LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

Uma Nova Restrição para o Problema Leasing k Median e Algoritmos Baseados em Metaheurísticas e Paralelismo

Lucas Gabriel de Godoi Arriel
Pontifícia Universidade Católica de Goiás
Av. Universitária 1440 - Setor Leste Universitário, Goiânia - GO, 74175-120
lucasgabrielgarriel@hotmail.com

Alexandre Ribeiro
Pontifícia Universidade Católica de Goiás
Av. Universitária 1440 - Setor Leste Universitário, Goiânia - GO, 74175-120
alexribeiro@pucgoias.edu.br

RESUMO

Definir locações de facilidades é um problema comum nas etapas de planejamento, tal etapa pode gerar uma grande diferença no custo final do projeto. Um modelo de Programação Linear Inteira (PLI) foi apresentado para esse esse problema e abordado como Leasing k Median (LKM). Esta pesquisa apresenta uma nova abordagem ao LKM, de forma a refletir melhor sua proposta original. 5 diferentes metaheurísticas implementadas com paralelismo e um *solver* de métodos exatos foram utilizados para encontrar soluções desse problema, onde as metaheurísticas conseguiram igualar ou ultrapassar 70% dos resultados do *solver*, além de encontrar resultados para 10 instâncias que o *solver* não encontrou em um tempo estipulado.

PALAVRAS CHAVE. Leasing k Median, Metaheurísticas, GVNS.

MH - Metaheurísticas, OC - Otimização Combinatória

ABSTRACT

Defining facilities location is a common step on project planning, which can make a big difference in the project's final cost. An Integer Linear Programming Model was presented for this problem and defined as Leasing k Median (LKM). This research shows a new approach to the LKM, so it reflects better the original propose. 5 different metaheuristics implemented with parallelism and a solver of exact methods were used to get solutions to this problem, where the metaheuristics were able to achieve 70% of the solver's results and found results to 10 other instances which the solver couldn't find within time limit.

KEYWORDS. Leasing k Median, Metaheuristics, GVNS.

MH - Metaheuristics, OC - Combinatorail Optimization

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

1. Introdução

O problema de otimização combinatória Leasing k Median (LKM), proposto por Lintzmayer e Mota [2017] e formulado em dos Santos et al. [2019], é uma generalização do conhecido problema k Median na literatura, ambos *NP-Difíceis*.

O LKM tem aplicações em diversos cenários da vida real, como a distribuição estratégica de pontos de vacinação e exames em áreas urbanas ou rurais, onde o acesso a esses serviços pode ser dificultado por fatores geográficos ou econômicos. Outro exemplo prático seria a localização de postos de bombeiros em regiões propensas a incêndios florestais, cuja ocorrência é sazonal e dependente da vegetação predominante.

O problema k Median pode ser descrito como a seleção de até k pontos estratégicos para a instalação de facilidades em um espaço métrico (V, d). Essas facilidades são selecionadas de forma a minimizar a soma das distâncias que cada clientes j, que pertencem a um conjunto D que é um subconjunto de V, percorre até a facilidade mais próxima dele [Kariv e Hakimi, 1979].

No entanto, no LKM, os clientes estão distribuídos ao longo de um conjunto de instantes T, onde $T = \{0, ..., t_{\max}\}$, e existem $L = \{1, ..., \delta_{l_{\max}}\}$ tipos de facilidades que podem ser abertas em um determinado instante $t \in T$. Cada tipo de facilidade, permanece ativa por δ_{l_i} instantes, ou até $t_{\max}$. Assim, o objetivo é determinar um conjunto de facilidades $M \subseteq V \times T \times L$, onde a quantidade de facilidades ativas em um único instante não excede o limite k, de tal forma que minimize a soma das distâncias de todos os clientes $j \in D_t$ para todo instante t.

A Figura 1 ilustra um exemplo de instância e solução. Essa instância é composta por $|T| = 4$ instantes de tempo, numerados de 0 a 3, $|V| = 7$ pontos para abertura de facilidades, $k = 2$ e somente um tipo de facilidade, com duração $\delta = 2$. Os clientes são representados pelas cores azuis, as facilidades selecionadas estão contornadas em vermelho e a distância entre os pontos até as facilidades mais próximas são representadas pelos pesos nas arestas. Na solução foram abertas facilidades nos pontos 3 e 5 no instante 0, permanecendo ativas até o instante 1; e em seguida foram abertas facilidades nos pontos 2 e 4 no instante 2, permanecendo ativas até o instante 3.

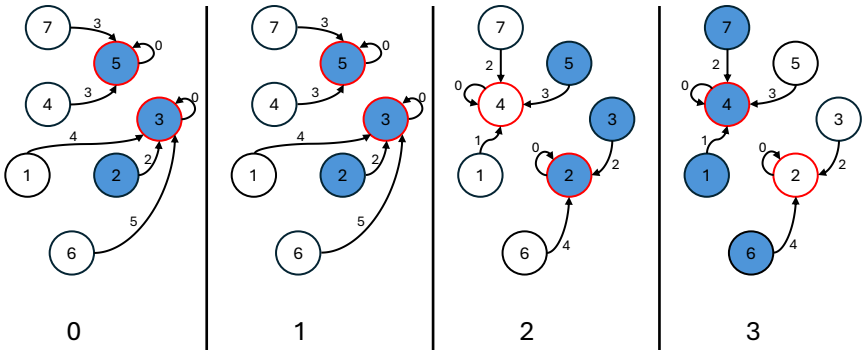

Figura 1: Exemplo de solução para uma instânica do LKM.

No trabalho de dos Santos et al. [2019], mais de uma facilidade pode estar ativa em um mesmo local durante o mesmo instante de tempo. Esta pesquisa aborda o LKM introduzindo uma nova restrição que limita a uma facilidade estar ativa em um local para cada instante de tempo. Observe que sem essa restrição, no exemplo apresentado na Figura 1, seria possível abrir uma facilidade no ponto 3 durante o instante 0 e abrir outra no mesmo ponto no instante 1; em seguida abrir uma facilidade no ponto 6 ao invés do ponto 2. Além disso, novas abordagens heurísticas com uso de *multithreading* para encontrar soluções do problema foram desenvolvidas.

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

O restante deste artigo está organizado como segue. A Seção 2 apresenta a formulação e a complexidade do problema, assim como a nova restrição abordada neste trabalho. A Seção 3 apresenta os métodos utilizados para a resolução do problema. Na Seção 4 é apresentado testes e resultados deles, a fim de realizar uma comparação na performance de cada um. Por fim, a Seção 5 discute o que foi desenvolvido nesta pesquisa e propõe abordagens para trabalhos futuros.

2. O Problema Leasing k Median

Esta seção apresenta conceitos bases e a formulação do Leasing k Median (LKM).

De acordo com dos Santos et al. [2019] o LKM considera um espaço métrico (V, d) e um conjunto de instantes T, junto ao conjunto de tipos de facilidades L, apresentados na Seção 1, tais facilidades podem atender clientes durante os instantes em que permanece ativa. Considere o último instante de uma facilidade ativa como $\gamma = [t, \min(t + \delta_l, t_{max} + 1))$, sendo $t \in T$ o instante em que a facilidade foi aberta. O modelo considera que os clientes escolhem a facilidade mais próxima deles, que pode ser definido por:

$$\sum_{t \in T} \sum_{j \in D_t} \min_{\substack{(i,l,t') \in M; \\ t' \in [t', \gamma)}} d(i, j)$$

Onde $d : (i, j) \to \mathbb{R}$ representa a distância entre o vértice i e j.
É utilizada a variável binária $x^t_{i,j}$ para representar:

$$x^t_{i,j} = \begin{cases} 1, & \text{se o cliente } j \text{ é atendido pela} \\ & \quad \text{facilidade } i \text{ no instante t} \\ 0, & \text{se não} \end{cases} \qquad \forall i \in V, \forall j \in D_t, \forall t \in T$$

A variável binária $y^t_{i,l}$ é introduzida para indicar os seguintes casos:

$$y^t_{i,l} = \begin{cases} 1, & \text{se a facilidade } i \text{ de tipo } l \\ & \quad \text{foi aberta no instante t} \\ 0, & \text{se não} \end{cases} \qquad \forall i \in V, \forall l \in L, \forall t \in T$$

Para simplificar a formulação considere $p^t_l = t - \delta_l + 1$, se $\delta_l \leq t$; ou $p^t_l = 1$, caso contrário. Pode-se então construir o seguinte modelo de programação linear inteira:

$$\text{minimizar} \sum_{t \in T} \sum_{i \in V} \sum_{j \in D_t} x^t_{i,j} d(i, j) \tag{1}$$

$$\text{sujeito a:} \quad \sum_{i \in V} x^t_{i,j} = 1 \qquad \forall t \in T, \forall j \in D_t \tag{2}$$

$$\sum_{t \in T} \sum_{l \in L} \sum_{t'=p^t_l}^{t} y^{t'}_{i,l} \leq k \qquad \forall t \in T \tag{3}$$

$$\sum_{l \in L} \sum_{t'=p^t_l}^{t} y^{t'}_{i,l} \geq x^t_{i,j} \qquad \forall i \in V, \forall j \in D_t, \forall t \in T \tag{4}$$

$$\sum_{l \in L} \sum_{l \in L} \sum_{t'=p^t_l}^{t} y^{t'}_{i,l} \leq 1 \qquad \forall i \in V, \forall t \in T \tag{5}$$

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

A função objetivo (1) tem como alvo minimizar a soma das distâncias percorridas por cada cliente até a facilidade que ele está sendo atendido. As restrições em (2) implicam que somente uma facilidade i pode servir o cliente j no instante t. As restrições em (3) garantem que k ou menos facilidades estejam ativas em um instante de tempo t. As restrições em (4) garantem que a facilidade i atende o cliente j somente se ela estiver ativa no instante t.

Esta pesquisa propõe as restrições em (5), que limitam a no máximo uma única facilidade estar ativa na localidade i durante o instante t. Dessa forma, o modelo de PLI atende melhor o problema proposto em Lintzmayer e Mota [2017].

Do ponto de vista de complexidade, o problema k-Median pertence a classe NP-Difícil. Haja vista que esse problema é o caso particular do LKM quando $|T| = 1$, o LKM também é NP-Difícil [dos Santos et al., 2019].

3. Abordagens Usadas

Nesta pesquisa, o problema LKM foi abrodado de forma exata e de forma heurística. Como algoritmo exato foi utilizado um *solver* de PLI para obter uma soluções ótima através da formulação apresentada na Seção 2. Do ponto ponto de vista de heurísticas, foram propostos algoritmos baseados nas metaheurísticas *Genetic Algorithm* (GA), *General Variable Neighborhood Search* (GVNS) e *Tabu Search* (TS); além disso, duas hibridizações entre o GA e GVNS, e entre o GVNS e TS. Todas as heurísticas utilizam de *multithreading* para acelerar o processo de resolução, ou para permitir explorar mais regiões do espaço de soluções. As soluções iniciais são construídas de forma aleatória, através da função GeraSolucoes(), e todas utilizaram 16 threads em sua implementação.

3.1. Programação Linear Inteira

O *solver* utilizado foi o *Gurobi* [Gurobi Optimization, LLC, 2023]. O *Gurobi* é um software *black box*, que utiliza algoritmos que representam o estado da arte para problemas de PLI. O *Gurobi* sempre consegue encontrar uma solução ótima, entretanto, ele não garante quando essa solução será encontrada, podendo levar um tempo significativo para convergir o resultado ótimo.

3.2. Busca Local

A busca local desenvolvida para melhorar as soluções foi baseada na técnica de *first improvement*, que utiliza movimentos para trocar os locais onde as facilidades devem ser abertas. O pseudocódigo da busca local é apresentado no Algoritmo 1, onde S é a solução em que será aplicada a busca local.

Algoritmo 1: Busca Local baseada em *first improvement*

```
1 Função BuscaLocal(S):
2     S' ← S
3     enquanto S' for igual ou pior que S faça
4         S' ← S
5         para Cada local de abertura de facilidade i em S' faça
6             Troque i por algum v ∈ V aleatório, tal que v ≠ i
7         se S' é inviável então
8             ConsertaSolucao(S')
9     retorna S'
```

Caso a solução S' seja inviável por causa do movimento na linha 6 ou algum outro movimento anterior, a solução passa por um processo de conserto, que a torna viável, descrito no Algoritmo 2. São selecionadas as melhores facilidades na solução sem violar alguma restrição e depois inseridas novas facilidades ou a abertura de outras facilidades é adiantada.

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

Algoritmo 2: Conserta Solucao

```
1  Função ConsertaSolucao(S):
2      S' ← ∅
3      Insira em S' as facilidades que mais atendem clientes em S sem violar as
         restrições do problema
4      t ← 0
5      enquanto t ≤ T faça
6          t ← Próximo instante onde há menos que k facilidade abertas
7          se for possível incluir uma facilidade no instante t então
8              Adicione uma facilidade de forma aleatória no instante t em S'
9          senão
10             Insira a primeira facilidade que foi aberta após o instante t em S'
11             t ← t + 1
12     retorna S'
```

3.3. Parallel Genetic Algorithm (PGA)

Inspirado na teoria da evolução Darwiniana, o GA é amplamente utilizado em problemas de otimização combinatória, junto a diversas variações e técnicas [Katoch et al., 2021]. O GA utilizado neste trabalho foi inspirado em Harada e Alba [2020] e Wang et al. [2022] que apresentam o GA de forma paralela, e Song e Xiao [2012] que utiliza um GA adaptativo. O PGA proposto nessa pesquisa segue o pseudocódigo apresentado no Algoritmo 3.

Algoritmo 3: PGA adaptado de Harada e Alba [2020], Wang et al. [2022] e Song e Xiao [2012]

```
1  Função GA (tamanho_populacao, elitismo, qnt_threads):
2      populacao ← GerarSolucoes(tamanho_populacao)
3      enquanto condição de parada não atingida faça
4          nova_populacao ← populacao
5          BuscaLocal(nova_populacao)
6          Cruzamento(nova_populacao)
7          Mutação(nova_populacao)
8          Ordene populacao e nova_populacao
9          Substitua os piores indivíduos da populacao pelos melhores da
             nova_populacao
10     retorna a melhor solução da populacao
```

Nesta pesquisa, para gerar os melhores resultados, foi utilizada a probabilidade adaptativa de cruzamento e mutação, proposta por Song e Xiao [2012]. A probabilidade segue a equação (6). Onde $k_2 = 0.25$ e $k_1 =$ valor entre 0 e k_2, f é o valor atual da função objetivo, f_{avg} é o valor médio e f_{max} o melhor valor.

$$p = \begin{cases} k_2 & \text{se } f < f_{avg} \\ k_1 \times \max(f_{avg} - f_{max}, 1) / \max(f - f_{max}, 1) & se f \geq f_{avg} \end{cases} \quad (6)$$

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

O tamanho da população e o fator de elitismo foram, respectivamente, 48 e 15%. Esses valores foram escolhidos de forma empírica.

O cruzamento é realizado através da seleção entre dois métodos, ambos com igual probabilidade de serem escolhidos: a seleção de um único ponto ou a seleção de múltiplos pontos. Os pais são selecionados através de um torneio com 5 candidatos. Já a mutação é realizada alterando uma solução de um individuo aleatório, que consiste na troca do instante de abertura, localidade e tipo de uma facilidade. A população foi divida em 16 blocos, para cada *thread* executar tais procedimentos.

3.4. Parallel General Variable Neighborhood Search (PGVNS)

O GVNS é baseado em uma busca na vizinhaça de soluções através de um *Variable Neighborhood Descentend* (VND). O GVNS obteve bons resultados para diversos problemas de otimização combinatória [Karakostas et al., 2019]. dos Santos [2019] também utilizou uma versão do *Variable Neighborhood Search* para o problema original, sem a restrição tratada neste trabalho, e conseguiu bons resultados.

O GVNS utilizado neste trabalho tem o pseudocódigo apresentado no Algoritmo 4. Neste algoritmo, cada *thread* executa uma instância do GVNS de forma independente. O VND utilizado na linha 9 segue o proposto em Gendreau e Potvin [2013], porém, com a busca local do Algoritmo 1. Os valores de r e $lmax$ são, respectivamente, 15 e 48 para todas as *threads*. Ambos valores foram escolhidos de forma empírica.

Algoritmo 4: GVNS

```
1  Função GVNS (r, lmax) :
2      S ← GerarSolucoes(1)
3      enquanto condição de parada não atingida faça
4          S' ← S
5          k ← 0
6          enquanto k ≤ r e condição de parada não atingida faça
7              SS ← S'
8              Embaralhe(SS, k + 1)
9              SS ← VND(SS, lmax)
10             se SS for melhor que S' então
11                 S' ← SS
12                 k ← 0
13             senão
14                 k ← k + 1
15         se S' for melhor que S então
16             S ← S'
17     retorna S
```

3.5. Parallel Tabu Search (PTS)

Com um princípio semelhante ao do GVNS, o *Tabu Search* (TS) parte da busca em vizinhança, porém, os vizinhos selecionados são guardados em uma lista tabu por um determinado tempo, com o intuito de que não sejam visitados novamente [Gendreau e Potvin, 2013]. Além disso, a fase de embaralhar as soluções pode ser omitida a fim de explorar melhor as vizinhanças.

O TS recebe como entrada dois inteiros r e k, que são, respectivamente, a quantidade de vizinhos a serem explorados e o tamanho da lista tabu, ambos foram escolhidos de forma emprírica,

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

onde $r = 5$ e $k = 10$. O Algoritmo 5 define o pseudocódigo do TS. Assim como no GVNS, 16 *threads* executando o TS de forma paralela e independente. Nesta pesquisa, a versão do TS utilizada foi adaptada de Talbi et al. [1998], que possui uma lista tabu com tamanho adaptativo.

Algoritmo 5: PTS adaptado de Talbi et al. [1998]

```
1  Função TS(r, k):
2      S ← GerarSolucoes(1)
3      S* ← S
4      contador ← 0
5      enquanto condição de parada não atingida faça
6          S ← Melhor vizinho de S que não seja tabu ou respeite uma função de
             aspiração
7          Atualize a Lista Tabu
8          se S for melhor que S* então
9              S* ← S
10             Volte a Lista Tabu e o contador ao seu tamanho original
11         senão
12             contador ← contatdor + 1
13         se contador = k então
14             Escolha entre aumentar ou limpar a Lista Tabu de forma aleatória e com
                 probabilidade igual
15     retorna S*
```

3.6. Híbrido PGA - PGVNS

Também foi desenvolvida uma versão híbrida do GA com o GVNS, que aproveita do ambiente *multithread*. Nessa versão, o GA e o GVNS executam de forma paralela, comunicando-se a cada 3 minutos, salvando as melhores soluções e executando a otimização nelas. Observe que cada algoritmo também utiliza *multithreading*.

O Algoritmo 6 apresenta o algoritmo principal, responsável por iniciar as *threads* do PGA e PGVNS, além de realizar a seleção das melhores soluções. Como 2 *threads* são responsáveis para executar cada algoritmo e ambos usam diferentes *threads* em sua execução, foi atribuído que cada algoritmo tivesse 14 *threads* para serem utilizadas em seus algoritmos de otimização. O tempo limite é utilizado como condição de parada para os algoritmos, os valores dos parâmetros, escolhidos de forma empírica, foram $tamanho\_populacao = 32$, $elitismo = 6$, $r = 15$, $lmax = 48$.

3.7. GVNS com Lista Tabu (GVNTS)

Moreno Pérez et al. [2003] aplica ao *Median Cycle Problem* uma versão do GVNS que utiliza o embaralhamento tabu, ou seja, uma série de embaralhamento é feita nos vizinhos de uma solução, e é selecionado o melhor embaralhamento que não foi inserido na lista tabu, ou segue uma função de aspiração. Matijević et al. [2023] conseguiu bons resultados para um problema de roteamento de caminhões em portos. Baseado nisso, também propomos uma versão desse algoritmo para o LKM.

O Algoritmo 7 apresenta o pseucódigo do GVNTS. Note que a única diferença entre o GVNS normal se dá na linha 9. Os parâmetros r e $lmax$ seguem o apresentado na Seção 3.4, o parâmetro $tabu\_k$ é o tamanho da lista tabu, sendo, também, o mesmo escolhido na seção 3.5. Assim como no GVNS e TS, este algoritmo executa de forma paralela em 16 *threads* independentes.

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

Algoritmo 6: PGA-PGVNS

```
1  Função GAGVNS (tamanho_populacao, elitismo, qnt_threads, r, lmax, tempo_limite) :
2      populacao ← GerarSolucoes(tamanho_populacao)
3      enquanto condição de parada não atingida faça
4          thread t1, t2
5          SolucoesGA ← Individuos(populacao)
6          SolucoesVNS ← Vizinhança(populacao)
7          t1.iniciar(GA(SolucoesGA, elitismo, qnt_threads, tempo_limite))
8          t2.iniciar(GVNS(SolucoesVNS, r, lmax, qnt_threads, tempo_limite))
9          Espere por t1 e t2 finalizarem
10         populacao ← EscolherMelhores(SolucoesGA, SolucoesVNS)
11     retorna a melhor solução da populacao
```

Algoritmo 7: GVNTS

```
1  Função GVNTS (r, lmax, tabu_k) :
2      S ← GerarSolucoes(1)
3      enquanto condição de parada não atingida faça
4          S' ← S
5          k ← 0
6          contador ← 0
7          enquanto k ≤ r e condição de parada não atingida faça
8              SS ← S'
9              Embaralhe k + 1 vizinhos de SS e selecione o que não seja tabu ou
                respeite uma função de aspiração
10             Atualize a Lista Tabu
11             SS ← VND(SS, lmax)
12             se SS for melhor que S' então
13                 S' ← SS
14                 k ← 0
15                 Volte a Lista Tabu e o contador ao seu tamanho original
16             senão
17                 k ← k + 1
18                 contador ← contador + 1
19             se contador = k então
20                 Escolha entre aumentar ou limpar a Lista Tabu de forma aleatória
21         se S' for melhor que S então
22             S ← S'
23     retorna S
```

4. Resultados e Experimentos Computacionais

Essa seção apresenta as instâncias de testes utilizadas e os resultados obtidos para as heurísticas apresentadas na Seção 3. Os testes foram executados em uma máquina pessoal com processador i5-12500H de 2.50 GHz, 24 GB de memória RAM utilizando o subsistema Linux WSL2

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

para Windows 11. O código do *Gurobi* foi desenvolvido utilizando o interpretador Python 3.9, já as heurísticas foram desenvolvidas na linguagem C++ 20. Todos algoritmos foram executados por 15 minutos no total, e cada instância foi executada pelo menos 10 vezes.

4.1. Instâncias de Testes

Para realizar os experimentos, utilizamos um conjunto de 30 instâncias do LKM de dos Santos [2019]. Elas estão divididas em 3 grupos, onde o 1° grupo representa isntâncias pequenas, o 2° instâncias médias e o 3° instâncias grandes. Note que a inclusão da nova restrição proposta neste trabalho não inviabiliza o uso dessas instâncias.

O grupo 1 é formado por instâncias onde o *Gurobi* consegue encontrar uma solução ótima dentro do tempo limite. Já o grupo 2 é formado por aquelas em que o *solver* não consegue encontrar uma solução ótima, mas obtém pelo menos uma solução viável dentro do tempo limite. Por fim, o grupo 3 possui instâncias onde o *Gurobi* não consegue encontrar alguma solução viável dentro do tempo limite.

A Tabela 1 descreve os detalhes das instâncias de forma mais técnica para os grupos 1 e 2, e a Tabela 2 descreve as instâncias do grupo 3.

Tabela 1: Definição dos grupos de instâncias 1 e 2

Grupo 1						Grupo 2					
Id	$\|V\|$	T	k	$\|L\|$	$\sum\|D_t\|$	Id	$\|V\|$	T	k	$\|L\|$	$\sum\|D_t\|$
1	100	7	5	4	420	11	170	25	15	8	1923
2	100	10	5	2	491	12	170	36	27	5	2990
3	100	1	5	1	53	13	172	33	13	8	3037
4	100	2	5	2	90	14	183	25	12	9	2049
5	100	4	5	2	125	15	127	31	18	14	1963
6	100	12	10	3	725	16	193	19	11	8	2069
7	100	8	10	5	262	17	201	16	8	12	1427
8	100	5	10	1	193	18	230	8	4	9	1104
9	100	15	10	1	958	19	200	17	10	7	1440
10	100	11	10	2	479	20	200	18	10	10	2029

Tabela 2: Definição do grupo de instâncias 3

Grupo 3					
Id	V	T	k	L	Dt
21	218	86	23	25	8751
22	268	89	8	21	12711
23	205	101	32	10	10754
24	322	31	8	8	4546
25	312	141	10	16	22980
26	348	42	19	21	7128
27	33	38	19	9	6350
28	493	50	7	9	13268
29	232	92	29	25	9870
30	503	30	11	7	7566

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

Tabela 3: *gaps* do grupo de instancias 1

Id	GUROBI	PGVNS	PGA	PTS	PGA-PGVNS	PGVNTS
1	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
2	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
3	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
4	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
5	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
6	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
7	**0,00**	0,12	0,31	0,13	0,14	0,18
8	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
9	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
10	**0,00**	0,34	0,50	0,43	0,41	0,41
Média	**0,00**	0,04	0,08	0,05	0,05	0,05

4.2. Resultados dos testes

A fim de comparar os resultados, foi utilizado a formula $gap = (S - S_0)/S \times 100$, para representar a diferença entre os resultados, onde S é a solução a ser comparada e S_0 é a melhor solução encontrada dentre todos os algoritmos.

A Tabela 3 contém os *gaps* das instâncias do grupo 1. Todas heurísticas conseguiram encontrar valores ótimos para 8 instâncias, e nas demais ficaram com *gap* igual ou menor que 0.5%. A média dos *gaps* de todas as heurísticas ficou abaixo de 0.1%, o PGVNS obteve a melhor média (0.04%) dentre elas.

A Tabela 4 contém os resultados das instâncias do grupo 2. Nesse grupo, todos algoritmos obtiveram uma média de resultados melhor que o *Gurobi*, e, novamente, o PGVNS sobressaiu com média de 0.31%. O PGVNS obteve as melhores soluções que todos os outros algoritmos em 3 instâncias, o GA-GVNS em 2 e o TS em 1 instância. Vale ressaltar que, o *gap* para as instâncias 14 e 18 entre o PGVNS e GA-GVNS ficou em 0.01%.

Tabela 4: *gaps* do grupo de instancias 2. * indica que o gurobi achou a solução ótima

Id	*Gurobi*	PGVNS	PGA	PTS	PGA-PGVNS	PGVNTS
11	1,49	**0,00**	0,21	0,37	0,04	0,06
12	**0,00**	0,79	3,19	4,98	0,97	0,96
13	2,08	0,05	0,34	0,48	**0,00**	0,06
14	4,27	0,01	0,16	0,06	**0,00**	0,07
15	**0,00**	0,44	0,81	0,88	0,37	0,44
16	6,86	0,04	0,08	**0,00**	0,05	0,14
17	14,82	**0,00**	0,40	0,10	0,08	0,21
18	12,08	**0,00**	0,08	0,03	0,01	0,08
19	**0,00***	0,56	0,77	0,58	0,78	0,67
20	**0,00***	1,17	1,18	1,34	1,41	1,19
Média	4,16	**0,31**	0,72	0,88	0,37	0,38

A Tabela 5 contém os *gaps* das instâncias do grupo 3. Todos algoritmos chegaram a uma solução viável para todas as instâncias. De forma que, novamente, o PGVNS obteve a melhor média (0.14%). Somente o TS não obteve uma média abaixo de 1.00%, porém, conseguiu obter 3 soluções

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

com o melhor valor dentre os obtidos. O PGVNS e o GA-GVNS obtiveram 3 melhores soluções e o GVNTS obteve 1 melhor solução.

A fim de comparações mais detalhadas dos resultados obtidos, os valores da função objetivo para cada instância e uma comparação com o algoritmo de dos Santos [2019] estão disponíveis em `github.com/LucasGabriel-CP/TCC/`, assim como a implementação de todos algoritmos propostos neste trabalho.

Tabela 5: *gaps* do grupo de instancias 3

Id	PGVNS	PGA	PTS	PGA-GVNS	PGVNTS
21	0,03	2,40	3,19	0,31	**0,00**
22	0,20	0,27	0,49	**0,00**	0,31
23	0,39	0,81	**0,00**	0,71	0,76
24	0,24	0,18	**0,00**	0,01	0,18
25	0,09	0,14	1,84	**0,00**	0,24
26	**0,00**	0,50	0,92	0,15	0,02
27	0,02	0,28	0,35	**0,00**	0,07
28	**0,00**	0,56	1,02	0,26	0,13
29	**0,00**	2,16	2,72	0,35	0,06
30	0,44	0,67	**0,00**	0,61	0,68
Média	**0,14**	0,80	1,05	0,24	0,25

5. Conclusão

Nesta pesquisa, uma nova restrição foi adicionada ao *Leasing K Median*, de forma que o modelo atenda melhor a proposta definida em Lintzmayer e Mota [2017]. Também, foram desenvolvidas diferentes abordagens para o LKM, sendo elas, 5 metaheurísticas de diferentes tipos, incluindo técinicas de híbridização e de paralelismo, e o uso do *Gurobi* para soluções exatas.

As metaheurísticas obtiveram 6 soluções melhores que o *solver* de PLI e uma média de *gap* melhor que o *solver* para o grupo 2, além de encontrar soluções para instâncias, as quais, o *solver* não encontrou, dentro de 15 minutos de execução. Dentre todas as metaheurísticas propostas, o PGVNS obteve a melhor média geral, em 0.16%, seguida por uma das hibridizações desse mesmo algoritmo, o PGA-PGVNS, que ficou com 0.21% de média geral.

Para trabalhos futuros, é possível adicionar novas restrições de forma acrescentar demandas diferentes para cada cliente e, também, uma capacidade máxima de atendimento das facilidades. Também uma outra possível modificação fica nos parâmetros dos algoritmos, fazendo análise mais profunda da performance de cada um.

Referências

dos Santos, J. (2019). Heurísticas para o problema leasing k -median. Trabalho de conclusão de curso, Pontifícia Universidade Católica de Goiás.

dos Santos, J., Londe, G., Ribeiro, A., e da Silva, W. (2019). Formulações e heurísticas para os problemas leasing k-median e leasing k-center. In *Anais do IV Encontro de Teoria da Computação*, Porto Alegre, RS, Brasil. SBC.

Gendreau, M. e Potvin, J.-Y. (2013). *Handbook of Metaheuristics*. Springer Cham.

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en

Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual. URL `https://www.gurobi.com`.

Harada, T. e Alba, E. (2020). Parallel genetic algorithms: A useful survey. *ACM Comput. Surv.*, 53.

Karakostas, P., Sifaleras, A., e Georgiadis, M. C. (2019). A general variable neighborhood search-based solution approach for the location-inventory-routing problem with distribution outsourcing. *Computers & Chemical Engineering*, 126:263–279.

Kariv, O. e Hakimi, S. L. (1979). An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37:513–538.

Katoch, S., Chauhan, S. S., e Kumar, V. (2021). A review on genetic algorithm: past, present, and future. In *Multimedia Tools and Applications*.

Lintzmayer, C. N. e Mota, G. O. (2017). Caderno de problemas. In *1o Workshop Paulista em Otimização, Combinatória e Algoritmos*. [S.l, S.n].

Matijević, L., Durasević, M., e Jakobović, D. (2023). A variable neighborhood search method with a tabu list and local search for optimizing routing in trucks in maritime ports. *Mathematics*, 11.

Moreno Pérez, J. A., Marcos Moreno-Vega, J., e Rodríguez Martín, I. (2003). Variable neighborhood tabu search and its application to the median cycle problem. *European Journal of Operational Research*, 151:365–378. Meta-heuristics in combinatorial optimization.

Song, X. e Xiao, Y. (2012). An improved adaptive genetic algorithm. *2012 First National Conference for Engineering Sciences (FNCES 2012)*, p. 0796–0799.

Talbi, E. G., Hafidi, Z., e Geib, J. M. (1998). A parallel adaptive tabu search approach. *Parallel Computing*, 24:2003–2019.

Wang, K. M., Wang, K.-J., e Chen, C. C. (2022). Capacitated production planning by parallel genetic algorithm for a multi-echelon and multi-site tft-lcd panel manufacturing supply chain. *Applied Soft Computing*, 127:109–371.

DOI: 10.59254/sbpo-2024-193842
https://proceedings.science/p/193842?lang=en