq \mathscr{N}_p$ *de medianas com capacidade para atender a demanda* q_j *do cliente* j *é não-vazio* **então**
20 Selecione aleatoriamente uma mediana $r \in \mathscr{N}_p^v$
21 $vc^0_j \leftarrow r$ /* Insira a mediana r na j-ésima posição do vetor de clientes vc^0 */
22 $\mathscr{M}' \leftarrow \mathscr{M}' \setminus \{j\}$ /* Atualize o conjunto de clientes */
23 **senão**
24 **se** *Tentativas* $\leq$ *MaxTentativas* **então**
25 Reinicialize a construção, retornando à linha 3
26 *Tentativas* $\leftarrow$ *Tentativas* $+ 1$
27 **senão**
28 **Retorne** $s_0 = \emptyset$, ou seja, não foi encontrada uma solução viável
29 **fim**
30 **fim**
31 **fim**
32 **Retorne** $s^0 = (vm^0, vc^0), \mathscr{N}_p$

procedimento, dado um conjunto $\mathscr{N}$ de instalações-candidatas e um conjunto $\mathscr{M}$ de clientes, cada elemento candidato $i \in \mathscr{N}$ é, inicialmente, avaliado por uma função guia $g(\cdot)$, dada por:

$$g(i) = \sum_{j \in \mathscr{M}} d_{ij} \ \ \forall i \in \mathscr{N} \tag{3.1}$$

que avalia a distância de uma instalação-candidata $i \in \mathscr{N}$ a todos os clientes.

A lista restrita de candidatos (LRC) é constituída das "melhores" instalações-candidatas avaliadas segundo a função (3.1). A Expressão (3.2) mostra como essa lista é constituída.

$$LRC = \{i \in \mathscr{N} \mid g(i) \leq g_{\min} + \alpha \times [g_{\max} - g_{\min}]\}. \tag{3.2}$$

Na Expressão (3.2), $g_{\min}$ representa o menor valor da função g nesse conjunto de candidatos e $g_{\max}$ o maior valor dessa função. Observe que a LRC é controlada pelo parâmetro real $\alpha \in [0,1]$, que define o quão gulosa ou aleatória é a construção da solução, de modo que, se $\alpha = 1$, a construção é totalmente aleatória, e se $\alpha = 0$, ela é totalmente gulosa. Formada a LRC com os melhores candidatos, um elemento $r \in LRC$ é escolhido aleatoriamente para compor a solução parcial. Ou seja, na fase de construção do GRASP, há uma regra de seleção que contém um fator aleatório para definir o candidato escolhido.

A solução inicial, via o procedimento da fase de construção GRASP, foi implementada conforme o Algoritmo 2. Os dados de entrada para este algoritmo são: conjuntos de clientes $\mathscr{M}$ e de instalações-candidatas $\mathscr{N}$; vetor Q com a capacidade de cada instalação-candidata; vetor q de demanda de cada cliente; número p de medianas desejadas; número m de clientes; número n de instalações-candidatas; função $g(\cdot)$; parâmetro α; e número máximo de tentativas *MaxTentativas* para encontrar uma solução inicial viável, fixado em p. Como no procedimento anterior, a inicialização é realizada nas linhas 3 a 8.

O algoritmo entra, então, em um laço formado pelas linhas 9-16 para definir as medianas dentre o conjunto de instalações-candidatas. Inicialmente, o contador do número de medianas é acrescido em uma unidade na linha 10. Em seguida, na linha 12, uma instalação $r \in LRC$ é aleatoriamente escolhida como mediana dentre as instalações-candidatas. Nas linhas 13-15, essa mediana r é inserida na δ-ésima posição do vetor de medianas vm^0, incluída no conjunto $\mathscr{N}_p$ de medianas e excluída do conjunto $\mathscr{N}'$ de instalações-candidatas, respectivamente. Uma vez escolhidas as medianas dentre as instalações-candidatas, inicia-se um laço, entre as linhas 17-31, para atribuir cada cliente a uma mediana. Essa atribuição é feita da mesma forma que no Algoritmo 1.

3.3 Estruturas de Vizinhança

Para explorar o espaço de soluções do problema, nós utilizamos três tipos de movimentos para serem aplicados a uma solução s. Cada movimento dá origem a uma estrutura de vizinhança $\mathscr{L}_k(s)$, $k = 1,2,3$, cada qual definida como segue:

(i) $\mathscr{L}_1(s)$ (Realocação): Consiste em realocar um cliente j de uma mediana $r_1 \in \mathscr{N}_p$ para uma outra mediana $r_2 \in \mathscr{N}_p$;

(ii) $\mathscr{L}_2(s)$ (Troca): Consiste em trocar um cliente j_1 de uma mediana $r_1 \in \mathscr{N}_p$ com um cliente j_2 de uma outra mediana $r_2 \in \mathscr{N}_p$;

(iii) $\mathscr{L}_3(s)$ (Substituição): Consiste em substituir uma mediana $r_1 \in \mathscr{N}_p$ por uma instalação-candidata $r_2 \in \mathscr{N} \setminus \mathscr{N}_p$.

3.4 Busca Local

Uma busca local é um procedimento de refinamento de uma solução s que utiliza uma ou mais estruturas de vizinhança para encontrar um ótimo local com relação às estruturas de vizinhança utilizadas na exploração do espaço de soluções.

Utilizamos duas versões diferentes para o procedimento de busca local, ambas baseadas na troca sistemática das estruturas de vizinhança. A primeira delas, apresentada no Algoritmo 3, aplica o método *Variable Neighborhood Descent* (VND) [7, 14], em sua forma clássica. A segunda versão aplica o método *Random Variable Neighborhood Descent* (RVND) [17, 20]. O RVND é uma variação do algoritmo VND e é apresentado no Algoritmo 4. A diferença entre as duas versões reside na ordem de exploração das vizinhanças. Enquanto no VND clássico a ordem das

Algoritmo 3: VND aplicado à solução do PPMC.

```
1  Entrada: Solução s, Conjunto 𝓛 de vizinhanças, Número k_max de vizinhanças
2  Saída: Solução s refinada
3  k ← 1                                              /* Inicie pela primeira vizinhança */
4  enquanto k ≤ k_max faça
5      Encontre o melhor vizinho s' ∈ 𝓛'_k(s) ⊂ 𝓛_k(s)
6      se f(s') < f(s) então
7          s ← s'                                          /* Atualize a solução corrente */
8          k ← 1                                  /* Retorne à primeira vizinhança de 𝓛 */
9      senão
10         k ← k+1                                /* Passe para a próxima vizinhança de 𝓛 */
11     fim
12 fim
13 Retorne s
```

Algoritmo 4: RVND aplicado à solução do PPMC.

```
1  Entrada: Solução s, Conjunto 𝓛 de vizinhanças, Número k_max de vizinhanças
2  Saída: Solução s refinada
3  𝓛ℛ ← 𝓛 em ordem aleatória
4  k ← 1                                          /* Inicie pela primeira vizinhança de 𝓛ℛ */
5  enquanto k ≤ k_max faça
6      Encontre o melhor vizinho s' ∈ 𝓛ℛ'_k(s) ⊂ 𝓛ℛ_k(s)
7      se f(s') < f(s)) então
8          s ← s'                                          /* Atualize a solução corrente */
9          k ← 1                                  /* Retorne à primeira vizinhança de 𝓛ℛ */
10     senão
11         k ← k+1                                /* Passe p/a próxima vizinhança de 𝓛ℛ */
12     fim
13 fim
14 Retorne s
```

vizinhanças é pré-definida e não muda durante as chamadas ao método, no RVND essa ordem é aleatória a cada chamada do método. Essa diferença entre essas duas buscas locais ocorre na linha 3 do Algoritmo 4, que inclui um procedimento para aleatorizar a ordem das vizinhanças. Note que, em ambos os métodos, a função $f(s)$ representa a função de avaliação definida na Subseção 3.1.

No método VND implementado, as estruturas de vizinhanças foram ordenadas como apresentado na Subseção 3.3. Esta ordenação segue o grau de complexidade da exploração das vizinhanças usadas, sendo a primeira vizinhança ($\mathscr{L}_3$) a de menor complexidade e a última, $\mathscr{L}_3$, a de maior complexidade. Esta estratégia é normalmente adotada em buscas locais baseadas em VND. Como a determinação do ótimo local de cada vizinhança é muito custosa computacionalmente em instâncias maiores do problema, em ambos os métodos reduzimos a análise da vizinhança a apenas uma parte dela. Mais precisamente, analisamos na vizinhança $\mathscr{L}_1$ apenas m vizinhos gerados por realocações aleatórias de clientes de uma mediana para outra na solução atual. Igualmente, na vizinhança $\mathscr{L}_2$ analisamos somente m vizinhos gerados por trocas aleatórias entre clientes de duas medianas na solução atual. Por fim, na vizinhança $\mathscr{L}_3$ analisamos m vizinhos gerados por movimentos aleatórios de substituição de medianas $r_1 \in \mathscr{N}_p$ da solução atual por instalações-

Algoritmo 5: Perturbação.

```
1  Entrada: Solução atual s, Vizinhança L_3, Número k de perturbações
2  Saída: Solução perturbada s'
3  s' ← s                                              /* Cópia da solução atual */
4  i ← 1                                               /* Inicie o contador de perturbações */
5  enquanto (i ≤ k) faça
6  |   Selecione aleatoriamente uma solução vizinha s'' ∈ L_3(s')
7  |   s' ← s''                                        /* Atualize a solução perturbada */
8  |   i ← i + 1                                       /* Atualize o contador de perturbações */
9  fim
10 Retorne s'
```

candidatas $r_2 \in \mathscr{N} \setminus \mathscr{N}_p$. Em todas essas análises, m indica o número de clientes. Para indicar que apenas uma parcela da vizinhança é analisada, representamos essa operação por $\mathscr{L}'_k$.

3.5 Perturbação

Para diversificar a busca para outras regiões do espaço de soluções do PPMC, utilizamos um procedimento de perturbação da solução atual. O Algoritmo 5 mostra o seu funcionamento. Na linha 1 são dadas como entradas, a solução atual s, a estrutura de vizinhança utilizada para a realização das perturbações, no caso, $\mathscr{L}_3$, e o número k de perturbações a serem realizadas nessa vizinhança. Na linha 3, é feita uma cópia da solução atual na solução s', a qual representará a solução perturbada após a aplicação do procedimento. Na linha 4, o contador i do número de perturbações a ser aplicado à solução s' é iniciado com o valor 1. Enquanto esse contador for menor ou igual ao número k de perturbações a serem aplicadas, o procedimento seleciona uma solução vizinha aleatória s'' na vizinhança da solução s' (veja linha 6). Em seguida, são atualizados a solução perturbada e o contador do número de perturbações nas linhas 7 e 8, respectivamente. O procedimento termina retornando a solução perturbada na linha 10.

3.6 Algoritmo GVNS

O Algoritmo 6 mostra o pseudocódigo do algoritmo GVNS implementado. Os parâmetros de entrada são o conjunto de estruturas de vizinhanças $\mathscr{L}$ e o número máximo permitido de iterações sem melhora *iterMaxSM*. A solução inicial s_0 (linha 1) é gerada ou aleatoriamente, como mostrado pelo Algoritmo 1, ou por meio da fase de construção do método GRASP, como mostrado pelo Algoritmo 2, ambos mostrados na Subseção 3.2.

Gerada a solução inicial, na linha 4, o algoritmo aplica sobre ela um método de busca local usando ou o procedimento VND, conforme Algoritmo 3, ou o procedimento RVND, conforme Algoritmo 4, para gerar a solução atual s. Em seguida, o algoritmo inicializa o contador do número de iterações sem melhora da solução atual e o contador k do número de perturbações a serem aplicadas sobre a solução atual para diversificar a busca para outras regiões do espaço de soluções do problema. Esse contador k é inicializado com o valor 2 porque a vizinhança utilizada pela perturbação é uma das vizinhanças usadas para aplicar a busca local. Assim, se $k = 1$,

Algoritmo 6: GVNS.

```
1  Entrada: 𝒩, iterMaxSM
2  Saída: s
3  s_0 ← GeraSolucaoInicial()                          /* Conforme Algoritmos 1 ou 2 */
4  s ← BuscaLocal(s_0, 𝒩, k_max)                       /* Conforme Algoritmos 3 ou 4 */
5  iter ← 0                          /* Contador do número de iterações sem melhora */
6  k ← 2                                    /* Contador do número de perturbações */
7  enquanto (iter < iterMaxSM) faça
8      s' ← Perturbação(s, 𝒩_3, k)                   /* De acordo com o Algoritmo 5 */
9      s'' ← Busca Local(s', 𝒩, k_max)                    /* Cfe. Algoritmos 3 ou 4 */
10     se f(s'') < f(s) então
11         s ← s''                                    /* Atualize a solução atual */
12         k ← 1                                  /* Retorne à primeira vizinhança */
13         iter ← 0                 /* Reinicialize o contador de iterações sem melhora */
14     senão
15         iter ← iter + 1              /* Atualize o contador de iterações sem melhora */
16         k ← k + 1                             /* Passe para a próxima vizinhança */
17     fim
18 fim
19 Retorne s
```

a solução perturbada seria um vizinho da solução atual e, portanto, uma busca local a partir dela alcançaria a própria solução atual. Em seguida, tendo como critério de parada o número máximo de iterações sem melhora (*iterMaxSM*), o algoritmo entra em um laço iterativo com aplicações sucessivas dos procedimentos de perturbação da solução atual e busca local da solução perturbada para explorar as soluções do espaço de soluções do problema. A perturbação (linha 8) é aplicada sobre a solução atual usando o Algoritmo 5. Esse Algoritmo realiza k movimentos aleatórios em s, produzindo vizinhos cada vez mais distantes da solução atual quando uma melhoria não é encontrada. Desta forma, esse procedimento diversifica o caminho de busca. Tal como na linha 4, o procedimento de busca local usa ou a heurística VND, conforme Algoritmo 3, ou a heurística RVND, conforme Algoritmo 4 (linha 9). A solução s'' gerada pelo procedimento de busca local será aceita se melhorar a solução atual (linhas 10-13). Caso isso ocorra, a solução atual é atualizada, a estrutura de vizinhança retorna à estrutura inicial ($k = 1$) e o procedimento continua. Caso contrário, a estrutura de vizinhança é incrementada (linhas 14-17). A melhor solução encontrada é retornada ao fim do processo de busca.

4 EXPERIMENTOS COMPUTACIONAIS

Esta seção apresenta os resultados dos experimentos computacionais realizados neste trabalho e faz uma análise estatística do desempenho das quatro variantes do algoritmo baseado na metaheurística *General Variable Neighborhood Search* (GVNS) para resolver o PPMC: (i) G-RVND: GVNS com a solução inicial gerada de forma randômica e busca local via *Random Variable Neighborhood Descent*; (ii) G-VND: GVNS com a solução inicial gerada de forma randômica e busca local via *Variable Neighborhood Descent*; (iii) GG-RVND: GVNS com a solução inicial gerada pela fase de construção do método GRASP e busca local via RVND; e (iv) GG-VND: GVNS com a solução inicial gerada pela fase de construção do método GRASP e

busca local via VND. Todos os algoritmos foram implementados na linguagem `Java` e testados em um computador Intel Core 2 duo 2.93 GHz, 32 bits, 3 MB de cache L2, 8 GB de memória RAM e sistema operacional Ubuntu Server.

Dois grupos de instâncias foram utilizados nos testes computacionais. O Grupo de instâncias 1, proposto em [1], possui 5 instâncias que variam de 3038 nós e 600 medianas a 3038 nós e 1000 medianas. Os resultados dos algoritmos neste grupo estão reportados na Subseção 4.1. O Grupo de instâncias 2, criado em [18], totaliza 30 instâncias, que variam de 318 nós e 5 medianas a 4461 nós e 1000 medianas. Os resultados para este grupo são mostrados na Subseção 4.2. Para comparar a qualidade das soluções produzidas pelos algoritmos, calcula-se o desvio relativo médio Δ_i^A:

$$\Delta_i^A = \frac{f_i^A - f_i^\star}{f_i^\star} \times 100, \tag{4.1}$$

entre o valor f_i^A encontrado pelo algoritmo A na instância i e a melhor solução encontrada na literatura nessa instância, representada por $f_i^\star$. Este valor, portanto, mostra o desvio percentual em relação ao melhor resultado da literatura e, assim, quanto mais próximo de 0, melhor é o resultado do algoritmo A.

Os valores dos parâmetros de todos os algoritmos implementados foram calibrados pela ferramenta IRACE (*Iterated Racing for Automatic Algorithm Configuration*) [12]. Para o treinamento da ferramenta, foram utilizadas as instâncias lin318_70, ali535_100, u724_125, rl1304_200, pr_300 e p3038_800. A Tabela 1 mostra os resultados desse treinamento. Nessa tabela, a primeira coluna apresenta o parâmetro testado, a segunda descreve seu significado e a terceira, a faixa de valores testados. Na última coluna, mostramos o valor retornado pelo IRACE.

Tabela 1: Faixa de valores testados e retornados pelo IRACE para os parâmetros nos Grupos de instâncias 1 e 2.

Parâmetro	Descrição	Faixa de valores	Valor retornado
α	Valor utilizado pela LRC	0,1 a 0,9	0,40
iterMaxSM c/busca local VND	Número máximo de iterações sem melhora	{500, 600, 700, 800 }	600
iterMaxSM c/busca local RVND	Número máximo de iterações sem melhora	{500, 600, 700, 800 }	700

4.1 Resultados no grupo de instâncias 1

Esta seção apresenta os resultados dos testes computacionais referentes ao Grupo de instâncias 1. Primeiramente, na Subseção 4.1.1, compara-se o desempenho das quatro variantes propostas, de modo a eleger aquela que possui o melhor comportamento. Os resultados desta variante são comparados com os resultados de outros algoritmos encontrados na literatura, tanto no que diz respeito aos valores obtidos, na Subseção 4.1.2, quanto, na Subseção 4.1, em relação ao tempo de execução despendido.

Tabela 2: Grupo de instâncias 1 - Valores médios para 30 execuções.

Instância	Melhor resultado da	G-RVND		G-VND		GG-RVND		GG-VND	
teste	literatura	Δ(%)	tempo(s)	Δ(%)	tempo(s)	Δ(%)	tempo(s)	Δ(%)	tempo(s)
p3038-600	122020,66	2,42	396,15	0,90	425,59	1,20	359,56	0,38	517,50
p3038-700	108685,59	2,60	467,28	0,53	544,48	1,54	458,07	0,63	660,18
p3038-800	98530,99	1,70	538,80	1,63	625,50	1,22	566,76	0,29	765,17
p3038-900	90239,65	2,11	803,41	1,92	789,47	1,48	709,60	0,28	942,04
p3038-1000	83231,58	2,39	1154,76	0,92	1069,50	0,78	1026,79	0,48	986,82

Tabela 3: Grupo de instâncias 1 - Teste de normalidade dos valores de desvio.

	G-RVND	G-VND	GG-RVND	GG-VND
p-valor	0,8167	0,9286	0,6197	0,4513

Tabela 4: Grupo de instâncias 1: Teste F para os valores de desvio percentual Δ.

Algoritmo	GG-VND	GG-RVND	G-VND	G-RVND
G-RVND	**0,0022**	0,0636	**0,01406**	–
G-VND	**6,2052E-06**	0,4371	–	–
GG-RVND	**3,3022E-05**	–	–	–

4.1.1 Comparação entre os algoritmos propostos

A Tabela 2 apresenta os resultados referentes à aplicação dos algoritmos G-RVND, G-VND, GG-RVND e GG-VND para o Grupo de instâncias 1, no que diz respeito aos desvios em relação aos resultados encontrados em [13]. A primeira coluna mostra as instâncias, na forma n-p, de modo que $n = 3038$ indica o número de clientes e p o número de medianas. A segunda coluna mostra os melhores resultados encontrados em [13]. Entre a terceira até a décima colunas são apresentados os valores dos desvios relativos médios Δ%, conforme a Expressão (4.1), e o tempo de execução associado ao uso de cada um dos quatro algoritmos implementados.

A Tabela 3 mostra o p-valor para cada um dos conjuntos de dados associados à implementação dos algoritmos no Grupo de instâncias 1. Como esses dados apresentam normalidade estatística, utilizou-se o Teste F [15] para verificar se há diferença estatística significativa entre os algoritmos. A Figura 2 mostra o gráfico *box-plot* dos algoritmos propostos no Grupo de instâncias 1 em relação ao desvio relativo médio Δ.

Analisando o gráfico da Figura 2 e os p-valores da Tabela 4, observa-se que há diferença estatística significativa entre a variante GG-VND e as demais. Portanto, comprova-se a superioridade desta variante em relação às demais, uma vez que, pela Tabela 2, os valores dos desvios relativos do GG-VND foram os menores dentre todos os algoritmos testados.

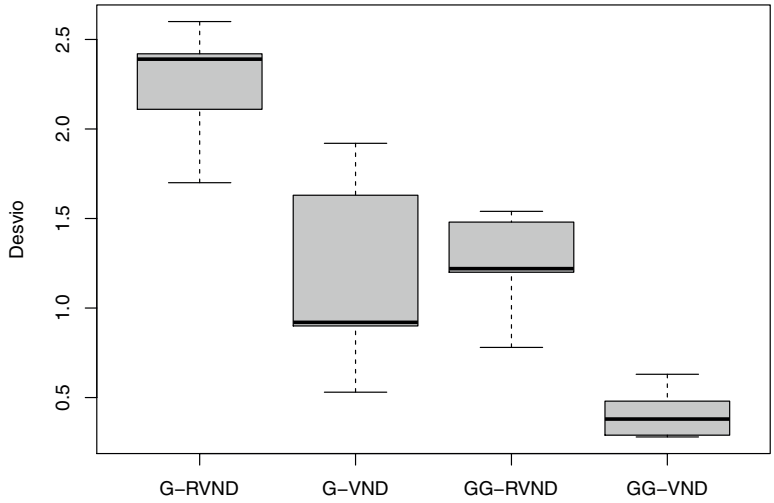

Figura 2: Gráfico *box-plot* mostrando o desvio relativo médio Δ% entre os algoritmos propostos no Grupo de instâncias 1.

4.1.2 Comparação com a literatura

Os testes realizados nas Subseção 4.1.1 mostraram estatisticamente que o algoritmo GG-VND é superior aos algoritmos G-RVND, G-GND e GG-RVND. Portanto, nas seções subsequentes, apenas o Algoritmo GG-VND será comparado em relação aos melhores resultados da literatura, apresentados em [18], [1] e [9]. A Tabela 5 mostra esta comparação. A primeira coluna mostra o conjunto de instâncias tratado. Entre a terceira e a sexta colunas são exibidos os melhores resultados encontrados em [18], [1], [9] e o referente ao algoritmo GG-VND. Em seguida, entre a sétima e a décima colunas são mostrados os resultados dos desvios relativos médios entre os resultados dos algoritmos da literatura e os do algoritmo GG-VND.

Tabela 5: Grupo de instâncias 1: comparação com a literatura.

Instâncias	Melhor Resultado da					Δ (%)			
teste	literatura	[18]	[1]	[9]	GG-VND	[18]	[1]	[9]	GG-VND
p3038-600	122020,66	122724,79	129194,11	125638,33	122752,19	0,57	5,879	2,96	**0,38**
p3038-700	108685,59	109695,61	117295,47	114977,77	109381,22	0,92	7,92	5,78	**0,63**
p3038-800	98530,99	100084,41	109532,61	105483,82	99054,98	1,57	11,16	7,05	**0,29**
p3038-900	90239,65	92317,78	102458,93	100372,64	90969,96	2,30	13,54	11,22	**0,28**
p3038-1000	83231,58	85856,05	97777,67	96290,2	83693,38	3,15	17,47	5,68	**0,48**

De acordo com o *p*-valor da Tabela 6, conclui-se que os dados apresentam padrão de normalidade. Tendo em vista o *p*-valor da Tabela 7 e o gráfico da Figura 3, conclui-se que há diferença significativa dos resultados do algoritmo GG-VND em relação aos dos algoritmos apresentados

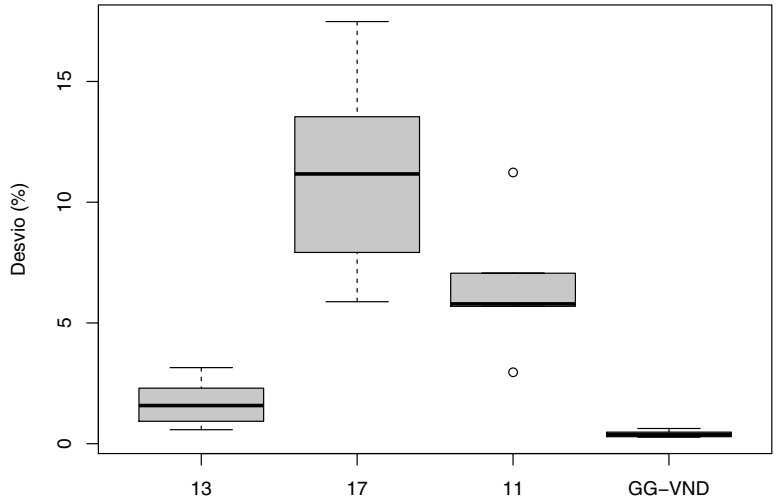

Figura 3: Gráfico *box-plot* mostrando o desvio relativo médio Δ entre o algoritmo GG-VND e os da literatura no Grupo de instâncias 1.

Tabela 6: Grupo de instâncias 1: Teste de normalidade dos valores de desvio $\times$ literatura. Intervalo de confiança: $\rho = 95\%$.

	[18]	[1]	[9]	GVNS
p-Value	0,8045	0,08549	0,548	0,4513

Tabela 7: Grupo de instâncias 1: Teste F dos valores de desvio relativo percentual Δ $\times$ valores da literatura. Intervalo de confiança: $\rho = 95\%$.

	[18]	[1]	[9]
p-valor GG-VND	**0,0022**	**6,2052E-06**	**3,3022E-05**

em [1], [9] e [18]. Por fim, analisando-se a Tabela 5, conclui-se que o algoritmo GG-VND possui melhor desempenho que estes algoritmos da literatura.

4.1.3 Comparação em relação aos tempos de execução

Nesta seção, é feita a comparação dos tempos de execução computacional do algoritmo GG-VND frente aos resultados dos algoritmos de [1], [18] e [9]. É importante destacar que não é uma tarefa simples comparar os tempos computacionais entre algoritmos executados em diferentes CPUs, linguagens de programação, *solvers* e Sistemas Operacionais. Para melhor compreensão, usou-se, como referência, a Tabela 8. Os fatores de comparação que compõem esta tabela foram obtidos em `https://www.cpubenchmark.net/compare/`, com acesso em 17 de novembro de 2019.

Tabela 8: Grupo de instâncias 1: comparação entre CPUs.

	Fator de desempenho CPU				Fator de correção (%)		
	GVNS	[18]	[1]	[9]	[18]	[1]	[9]
single-thread	1143	1575	793	2125	1,3779	0,6937	1,8591

Tabela 9: Grupo de instâncias 1: comparação com a literatura para tempo de execução.

Instâncias	Resultados da literatura (s)			Tempo de execução ajustado (s)			Tempo (s)
teste [18]	[1]	[9]	[18]	[1]	[9]	[18]	GG-VND
p3038-600	2685,38	39379,33	3600	3729,69	27171,73	6792,45	**517,50**
p3038-700	2239,84	47286,32	3600	3110,88	32627,56	6792,45	**660,18**
p3038-800	2819,26	54584,69	3600	3915,63	37663,43	6792,45	**765,17**
p3038-900	1578,17	63040,95	3600	2191,90	43498,25	6792,45	**942,04**
p3038-1000	1874,08	70413,38	3600	2602,88	48585,23	6792,45	**986,82**

A Tabela 9 mostra a comparação entre o tempo de execução do algoritmo GG-VND e o alcançado pelos algoritmos de [1], [18] e [9]. Nesta tabela, a sexta, a sétima e a oitava colunas apresentam os valores de tempos de execução corrigidos a partir dos fatores da Tabela 8, e, portanto, conclui-se que o algoritmo GG-VND requer menor tempo de processamento e, em vista dos resultados apresentados na Tabela 5, possui melhor desempenho que os demais algoritmos analisados.

4.2 Resultados no grupo de instâncias 2

Esta seção apresenta os resultados referentes à aplicação do Algoritmo GG-VND para a solução do Grupo de instâncias 2, proposto em [18], conforme já citado. Este grupo é dividido em subgrupos, em que o primeiro número indica o número de clientes (n) e o segundo o número de medianas (p) adotado. A Tabela 10 mostra os resultados médios obtidos pelo algoritmo GG-VND e a comparação em relação aos melhores valores conhecidos da literatura (MSC), encontrados em `http://www-usr.inf.ufsm.br/~stefanello/instances/cpmp/`, com os alcançados em [18] e [19]. Observa-se que os desvios em relação ao MSC obtidos pelo algoritmo GG-VND são menores que os das demais referências em 17 das 30 instâncias desse grupo.

A Tabela 11 mostra a comparação entre o algoritmo GG-VND e os algoritmos encontrados em [18] e [19], com relação ao tempo de execução, em segundos. Novamente, a superioridade do algoritmo GG-VND proposto é realçada, uma vez que, tendo em vista os tempos computacionais corrigidos, o algoritmo GG-VND possui tempo de execução menor em 28 das 30 instâncias analisadas.

A Tabela 12 mostra os resultados do teste de normalidade dos valores de desvio percentual para o Grupo de instâncias 2. Como a normalidade não foi comprovada para todas as instâncias, foi realizado o teste não-paramétrico Kruskall-Wallis. Os resultados deste teste, apresentados na Tabela 13, mostram que não há diferença estatística entre os algoritmos avaliados para todas as instâncias de teste. No entanto, deve-se enfatizar que os valores de desvio relativo médio

Tabela 10: Grupo de instâncias 2: Comparação com os métodos da literatura.

		Resultados da literatura e GG-VND			Desvio (%) em relação aos MSC		
Instância	MSC	[18]	[19]	GG-VND	[18]	[19]	GG-VND
lin318-005	180281,2	180281,2	–	180284,20	0,00	–	**0,03**
lin318-015	88901,56	88901,56	–	88905,20	0,00	–	**0,02**
lin318-040	47988,38	48003,88	–	47995,69	0,03	–	**0,07**
lin318-070	32198,64	32290,39	–	32200,53	0,28	–	**0,03**
lin318-100	22942,69	22942,69	–	23168,12	0,00	–	**1,02**
ali535-005	9956,77	10210,58	–	9956,60	2,54	–	0,07
ali535-025	3695,15	3701,88	–	3695,22	0,18	–	**0,06**
ali535-050	2460,93	2478,04	–	2510,21	0,69	–	2,21
ali535-100	1437,62	1448	–	1451,92	0,72	–	1,05
ali535-150	1032,28	1037,7	–	1032,45	0,52	–	**0,03**
u724-010	181580,86	182611,19	182382	181586,01	0,56	0,44	**0,08**
u724-030	95034,01	95159,96	95824	95043,90	0,13	0,83	**0,02**
u724-075	54735,05	54735,05	54867	54737,77	0,00	0,24	**0,04**
u724-125	38976,76	38976,76	39808	38976,60	0,00	2,13	0,06
u724-200	28079,97	28082,72	28881	28592,14	0,00	2,85	1,91
rl1304-010	2146484,1	2166552	2149366	2158039,60	0,93	0,13	0,61
rl1304-050	799694,32	806425,28	804254	799718,88	0,84	0,57	**0,10**
rl1304-100	495925,93	498411,69	502275	495945,74	0,50	1,28	**0,03**
rl1304-200	276977,60	276983,73	280879	277477,09	0,00	1,40	**0,17**
rl1304-300	191224,85	191258,69	195690	192995,41	0,01	2,33	0,87
pr2392-020	2235376,7	2250292,40	2235790	2235448,34	0,66	0,01	0,06
pr2392-075	1092294	1098560	1092917	1092362,69	0,57	0,05	**0,05**
pr2392-150	711115,25	711315,15	715719	711157,29	0,02	0,64	**0,03**
pr2392-300	458145,29	458221,63	462726	458170,55	0,01	0,99	0,07
pr2392-500	316042,97	316092,46	323348	316056,12	0,01	2,31	0,06
fnl4461-0020	1283536,7	1292621,6	1284037	1296295,22	0,70	0,03	1,07
fnl4461-0100	548909,01	550758,21	549126	548917,23	0,33	0,03	**0,05**
fnl4461-0250	335888,87	336006,96	337205	335911,49	0,03	0,39	**0,06**
fnl4461-0500	224662,49	224684,37	226283	228165,55	0,00	0,72	1,61
fnl4461-1000	145862,47	145870,78	148454	147703,03	0,00	1,77	1,37

percentual mostrados na Tabela 10 indicam um melhor desempenho do algoritmo GG-VND em relação aos algoritmos usados para comparação.

5 CONCLUSÕES

Neste trabalho, quatro variantes da metaheurística GVNS foram estudadas para a solução do problema das *p*-Medianas Capacitado (PPMC), nomeadas G-VND, G-RVND, GG-RVND e GG-VND. Estas variantes envolvem dois métodos para a geração da solução inicial (solução inicial aleatória, como na Subseção 3.2.1, ou solução via fase de construção do GRASP, como na Subseção 3.2.2), bem como dois métodos adotados como algoritmo de busca local (método VND, mostrado no Alg. 3, e método RVND, mostrado no Alg. 4).

Tabela 11: Grupo de instâncias 2: análise de tempo de execução dos métodos.

	Resultados da literatura (s)		Tempo de execução ajustado (s)		Tempo (s)
Instância	[18]	[19]	[18]	[19]	GG-VND
lin318-005	9,15	–	13,43	–	**2,10**
lin318-015	26,35	–	38,67	–	**2,61**
lin318-040	319,46	–	468,92	–	**2,95**
lin318-070	127,45	–	187,07	–	**4,32**
lin318-100	364,87	–	535,57	–	**5,91**
ali535-005	45,42	–	66,66	–	**6,08**
ali535-025	544,26	–	798,89	–	**7,60**
ali535-050	726,3	–	1066,10	–	**11,42**
ali535-100	637,64	–	935,96	–	**18,12**
ali535-150	761,31	–	1117,49	–	**28,95**
u724-010	59,65	0,98	87,55	4,28	5,36
u724-030	300,72	6,63	441,41	28,99	**23,55**
u724-075	546,39	14,68	802,01	64,20	**26,39**
u724-125	643,31	25,99	944,28	113,67	**35,06**
u724-200	706,29	34,53	1036,72	151,02	**69,75**
rl1304-010	181,66	1,63	266,65	7,12	**49,39**
rl1304-050	1200	14,7	1761,39	64,29	**90,17**
rl1304-100	1634,2	35,04	2398,73	153,25	**79,73**
rl1304-200	1227,8	71,41	1802,19	312,32	**100,98**
rl1304-300	951,75	117,7	1397,02	514,70	**170,23**
pr2392-020	551,82	6,07	809,99	26,54	32,81
pr2392-075	825,85	51,11	1212,22	223,54	**38,09**
pr2392-150	2019,20	119	2963,93	520,38	**65,25**
pr2392-300	2382,4	171,1	3496,99	748,52	**101,32**
pr2392-500	2402,6	233	3526,66	1019,03	**200,47**
fnl4461-0020	538,97	21,24	791,12	92,89	**79,25**
fnl4461-0100	3880,4	92,58	5695,80	404,92	**75,01**
fnl4461-0250	4592,7	178,9	6741,34	782,28	**167,23**
fnl4461-0500	3912,4	286,9	5742,76	1254,73	**323,97**
fnl4461-1000	3433,3	57	5039,51	2519,18	**599,95**

Tabela 12: Grupo 2: teste de normalidade dos valores de desvio percentual $\times$ valores da literatura, considerando valores médios do GG-VND. Intervalo de confiança: $\rho = 95\%$.

Instância subgrupo	[18]	[19]	GG-VND
lin318-p	0,01361	–	0,0001
ali535-p	0,03903	–	0,0469
u724-p	0,00973	0,3420	0,0001
rl1304-p	0,2339	0,8802	0,2570
pr2392-p	0,02252	0,2829	0,4925
fnl4461-p	0,06954	0,1665	0,1882

Tabela 13: Grupo de instâncias 2: teste de significância estatística dos valores de desvio percentual × valores de literatura.

Instância subgrupo	[18]	[19]
lin318-p	**0,0454**	–
ali535-p	0,406	–
u724-p	0,6547	**0,03623**
rl1304-p	0,9283	0,9842
pr2392-p	**3,542e-04**	**2,5137e-05**
fnl4461-p	0,2547	0,9995

As variantes do GVNS foram testadas em dois conjuntos de instâncias da literatura com número elevado de depósitos e clientes, nomeadas como Grupo de instâncias 1 e Grupo de instâncias 2. Primeiramente, o desempenho destas quatro variantes foi avaliado no Grupo 1. A partir de análises estatísticas dos resultados, comprovou-se a superioridade da variante GG-VND sobre as demais. Isto se explica pelo fato de a variante GG-VND utilizar, em sua solução inicial, a fase de construção do método GRASP, que gera uma solução inicial de qualidade, ao mesmo tempo que o método VND, como busca local, promove buscas exaustivas nas vizinhanças exploradas.

Em seguida, comparou-se o desempenho da variante GG-VND com os principais algoritmos da literatura utilizados para a solução do PPMC. Para o Grupo 1, comprovou-se, após a análise estatística associada, a superioridade desta variante em relação aos algoritmos de [1], [9] e [18], no que diz respeito aos valores de desvio médio. Quanto ao tempo de execução computacional, usando-se fatores de correção para tornar justa a comparação entre as diferentes CPUs utilizadas, mostra-se que a variante GG-VND também possui melhor desempenho que os demais algoritmos analisados. Para o Grupo 2, quanto ao desvio médio relativo, a variante produziu resultados melhores em 17 das 30 instâncias analisadas. No que diz respeito ao tempo de execução, novamente utilizando fatores de correção, a variante GG-VND demandou menor tempo de execução em 28 das 30 instâncias. Portanto, conclui-se que esta variante mostrou-se fortemente competitiva frente aos algoritmos existentes na literatura para a solução do PPMC.

Agradecimentos

Os autores agradecem ao CNPq (processo 303266/2019-8), FAPEMIG (processo PPM-CEX 676/17), CAPES (Código de financiamento 001), CEFET-MG e UFOP pelo apoio a esta pesquisa.

ABSTRACT. The Capacitated p-Median Problem (CPMP) consists of locating p facilities in a network composed of n clients and deciding which facility will serve each client to minimize the sum of all distances from each facility to each client and meet the facility capacity constraints. Since the CPMP belongs to the NP-hard class, in this work, four variants of the General Variable Neighborhood Search (GVNS) metaheuristic, named G-VND, G-RVND, GG-VND, and GG-RVND, are implemented for treating the CPMP. They

differ concerning the method used to build an initial solution and the one used for local search. In the first two variants, the initial solution is generated randomly, and the local search method is performed via Variable Neighborhood Descent (VND) or Random Variable Neighborhood Descent (RVND), respectively. In turn, in the last two, the initial solution is executed via the construction phase of the Greedy Randomized Adaptive Search Procedure (GRASP) method. Using benchmark instances from the literature, we initially showed that the GG-VND variant performed better than others. Then, we show that this variant has an equivalent or superior performance when compared with the literature algorithms.

Keywords: Capacitated p-Median Problem, General Variable Neighborhood Search, GRASP, metaheuristics.

REFERÊNCIAS

[1] A.A. Chaves & L.A.N. Lorena. Clustering search algorithm for the capacitated centered clustering problem. *Computers & Operations Research*, **37**(3) (2010), 552–558. Hybrid Metaheuristics.

[2] E.S. Correa, M.T.A. Steiner, A.A. Freitas & C. Carnieri. A Genetic Algorithm for Solving a Capacitated p-Median Problem. *Numerical Algorithms*, **35**(2) (2004), 373–388.

[3] J.A. Díaz & E. Fernández. Hybrid scatter search and path relinking for the capacitated p-median problem. *European Journal of Operational Research*, **169**(2) (2006), 570–585.

[4] T.A. Feo & M.G.C. Resende. Greedy Randomized Adaptative Search Procedures. *Journal of Global Optimization*, **6** (1995), 109–133.

[5] K. Fleszar & K. Hindi. An effective VNS for the capacitated p-median problem. *European Journal of Operational Research*, **191**(3) (2008), 612 – 622.

[6] S. Hakimi. Optimum Locations of Switching Center and the Absolute Centers and Medians of a Graph. *Operations Research*, **12** (1964), 45–59.

[7] P. Hansen, N. Mladenović, R. Todosijević & S. Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, **5** (2017), 423–454. doi:10.1007/s13675-016-0075-x.

[8] M. Herda. Combined genetic algorithm for capacitated p-median problem. In "2015 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)" (2015), p. 151–154.

[9] L. Jánošíková, M. Herda & M. Haviar. Hybrid genetic algorithms with selective crossover for the capacitated p-median problem. *Central European Journal of Operations Research*, **25**(3) (2017), 651–664. doi:10.1007/s10100-017-0471-1.

[10] O. Kariv & S.L. Hakimi. An Algorithmic Approach to Network Location Problems. II: The p-Medians. *SIAM Journal on Applied Mathematics*, **37**(3) (1979), 539–560. doi:10.2307/2100911.

[11] I. Landa-Torres, J.D. Ser, S. Salcedo-Sanz, S. Gil-Lopez, J. Portilla-Figueras & O. Alonso-Garrido. A comparative study of two hybrid grouping evolutionary techniques for the capacitated P-median problem. *Computers & Operations Research*, **39**(9) (2012), 2214–2222. doi:https://doi.org/10.1016/j.cor.2011.11.004.

[12] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari & T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, **3** (2016), 43–58.

[13] L.A.N. Lorena & E.L.F. Senne. A Column Generation Approach to Capacitated P-median Problems. *Computers & Oper. Research*, **31**(6) (2004), 863–876.

[14] A. Mjirda, R. Todosijević, S. Hanafi, P. Hansen & N. Mladenović. Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. *Int. Transactions in Operational Research*, **24** (2017), 615–633.

[15] D.C. Montgomery & G.C. Runge. "Estatística aplicada e probabilidade para engenheiros". LTC, 6 ed. (2016). ISBN 978-85-216-3241-2.

[16] H.I. Osman & N. Christofides. Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, **1**(3) (1994), 317–336.

[17] M.J.F. Souza, I.M. Coelho, S. Ribas, H.G. Santos & L.H.C. Merschmann. A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, **207**(2) (2010), 1041–1051.

[18] F. Stefanello, O.C.B. de Araújo & F.M. Müller. Matheuristics for the capacitated p-median problem. *International Transactions in Operational Research*, **22**(1) (2015), 149–167. doi:10.1111/itor.12103.

[19] M. Steglich. A Hybrid Heuristic Based On Self-Organising Maps And Binary Linear Programming Techniques For The Capacitated P-Median Problem. In M. Iacono, F. Palmieri, M. Gribaudo & M. Ficco (editors), "Proceedings of the 33rd International ECMS Conference on Modelling and Simulation, ECMS 2019 Caserta, Italy, June 11-14, 2019". European Council for Modeling and Simulation (2019), p. 267–276. doi:10.7148/2019-0267. URL `https://doi.org/10.7148/2019-0267`.

[20] A. Subramanian, L.M.A. Drummond, C. Bentes, L.S. Ochi & R. Farias. A parallel heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. *Computers & Operations Research*, **37**(11) (2010), 1899 – 1911.

[21] M.B. Teitz. Toward a theory of urban public facility location. *Papers in Regional Science*, **21**(1) (1968), 35–51.

[22] M. Yaghini, M. Momeni, M. Sarmadi & H.R. Ahadi. An efficient heuristic algorithm for the capacitated p-median problem. *4OR*, **11**(3) (2013), 229–248.

[23] M. Yaghini, M. Momeni, M. Sarmadi & H.R. Ahadi. A hybrid metaheuristic approach for the capacitated p-median problem. *Applied Soft Computing*, **13**(9) (2013), 3922–3930.