PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

HEURÍSTICAS PARA O PROBLEMA LEASING k-MEDIAN

Jorge Menezes dos Santos

GOIÂNIA
2019

Jorge Menezes dos Santos

HEURÍSTICAS PARA O PROBLEMA LEASING k-MEDIAN

Trabalho de Conclusão de Curso apresentado à Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Alexandre Ribeiro
Coorientador: Me. Welverton Rodrigues da Silva

GOIÂNIA
2019

Jorge Menezes dos Santos

HEURÍSTICAS PARA O PROBLEMA LEASING k-MEDIAN

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do grau de Bacharel em Ciência da Computação e aprovado em sua forma final pela Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, em 05 de dezembro de 2019.

Profª. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

Orientador: Prof. Me. Alexandre Ribeiro

Profª. Dra. Carmen Cecilia Centeno

Prof. Me. Max Gontijo de Oliveira

GOIÂNIA
2019

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por sempre ter me guiado e por ter possibilitado que tudo acontecesse.

Aos meus pais e ao meu irmão por sempre estarem presentes me motivando e me dando forças para continuar.

Ao meu professor, orientador, treinador de maratona e amigo Alexandre Ribeiro por todos os ensinamentos, conselhos, sugestões, incentivos, apoio e, principalmente, pela paciência que teve comigo. Ao meu coorientador e companheiro de maratona Welverton Rodrigues, por toda a atenção e dedicação, por todos os comentários e recomendações, e por ter destinado muito de seu tempo a esta orientação, mesmo estando a cursar o mestrado e o doutorado. Obrigado pela orientação e por terem sido capazes de tirar leite de pedra.

A todos os amigos que conheci durante a realização deste curso. Em especial, ao Iuri Cesar, Guilherme Pereira e Saymon Galvão. Sem vocês eu certamente não concluiria esta etapa.

A todos os professores da universidade e do colégio. Obrigado por todos os conhecimentos transmitidos e por serem minha inspiração para continuar aprendendo e seguindo na área da computação.

RESUMO

Neste trabalho de conclusão de curso foi abordada uma generalização do problema *k-median*, chamada *leasing k-median*. Nessa generalização são dados um conjunto de instantes de tempo, um conjunto de clientes que precisam ser atendidos para cada instante de tempo, um conjunto de pontos onde podem ser abertas facilidades para atender os clientes, as possíveis durações para a abertura de uma facilidade e uma constante k. O objetivo do problema é determinar quais, quando e por quanto tempo as facilidades devem ser abertas para minimizar os custos de atender os clientes, respeitando uma restrição de que no máximo k facilidades podem estar abertas ao mesmo tempo. Ao longo do trabalho foram propostos uma formulação de programação linear inteira e dois algoritmos heurísticos baseados nas meta-heurísticas BRKGA e VNS. A formulação de programação linear foi utilizada junto a um resolvedor comercial como um *benchmark* para os experimentos computacionais. Os experimentos consistiram em executar cada uma das abordagens por 10 minutos e considerar a melhor solução obtida nesse prazo para um conjunto de instâncias pseudoaleatórias. O resolvedor mostrou-se bastante útil na resolução de instâncias pequenas, encontrando, durante os testes, soluções ótimas para todas as instâncias desse tipo em menos de 30 segundos. O BRKGA apresentou em média um *gap* de otimalidade de $7,08\%$ entre um limite inferior do valor ótimo e o valor da solução incumbente encontrada pela heurística. O VNS mostrou-se, dentre as três abordagens, como a melhor opção para as instâncias de médio e de grande portes obtendo em média *gaps* de otimalidade, respectivamente, de $2,08\%$ e $5,70\%$ entre o valor da solução incumbente e um limite inferior do valor ótimo.

Palavras-chave: Problema *leasing k-median*. Localização de facilidades. Heurísticas.

ABSTRACT

In this term paper a generalization of the k-median problem, known as leasing k-median problem, was studied. For this generalization are given a set of time instants, sets of customers that need to be served at each time instant, a set of points where facilities can be opened to serve customers, the possible durations for a facility opening and a constant value k. The objective of the problem is to determine which, when and for how long the facilities should be opened to minimize the costs of serving the customers, while respecting a rescriction that at most k facilities may be open at the same time. Throughout the work, an integer linear programming formulation and two heuristics based on the BRKGA and VNS metaheuristics were proposed. The linear programming formulation was used with a commercial solver as a benchmark for the computational experiments. The experiments were performed by running each approach for 10 minutes and considering the best solution obtained within that timeframe for a set of random instances. The solver was very useful in resolving small instances, finding, during the test phase, an optimal solution for all instances of this type in less than 30 seconds. The BRKGA presented an average optimality gap of 7.08% between a lower bound of the optimal value and the value of the incumbent solution found by the heuristic. Among the three approaches, the VNS was the best option for medium and large-sized instances, obtaining, on average, optimal gaps, respectively, of 2.08% and 5.70% between the value of the incumbent solution and a lower bound of the optimal value.

Keywords: Leasing k-median problem. Facility location. Heuristics.

LISTA DE FIGURAS

Figura 1 – Instância de exemplo 12
Figura 2 – Solução para a instância de exemplo: meses 1-3 12
Figura 3 – Solução para a instância de exemplo: meses 4-6 13
Figura 4 – Solução para a instância de exemplo: meses 7-9 13
Figura 5 – Processo de decodificação 18
Figura 6 – Esquematização do VNS 20
Figura 7 – Exemplo de solução para instância com $t_{max} = 4$ e $k = 3$ 29
Figura 8 – Exemplo de solução para instância com $t_{max} = 5, k = 3$ e durações de 2 e 3 instantes 30
Figura 9 – Solução após execução do movimento m_2 sem tratamento 30
Figura 10 – Solução após execução do movimento m_2 com tratamento 31
Figura 11 – Instância de exemplo com 5 instantes de tempo 32
Figura 12 – Soluções do *k-median* para a instância de exemplo (Etapa 1) 33
Figura 13 – Solução do *leasing k-median* para a instância de exemplo (Etapa 2) . . 36
Figura 14 – Gráfico de desempenho para o Grupo I 41
Figura 15 – Gráfico de desempenho para o Grupo II 43
Figura 16 – Gráfico de desempenho para o Grupo III 44

LISTA DE TABELAS

Tabela 1 – Descrição das instâncias do Grupo I 39
Tabela 2 – Descrição das instâncias do Grupo II 39
Tabela 3 – Descrição das instâncias do Grupo III 39
Tabela 4 – Valores das soluções para o Grupo I 42
Tabela 5 – Valores das soluções para o Grupo II 42
Tabela 6 – Valores das soluções para o Grupo III 44

LISTA DE ALGORITMOS

Algoritmo 1 – BRKGA 19
Algoritmo 2 – Busca Local 20
Algoritmo 3 – VNS 21

Algoritmo 4 – Decodificador 28
Algoritmo 5 – Busca Local de Arya et al. (2003) 32
Algoritmo 6 – Converte Solução: converte(k', t) 34
Algoritmo 7 – Salva Solução: salvaSolucao(k', t) 35
Algoritmo 8 – VNS 37

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivos 14
1.2 Organização textual 14

2 CONCEITOS PRELIMINARES 15
2.1 Espaço Métrico 15
2.2 Problemas de Otimização Combinatória 15
2.3 Programação Linear Inteira 16
2.4 Abordagens Heurísticas 17
2.4.1 *Biased Random-Key Genetic Algorithm* 18
2.4.2 Busca Local 19
2.4.3 *Variable Neighborhood Search* 20

3 O PROBLEMA *LEASING k-MEDIAN* 22
3.1 Definição Formal 22
3.2 Formulação 22

4 TRABALHOS RELACIONADOS 24

5 ABORDAGENS UTILIZADAS 26
5.1 Resolvedor de Programação Linear Inteira 26
5.2 BRKGA 26
5.2.1 Codificação da Solução 27
5.2.2 Decodificador 27
5.3 VNS 28
5.3.1 Representação das Soluções 28
5.3.2 Estruturas de Vizinhança 29
5.3.3 Solução Inicial 31
5.3.3.1 Etapa 1 31
5.3.3.2 Etapa 2 33
5.3.4 Pseudocódigo do Algoritmo 36

6 EXPERIMENTOS COMPUTACIONAIS 38
6.1 Elaboração das Instâncias 38
6.2 Definição dos Parâmetros 40
6.3 Resultados 40
6.3.1 Grupo I 40

6.3.2 Grupo II 41
6.3.3 Grupo III 43

7 CONCLUSÃO 45

REFERÊNCIAS 46

1 INTRODUÇÃO

Definir pontos estratégicos para instalar facilidades é uma decisão de projeto comum a várias empresas. Essa etapa de planejamento exige atenção, pois, manter um conjunto de facilidades mal posicionadas pode gerar prejuízos e até mesmo inviabilizar o projeto. Normalmente, o processo de instalação demanda tempo e gastos não desprezíveis.

O problema de localização de facilidades é enunciado sobre um conjunto de pontos onde facilidades podem ser instaladas e sobre um conjunto de pontos onde situam-se os clientes que precisam ser servidos pelas facilidades. O objetivo desse problema é escolher as localizações (i.e., os pontos) para abrir as facilidades de maneira a facilitar o acesso dos clientes aos recursos fornecidos por elas.

Uma série de variações do problema de localização de facilidades foi proposta para se adequar a diferentes cenários. Uma das variações, em especial, é o problema *k-median*. No *k-median* é dado um limitante k para a quantidade de facilidades que podem ser abertas e o objetivo consiste em selecionar até k pontos para abrir facilidades de modo a minimizar o somatório dos custos de servir os clientes, em que cada cliente é servido pela facilidade mais próxima e o custo do serviço está relacionado à distância do cliente à facilidade. Neste trabalho é abordada uma generalização do problema *k-median*.

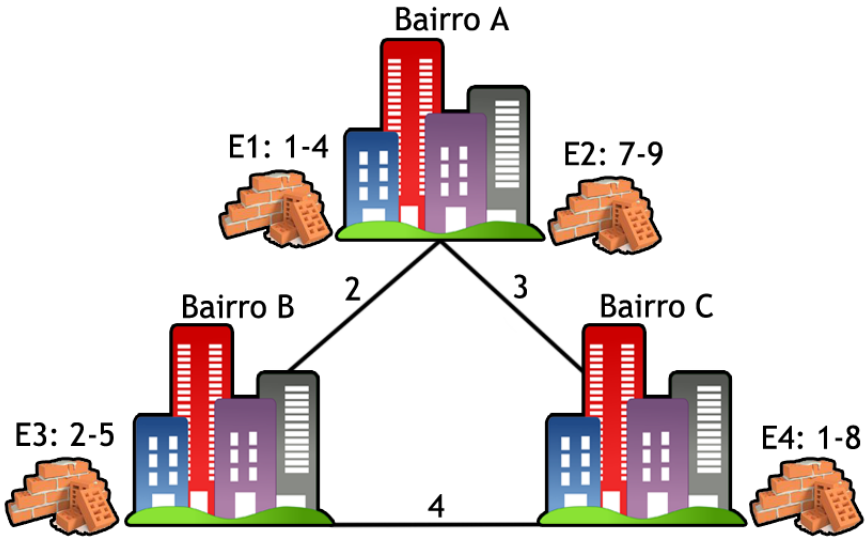
Para ilustrar o propósito da generalização abordada, considere o problema a seguir de uma empresa construtora de imóveis que trabalha com vários empreendimentos concorrentemente. A construção de imóveis, por vezes, é demorada e pode ser realizada em localidades distintas (avenidas, bairros, cidades, etc.). Para manter as máquinas, equipamentos e materiais de construção mais próximos de um empreendimento é preciso mantê-los em depósitos temporários que são alugados em pontos estratégicos. As políticas de arrendamento de depósitos podem estabelecer prazos para a duração de um contrato (podendo ser mensal, semestral, anual, etc.).

Por questões de orçamento, a construtora pode manter no máximo k depósitos alugados ao mesmo tempo. Uma vez que o cronograma dos empreendimentos está estabelecido, surge o problema de otimização que consiste em determinar quais, quando e por quanto tempo os depósitos devem ser alugados para que a soma total das distâncias de cada empreendimento ao depósito mais próximo seja mínima, respeitando a restrição de que no máximo k depósitos estejam alugados ao mesmo tempo.

Considere como um exemplo a instância ilustrada na Figura 1. Nessa instância, a construtora possui empreendimentos agendados nos bairros A, B, e C e deve construí-los em algum período entre os meses 1 e 9. A distância em quilômetros (km) entre cada bairro é expressa pelo valor da aresta que os conecta.

No bairro A, a construtora realizará o empreendimento $E1$ durante os meses de 1 a 4 e realizará o empreendimento $E2$ durante os meses de 7 a 9, denotados, respectivamente,

Figura 1 – Instância de exemplo

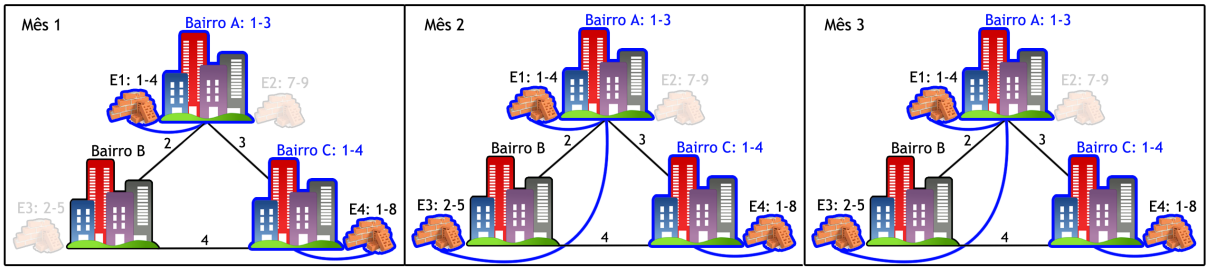


Fonte: elaborada pelos autores.

como $E1$: 1-4 e $E2$: 7-9. De forma análoga, no bairro B, o empreendimento $E3$ ocorrerá nos meses de 2 a 5 e no bairro C, o empreendimento $E4$ será realizado entre os meses 1 e 8.

Para a construção de uma solução, considere que a construtora poderá manter no máximo 2 depósitos abertos por mês e que a duração de um contrato de arrendamento poderá ser de 3 ou 4 meses. As Figuras 2-4 ilustram uma solução para a instância em questão. Na ilustração, um bairro contornado em azul indica que há um depósito alugado naquele local durante o período especificado após o nome do bairro. Cada aresta azul conectando um empreendimento e um bairro indica que o empreendimento está utilizando o depósito aberto naquele bairro.

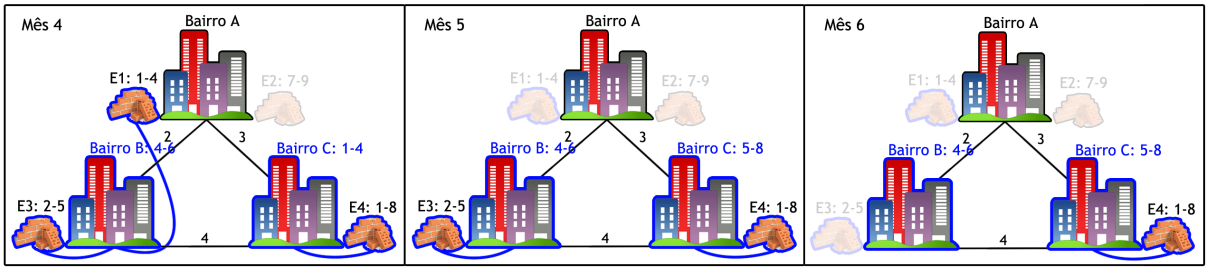
Figura 2 – Solução para a instância de exemplo: meses 1-3



Fonte: elaborada pelos autores.

No início do primeiro mês (Figura 2) dois depósitos são alugados, um no bairro A pelo período de 3 meses para servir o empreendimento $E1$ e outro no bairro B por 4 meses para servir o empreendimento $E4$. No segundo mês, o empreendimento $E3$ é iniciado e é atribuído ao depósito alugado no bairro A, uma vez que este é o depósito ativo mais próximo.

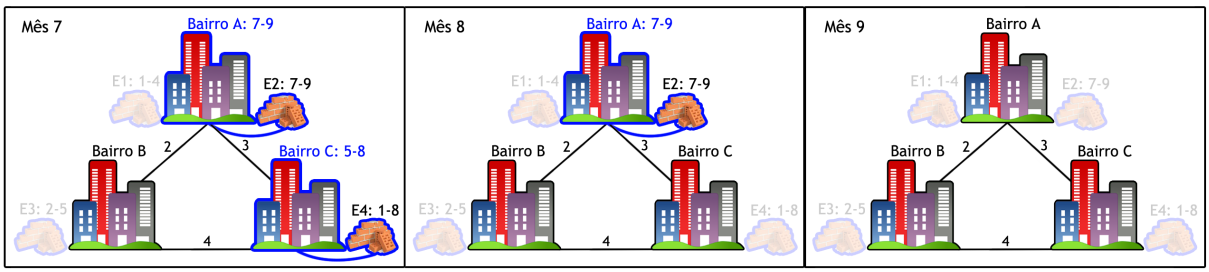
Figura 3 – Solução para a instância de exemplo: meses 4-6



Fonte: elaborada pelos autores.

No quarto mês (Figura 3), o contrato de aluguel em A é finalizado, e um depósito é alugado no bairro B. Como consequência, os empreendimentos $E1$ e $E3$ são realocados para o depósito mais próximo, ou seja, o depósito em B. Nos quinto e sexto meses, os empreendimentos $E1$ e $E3$ são finalizados.

Figura 4 – Solução para a instância de exemplo: meses 7-9



Fonte: elaborada pelos autores.

No sétimo mês (Figura 4), o contrato de aluguel no bairro B é encerrado e um novo depósito é arrendado em A para servir $E2$. Durante os meses 8 e 9, os empreendimentos $E4$ e $E2$ são finalizados, respectivamente.

Apenas no estado de São Paulo, por exemplo, a empresa MRV Engenharia está construindo 38 empreendimentos concorrentemente (MRV Engenharia, 2019). Cenários como esse podem ser beneficiados com soluções para problemas semelhantes ao da construtora, dado que, inserir a resolução de tarefas de otimização na rotina de grandes empresas pode poupar gastos significativos.

O problema motivacional descrito anteriormente é uma aplicação do problema *leasing k-median*, proposto em (LINTZMAYER; MOTA, 2017). Informalmente, no problema *leasing k-median*, os clientes precisam ser atendidos em instantes de tempo específicos e, para isso, as facilidades podem ser abertas por intervalos de tempo definidos previamente. O objetivo do problema é determinar as localidades e os períodos de tempo em que as facilidades devem ser abertas, de modo que o custo total de atender os clientes seja mínimo, respeitando a restrição de que para cada instante de tempo existam no máximo k facilidades abertas.

Além do problema motivacional da construtora, a determinação de localidades para agências imobiliárias, o escalonamento de agências de *telemarketing* e a análise

de intervalos de séries temporais podem ser modelados como instâncias do problema *leasing k-median*.

O problema *leasing k-median* pertence à classe NP-Difícil, dado ser uma generalização do *k-median*, que foi provado ser NP-Difícil por KARIV; HAKIMI (1979). Devido ao fato de provavelmente não ser possível obter uma solução ótima em tempo polinomial para o *leasing k-median*, abre-se espaço para a utilização de abordagens heurísticas para tratar o problema.

1.1 Objetivos

O objetivo deste trabalho de conclusão de curso é apresentar as abordagens desenvolvidas para o problema *leasing k-median*. Ao longo do trabalho foram propostos dois algoritmos heurísticos e uma abordagem exata para o tratamento de instâncias pequenas. Os algoritmos propostos foram comparados com o auxílio de experimentos computacionais.

1.2 Organização textual

O restante deste trabalho está textualmente assim organizado: o Capítulo 2 apresenta os conceitos preliminares sobre espaço métrico, otimização combinatória, programação linear inteira e heurística, também são introduzidas duas abordagens meta-heurísticas; o Capítulo 3 é composto da definição formal e de uma formulação matemática para o problema *leasing k-median*; o Capítulo 4 introduz os principais trabalhos e os trabalhos mais recentes relacionados ao problema abordado; o Capítulo 5 descreve as três abordagens utilizadas para tratar o *leasing k-median*, a saber: programação linear inteira, BRKGA e VNS; o Capítulo 6 apresenta os resultados dos experimentos computacionais realizados; por fim o Capítulo 7 conclui com algumas considerações finais.

2 CONCEITOS PRELIMINARES

Neste capítulo são apresentados os fundamentos teóricos necessários para o entendimento deste trabalho. Ao longo das Seções 2.1 e 2.2 são introduzidos os conceitos de espaço métrico e de otimização combinatória, necessários para a definição formal do problema. Em seguida, na Seção 2.3 a programação linear inteira é apresentada como uma forma exata para tratar problemas de otimização combinatória. Por fim, no restante das seções são expostas duas abordagens heurísticas: o *biased random-key genetic algorithm* e a *variable neighborhood search*.

2.1 Espaço Métrico

Espaço métrico é um conjunto e uma função métrica capaz de determinar a distância entre quaisquer dois elementos pertencentes ao conjunto. Um exemplo de espaço métrico é o espaço euclidiano e a norma euclidiana.

Formalmente, um *espaço métrico* é composto por um conjunto V e uma função $d\colon V \times V \to \mathbb{R}_+$, em que d satisfaz às seguintes propriedades:

1. *distância nula*: $d(i, j) = 0$ se e somente se $i = j$, para todo $i, j \in V$;
2. *simetria*: $d(i, j) = d(j, i)$, para todo $i, j \in V$;
3. *desigualdade triangular*: $d(i, j) \leq d(i, k) + d(k, j)$, para todo $i, j, k \in V$.

A propriedade 1 garante que a distância de um elemento para ele mesmo seja nula e que, por outro lado, a distância seja positiva entre quaisquer elementos distintos. A propriedade 2 garante que ir de i para j não seja mais fácil e nem mais difícil do que ir de j para i. Por fim, a propriedade 3 garante que um dos caminhos mais curtos entre dois elementos seja sempre a rota direta entre os dois.

2.2 Problemas de Otimização Combinatória

Existem problemas que se resumem a encontrar dentro de um conjunto discreto, uma combinação de elementos que gere a melhor solução possível para o problema. Na ciência da computação, o segmento responsável por estudar esses problemas é a *otimização combinatória* (PAPADIMITRIOU; STEIGLITZ, 1998). Segundo CARVALHO et al. (2001), um problema de otimização combinatória Π pode ser descrito por três elementos principais: um conjunto de instâncias, um conjunto de soluções possíveis $sol(I)$ para cada instância I e uma função objetivo $val\colon sol(I) \to \mathbb{Q}$.

Uma solução S para uma instância qualquer de Π é chamada de solução viável, se S atende a todas as restrições do problema. Note que nem toda solução necessariamente é uma solução viável. Considere, por exemplo, um problema bem simples em que o objetivo

seja minimizar uma variável x, tal que x é um número natural. Para esse problema $x = 2$ e $x = -1$ são duas soluções candidatas, entretanto, apenas $x = 2$ é viável. Nestas situações dizemos que a solução $x = -1$ é uma solução não viável.

O conjunto que reune todas as soluções viáveis para uma instância I do problema é o conjunto $sol(I)$, chamado *conjunto de soluções viáveis*. Em um problema de otimização combinatória, esse conjunto deve ser discreto, ou ao menos, deve ser posssível reduzí-lo a um conjunto discreto.

A função objetivo val, mapeia uma solução S para um número racional que representa o valor da solução. Em um problema de otimização combinatória, o interesse está em uma solução viável do problema que possua o melhor valor possível. Portanto, quando Π se tratar de um problema de minimização, uma solução $S^* \in sol(I)$ é chamada *solução ótima* se $val(S^*) \leq val(S)$, para toda solução $S \in sol(I)$. Analogamente, quando Π se tratar de um problema de maximização, S^* é uma solução ótima se $val(S^*) \geq val(S)$.

2.3 Programação Linear Inteira

No mundo real, diversos processos e fenômenos só fazem sentido quando são representados por variáveis inteiras. É o que acontece, por exemplo, na determinação do número de peças a serem produzidas em uma fábrica ou na determinação da quantidade de ônibus que devem ser alocados para algum período do dia. É em situações como essas que a programação linear inteira pode ser empregada.

Um problema de programação linear é um problema de otimização combinatória constituído por uma função objetivo linear e por um conjunto de variáveis de decisão que devem satisfazer a um sistema de equações e inequações lineares. Quando algumas das variáveis de decisão precisam ser inteiras tem-se um problema de programação linear inteira (PAPADIMITRIOU; STEIGLITZ, 1998).

Há uma série de algoritmos já consolidados para resolver problemas de programação linear, tais como, o *Simplex*, o método de pontos interiores (KARMARKAR, 1984) e o método dos elipsóides (KHACHIYAN, 1980), sendo que os dois últimos são de complexidade polinomial. Contudo, quando se trata da programação linear inteira, a situação é um pouco diferente, já que o problema de programação linear inteira em sua forma geral é NP-Completo (GAREY; JOHNSON, 1990).

Embora não sejam de complexidade polinomial, existem alguns algoritmos para tratar problemas de programação linear inteira de forma exata e esses são classificados em algoritmos enumerativos e em algoritmos de plano de corte. A ideia por trás dos algoritmos enumerativos é listar as soluções viáveis de maneira inteligente, sendo que o principal método que emprega essa técnica é o algoritmo *Branch-and-Bound* (GOLDBARG; LUNA, 2005). Os algoritmos de plano de corte introduzem novas restrições ao problema sucessivamente, "cortando" o conjunto de possíveis soluções para eliminar soluçoes candidatas, sem contudo

descartar qualquer solução inteira (FÜGENSCHUH; MARTIN, 2005). Um exemplo de algoritmo que emprega esta técnica é a decomposição de Benders (BELFIORE; COSTA; FÁVERO, 2005).

Outra alternativa é tratar problemas de programação linear inteira de forma não exata. Uma técnica para atingir esse fim, ou para, pelo menos, realizar uma análise, é a relaxação, em que as restrições de integralidade do problema são removidas, transformando-o em um problema de programação linear convencional (CONFORTI; CORNUÉJOLS; ZAMBELLI, 2014).

2.4 Abordagens Heurísticas

Muitos dos problemas de otimização combinatória ainda não possuem algoritmos capazes de fornecer soluções ótimas para instâncias práticas em tempo hábil. Contudo, em alguns casos, o interesse pode não estar em uma solução ótima do problema, mas sim em uma solução que seja "boa", i.e. que seja relativamente próxima do ótimo, e que possa ser computada em um prazo de tempo aceitável. É nesse contexto em que são empregadas as heurísticas.

Informalmente, uma *heurística* é um algoritmo que não apresenta garantia sobre a qualidade da solução obtida em relação a uma solução ótima. Em termos gerais, uma heurística explorara características a respeito do problema para poder gerar as soluções viáveis rapidamente (FERLAND; COSTA, 2001).

Algumas abordagens heurísticas são genéricas o suficiente para serem adaptadas a problemas diferentes. Esse tipo de heurística é chamado meta-heurística. As meta-heurísticas descrevem estratégias de alto-nível, inspiradas na natureza ou não, para realizar buscas em diferentes regiões do espaço de soluções. Uma meta-heurística é aplicada a um problema de otimização, geralmente, implementando apenas alguns trechos da estratégia para inserir as características particulares do problema (GENDREAU; POTVIN, 2010).

Em alguns casos é possível encontrar limitantes que definem até quão longe do valor ótimo uma solução heurística pode chegar. A esse tipo de heurística, dá-se o nome algoritmo de aproximação. Um *algoritmo de α-aproximação* é um algoritmo polinomial que produz soluções para um problema de otimização combinatória, no qual o valor das soluções produzidas é no máximo um fator de α do valor da solução ótima (WILLIAMSON; SHMOYS, 2011).

A constante α é o *fator de aproximação* do algoritmo. Em um problema de minimização o valor de uma solução aproximada deve ser no máximo $\alpha val(S^*)$, em que $val(S^*)$ é o valor ótimo. Ao passo que, em um problema de maximização o valor da solução aproximada deve ser no mínimo $\frac{val(S^*)}{\alpha}$.

O intuito deste trabalho não é desenvolver algoritmos de aproximação. Entretanto, o termo foi aqui definido pois os trabalhos mais recentes sobre o *k-median* concentram-se

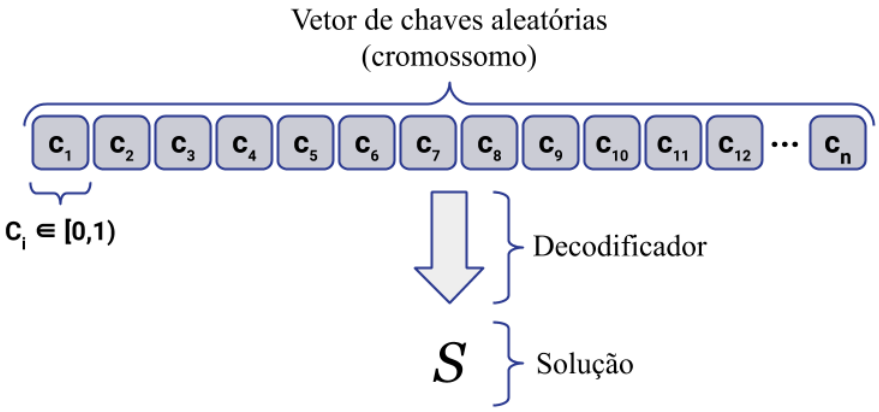
no estudo desses algoritmos.

2.4.1 *Biased Random-Key Genetic Algorithm*

O *Biased Random-Key Genetic Algorithm* (BRKGA) é uma meta-heurística baseada nos algoritmos genéticos proposta por GONÇALVES; RESENDE (2011). A principal característica do BRKGA é ser estritamente elitista, ou seja, além de atribuir maior probabilidade para que as melhores características sejam propagadas ao longo das gerações, os melhores cromossomos são sempre perpetuados sem a incidência de qualquer fator aleatório nessa decisão.

No BRKGA um cromossomo é representado por um vetor com n chaves aleatórias, em que cada *chave aleatória* é um número racional pertencente ao intervalo $[0, 1)$. Para que um cromossomo seja mapeado para uma solução do problema de otimização, utiliza-se um algoritmo chamado decodificador. O decodificador recebe como entrada um vetor de chaves aleatórias e devolve a solução. A Figura 5 ilustra o processo de decodificação.

Figura 5 – Processo de decodificação



Fonte: elaborada pelos autores.

O BRKGA evolui uma população de vetores de chaves aleatórias por várias gerações. Para isso, o algoritmo é inicializado com uma população contendo p vetores de n chaves aleatórias. A cada geração, a população é dividida em um conjunto elite, contendo os $p_e < \frac{p}{2}$ vetores melhor avaliados pelo decodificador, e em um conjunto não elite, contendo o resto da população (RESENDE, 2013).

Todos os vetores do conjunto elite são copiados diretamente para a próxima geração do BRKGA. Em seguida, um pequeno conjunto de $p_m < \frac{p}{2}$ vetores mutantes, gerados de forma pseudoaleatória, são introduzidos na próxima geração para evitar que as soluções convirjam para um ótimo local. Os demais $p - p_e - p_m$ vetores são gerados através do processo de cruzamento.

Um processo de cruzamento mescla dois cromossomos de uma população para gerar uma nova solução. No BRKGA, o cruzamento toma como pais dois vetores a e b, sendo o pai a proveniente do conjunto elite e o pai b do conjunto não elite, e gera um

vetor descendente c. A i-ésima chave do vetor c, recebe a i-ésima chave do vetor pai a com probabilidade p_a e recebe a i-ésima chave do vetor pai b com probabilidade $1 - p_a$, sendo que $p_a > 1/2$. O Algoritmo 1 apresenta um pseudocódigo do BRKGA.

Algoritmo 1: BRKGA

Entrada: p, n, p_e, p_m, p_a
Saída: melhor solução encontrada (S^*)
início
  Inicialize a população inicial P com p vetores de chaves aleatórias;
  enquanto *condição de parada não é satisfeita* **faça**
    Decodifique os novos vetores da população P;
    Separe P em um conjunto elite P_e e em um conjunto não elite $P_{\bar{e}}$;
    Inicialize a próxima geração com o conjunto elite: $P^+ \leftarrow P_e$;
    Gere um conjunto P_m com p_m vetores de chaves aleatórias mutantes;
    Introduza os vetores mutantes na próxima geração: $P^+ \leftarrow P^+ \cup P_m$;
    para $i \leftarrow 1$ **até** $p - p_e - p_m$ **faça**
      Selecione o pai a aleatoriamente de P_e;
      Selecione o pai b aleatoriamente de $P_{\bar{e}}$;
      para $j \leftarrow 1$ **até** n **faça**
        Gere um número real aleatório $r \in [0, 1]$;
        se $r \leq p_a$ **então**
          $c[j] \leftarrow a[j]$;
        senão
          $c[j] \leftarrow b[j]$;
        fim
      fim
      Adicione o filho c à próxima geração: $P^+ \leftarrow P^+ \cup \{c\}$;
    fim
    Atualize a geração atual: $P \leftarrow P^+$;
  fim
  Encontre a melhor solução S^* em P;
fim

2.4.2 Busca Local

Uma busca local é uma heurística empregada na tentativa de melhorar uma solução já existente para um determinado problema. De modo geral, uma busca local inicia com uma solução qualquer e nela aplica pequenas mudanças para gerar as novas soluções (MICHALEWICZ; FOGEL, 2004).

Cada tipo de mudança que é aplicada em uma solução é chamada de *movimento*. Suponha que a solução para um problema qualquer possa ser representada por um vetor.

Exemplos de movimento para essa solução podem ser a inserção de um novo elemento no vetor, a remoção de um elemento existente, a troca entre duas posições do vetor, etc.

Uma solução S' obtida a partir da aplicação de um movimento específico na solução S é chamada de *vizinho* de S. O conjunto $\mathcal{N}(S)$ que reune todos os vizinhos de S com respeito a um determinado movimento é denominado *vizinhança* da solução S. O Algoritmo 2 apresenta o pseudocódigo de uma busca local.

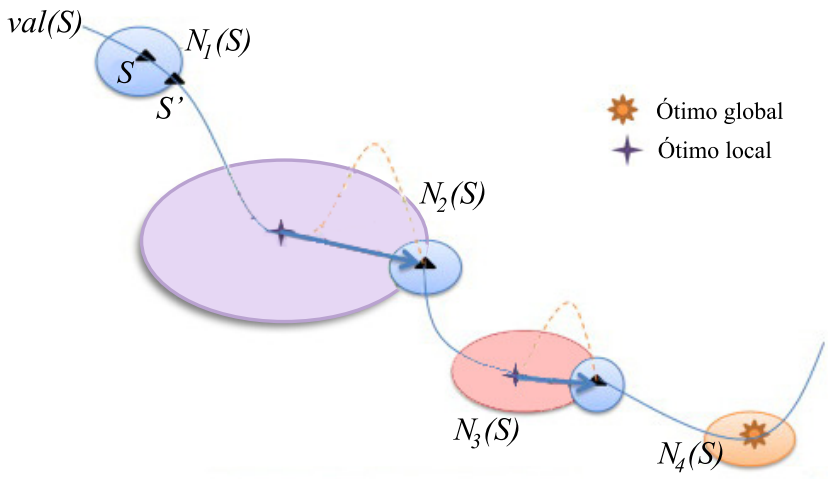
Algoritmo 2: Busca Local

Entrada: solução inicial S^0
Saída: melhor solução encontrada S^*
início
  $S^* \leftarrow S^0$;
  repita
    $S' \leftarrow \arg \min_{S \in \mathcal{N}(S^*)} val(S)$;
    se $val(S') < val(S^*)$ **então**
      $S^* \leftarrow S'$;
    fim
  até *que nenhum vizinho de* S^* *possua valor menor que* $val(S^*)$;
fim

2.4.3 *Variable Neighborhood Search*

O *Variable Neighborhood Search* (VNS), em português, busca em vizinhança variável, é um algoritmo que mescla a busca local com a exploração de diferentes vizinhanças ao longo de sua execução. O VNS foi proposto por MLADENOVIĆ; HANSEN (1997) e se baseia no princípio de que um ótimo global para uma instância de um problema é o ótimo local de todas as vizinhanças possíveis.

Figura 6 – Esquematização do VNS



Fonte: adaptado de CHEN et al. (2013).

O VNS utiliza de um conjunto preestabelecido de R estruturas de vizinhança $\{\mathcal{N}^1, \mathcal{N}^2, \ldots, \mathcal{N}^R\}$, sendo que $\mathcal{N}^r(S)$ é a vizinhança da solução S com respeito ao movimento de índice r. A Figura 6 esquematiza a lógica utilizada nessa meta-heurística. A partir de uma solução inicial S, o algoritmo explora uma determinada estrutura de vizinhança na tentativa de melhorar a solução. Quando um mínimo local é encontrado, o VNS move para outras estruturas de vizinhança, garantindo, assim, a exploração de diferentes regiões do espaço de busca.

Em termos práticos, a cada iteração do algoritmo, uma solução S' de $\mathcal{N}^r(S)$ é escolhida de maneira pseudoaleatória e é submetida a um processo de busca local para refinamento. Caso a solução refinada S' seja melhor do que a solução corrente S, S' substitui S e o processo é reiniciado a partir da primeira estrutura de vizinhança ($r = 1$); caso contrário, move-se para a próxima estrutura de vizinhança $r + 1$ e o processo repete-se (BOUSSAÏD; LEPAGNOT; SIARRY, 2013). O Algoritmo 3 exibe um pseudcódigo para o VNS.

Algoritmo 3: VNS

Entrada: solução inicial S^0, quantidade de estruturas de vizinhança R
Saída: melhor solução encontrada S^*
início
  $S^* \leftarrow S^0$;
  enquanto *condição de parada não é satisfeita* **faça**
    $r \leftarrow 1$;
    enquanto $r \leq R$ **faça**
      Gere uma solução qualquer S' de $\mathcal{N}^r(S^*)$;
      $S'' \leftarrow BuscaLocal(S')$;
      se $val(S'') < val(S^*)$ **então**
        $S^* \leftarrow S''$;
        $r \leftarrow 1$;
      senão
        $r \leftarrow r + 1$;
      fim
    fim
  fim
fim

3 O PROBLEMA *LEASING k-MEDIAN*

O problema *leasing k-median* é um problema de otimização combinatória que possui o objetivo de selecionar dentro de um conjunto de localidades, instantes de tempo e durações, as facilidades que devem ser abertas para que o custo de atender todos os clientes seja mínimo.

3.1 Definição Formal

Considere $T = \{0, \ldots, t_{max}\}$ o conjunto dos instantes de tempo em que os clientes devem ser servidos, em que t_{max} é o instante máximo. Seja V o conjunto que reune todos os clientes e os pontos onde facilidades podem ser abertas. Os clientes que precisam ser servidos no instante t, são agrupados no subconjunto D_t de V, para t em T. Considere, portanto, $\mathcal{D} = \{D_1, \ldots, D_{t_{max}}\}$ uma coleção de clientes.

Considere $L = \{1, \ldots, l_{max}\}$ o conjunto dos tipos de facilidades que podem ser abertas, em que l_{max} é o último tipo considerado. Uma facilidade pode ser aberta por um dentre $|L|$ períodos de tempo $\delta_1, ..., \delta_{l_{\max}}$, de modo que uma facilidade do tipo l aberta no instante t permanece ativa durante o intervalo $[t, t + \delta_l)$, para l em L.

Formalmente, o problema *leasing k-median* é definido da seguinte maneira: dados um espaço métrico (V, d), inteiros positivos $\delta_1, ..., \delta_{l_{\max}}$, uma coleção de clientes $\mathcal{D}$ e um inteiro positivo k, encontrar um conjunto $\mathcal{M} \subseteq V \times L \times T$ tal que, para todo t em T, vale $|\{(i, l, t') \in \mathcal{M} : t \in [t', t' + \delta_l)\}| \leq k$, e minimiza a soma das distâncias de cada cliente à facilidade ativa mais próxima, dada por

$$\sum_{t \in T} \sum_{j \in D_t} \min_{\substack{(i,l,t') \in \mathcal{M} \\ t \in [t', t' + \delta_l)}} d(i, j).$$

3.2 Formulação

O problema *leasing k-median* pode ser formulado como um problema de programação linear inteira a partir de uma adaptação da formulação proposta por AHN et al. (1988).

Na formulação a seguir, a variável x_{ij}^t indica se a facilidade i atende o cliente j no instante t. A variável y_{il}^t indica se a facilidade i do tipo l é aberta no instante t. Com o intuito de simplificar a formulação, considere p_l^t como o menor instante de tempo em que uma facilidade do tipo l deve ser aberta para continuar ativa até o instante t, sendo $p_l^t = t - \delta_l + 1$, se $\delta_l \leq t$; ou $p_l^t = 1$, caso contrário.

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

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

$$\sum_{i \in V} \sum_{l \in L} \sum_{t' = p_l^t}^{t} y_{il}^{t'} \leq k \qquad \forall t \in T \tag{3}$$

$$\sum_{l \in L} \sum_{t' = p_l^t}^{t} y_{il}^{t'} \geq x_{ij}^t \qquad \forall t \in T,\ \forall i \in V,\ \forall j \in D_t \tag{4}$$

$$x_{ij}^t \in \{0,1\} \qquad \forall t \in T,\ \forall i \in V,\ \forall j \in D_t \tag{5}$$

$$y_{il}^t \in \{0,1\} \qquad \forall t \in T,\ \forall i \in V,\ \forall l \in L \tag{6}$$

A função objetivo (1) minimiza a soma das distâncias de cada cliente à facilidade ativa mais próxima, enquanto que as restrições em (2) asseguram para cada instante t que cada cliente j em D_t seja servido por uma única facilidade i. As restrições em (3) garantem para cada instante t que no máximo k facilidades estejam ativas. As restrições em (4) asseguram que um cliente j em D_t possa ser servido pela facilidade i em V, apenas se i estiver ativa no instante t.

4 TRABALHOS RELACIONADOS

O problema *leasing k-median* é relativamente recente na literatura e, até o presente momento, não encontramos artigos sobre o mesmo. Por conta disso, ao longo deste capítulo são abordados trabalhos relacionados ao *k-median*.

Uma das primeiras heurísticas para o *k-median* foi apresentada através do trabalho de KUEHN; HAMBURGER (1976). A heurística sugerida trata-se de um algoritmo guloso que seleciona para ser aberta, a cada iteração, a facilidade que provoca a maior redução no custo da solução. O processo se repete até que k facilidades tenham sido abertas.

Feldman, Lehrer e Ray propuseram um algoritmo reverso à heurísica de Kuehn e Hamburger. Esse algoritmo inicia com todas as $|V|$ possíveis facilidades abertas. A cada iteração, a heurística remove a facilidade para a qual o custo da solução formada com as facilidades restantes é o mínimo. O processo de remoção se repete até que existam apenas k facilidades abertas (FELDMAN; LEHRER; RAY, 1966).

O primeiro algoritmo de aproximação de fator constante para o *k-median* foi proposto em CHARIKAR et al. (1999). O algoritmo baseia-se em resolver a relaxação de um problema de programação linear inteira, simplificar os conjuntos de possíveis facilidades e de clientes e arredondar os resultados. Uma vez que o problema relaxado é resolvido, o algoritmo utiliza os valores fracionários das variáveis de decisão para excluir alguns dos clientes e desconsiderar as facilidades distantes aos clientes que restaram. Uma solução para a nova instância é encontrada ao construir uma floresta com os valores fracionários e ao resolver uma versão do *k-median* em que as distâncias entre as facilidades são proporcionais às distâncias entre os vértices de uma floresta. Para construir a floresta um vértice é adicionado para representar cada uma das facilidades e uma aresta direta é inserida entre uma facilidade i e sua facilidade mais próxima, caso a variável de decisão y_i, semelhante à variável y_{il}^t (6), possua valor $\frac{1}{2}$. A solução da instância modificada é então convertida para uma solução da instância original e resulta em um fator de aproximação de $6 + \frac{2}{3}$.

Um algoritmo de busca local foi proposto em (ARYA et al., 2003) e foi provado possuir um fator de aproximação de $3 + \frac{2}{p}$, quando no máximo p facilidades são substituídas a cada iteração do algoritmo. O algoritmo utilizado por Arya et al. inicia com um conjunto qualquer contendo k facilidades e realiza substituições sucessivas de p facilidades até que o custo da solução obtida não possa ser melhorado. A estratégia de busca local proposta permaneceu com o melhor fator de aproximação por cerca de uma década.

Anos depois, foi provado em (LI; SVENSSON, 2016) que, para obter um algoritmo de α-aproximação para o *k-median*, é suficiente obter um algoritmo com fator de aproximação α que gere soluções com até $k + O(1)$ facilidades abertas. Esse tipo de algoritmo é chamado algoritmo de pseudoaproximação e pode ser convertido em um novo método

que abre apenas k facilidades. No mesmo trabalho os autores apresentam um algoritmo de pseudoaproximação com fator $1 + \sqrt{3} + \epsilon$, para algum $\epsilon > 0$. Posteriormente, melhorias introduzidas por BYRKA et al. (2017), no algoritmo em questão, culminaram em um fator de aproximação de $2.675 + \epsilon$, para algum $\epsilon > 0$, o melhor fator até a presente data.

5 ABORDAGENS UTILIZADAS

Neste capítulo são descritas as abordagens utilizadas para tratar o problema *leasing k-median*. Ao longo do texto são apresentados pseudocódigos para os trechos mais importantes dos algoritmos.

5.1 Resolvedor de Programação Linear Inteira

Para abordar o problema *leasing k-median* de forma exata e, assim, prover um meio de *benchmark* para realizar comparações com as abordagens heurísticas propostas, a formulação matemática da Seção 2.3 foi utilizada junto a um resolvedor comercial de programação linear inteira, chamado Gurobi.

Resolvedores de programação linear inteira são *frameworks* destinados a mesclar as abordagens tradicionais aplicadas na programação linear com os resultados mais recentes da literatura. Alguns dos principais resolvedores são: o Gurobi (Gurobi Optimization, 2019), o GNU Linear Programming Kit (MAKHORIN, 2012) e o CPLEX Optimizer (IBM, 2019). Neste trabalho optou-se por utilizar o *framework* Gurobi na versão 8.1.0, um dos resolvedores melhor avaliados em testes de performance (MEINDL; TEMPL, 2012; SCHMIDT; OLIVEIRA; SILVA, 2017; SOUZA et al., 2017).

O método utilizado pelo Gurobi inicia reduzindo as dimensões da modelagem do problema ao excluir as variáveis e as restrições que sejam redundantes ou triviais. Em seguida, um método de busca enumerativo baseado no algoritmo *Branch-and-Bound* é executado, combinando características dos métodos de plano de corte e aplicando heurísticas às soluções visitadas a fim de acelerar a busca. O Gurobi também utiliza de outras estratégias como o paralelismo e a detecção de simetria (Gurobi Optimization, 2019).

Ao longo do processo de busca das soluções, o Gurobi mantém um *log* com a melhor solução viável encontrada até o momento, chamada solução incumbente. Essa solução também é utilizada neste trabalho para possibilitar a realização de comparações com as soluções geradas pelas abordagens heurísticas.

5.2 BRKGA

A primeira abordagem heurística proposta para o problema *leasing k-median* foi o BRKGA. A estrutura geral do algoritmo segue o Algoritmo 1, apresentado no Capítulo 2. Portanto, nesta seção são descritos apenas a codificação da solução e o algoritmo decodificador.

5.2.1 Codificação da Solução

Para o problema *leasing k-median*, um cromossomo é codificado como uma matriz de chaves aleatórias $A = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0(|V|-1)} \\ a_{10} & a_{11} & \dots & a_{1(|V|-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{t_{max}0} & a_{t_{max}1} & \dots & a_{t_{max}(|V|-1)} \end{bmatrix}$ de dimensões $|T| \times |V|$, em que cada elemento a_{ti} corresponde à possibilidade de abrir a facilidade i no instante de tempo t.

5.2.2 Decodificador

Um decodificador para o problema recebe como entrada uma matriz de chaves aleatórias A e devolve uma solução S com os tipos de facilidades que devem ser abertos. O pseudocódigo do decodificador proposto é descrito pelo Algoritmo 4.

Para uma instância qualquer do problema, uma aproximação para a duração média de uma facilidade pode ser dada por $\delta_\mu = \frac{1}{|L|} \sum_{l \in L} \delta_l$. Considerando que cerca de k facilidades estejam ativas a cada instante de tempo, uma aproximação para a quantidade total de facilidades abertas em uma instância pode ser calculada por $q = \frac{|T|k}{\delta_\mu}$.

Do total de facilidades representadas por um cromossomo, deseja-se que apenas cerca de q facilidades sejam abertas. Entre outras palavras, seja v a probabilidade de abrir cada facilidade representada pelo cromossomo, em que $v = \frac{q}{|T||V|} = \frac{k}{\delta_\mu |V|}$, a i-ésima facilidade do tipo $l = 1 + \lfloor \frac{a_{ti}|L|}{v} \rfloor$ poderá ser aberta no instante t se o número aleatório a_{ti} for menor que v.

Durante a decodificação de uma solução, caso $a_{ti} < v$ e a abertura da i-ésima facilidade no instante t implique em ultrapassar o limite de k facilidades ativas, a abertura é ignorada.

Caso nenhuma facilidade esteja aberta no instante t e o conjunto D_t seja não vazio, a facilidade $i_{tmin} = \arg\min_{i \in V} \sum_{j \in D_t} d(i,j)$ será alocada em t por $\delta_{min} = \min_{l \in L} \delta_l$ instantes de tempo. O intuito aqui é possibilitar que a solução se torne viável introduzindo uma alteração que gere o mínimo possível de mudanças no cromossomo.

Algoritmo 4: Decodificador

Entrada: matriz de chaves aleatórias A

Saída: solução decodificada S

início

Inicialize um contador de facilidades ativas em zero para cada instante de tempo: fAtivas $\leftarrow \{0, \ldots, 0\}_{|T|}$;

$\delta_{min} \leftarrow \min_{l \in L} \delta_l$;

para $t \leftarrow 0$ **até** t_{max} **faça**

para $i \in V$ **faça**

se $a_{ti} < v$ **e** $fAtivas[t] < k$ **então**

$l \leftarrow 1 + \lfloor \frac{a_{ti}|L|}{v} \rfloor$;

Adicione (i, t, δ_l) em S;

fAtivas$[t, \ldots, t + \delta_l - 1] \leftarrow$ fAtivas$[t, \ldots, t + \delta_l - 1] + 1$;

fim

fim

// Verifica se nenhuma facilidade está aberta no instante t e se algum cliente precisa ser servido nesse instante

se fAtivas$[t] = 0$ **e** $|D_t| > 0$ **então**

$i_{tmin} \leftarrow \arg\min_{i \in V} \sum_{j \in D_t} d(i, j)$;

Adicione $(i_{tmin}, t, \delta_{min})$ em S;

fAtivas$[t, \ldots, t + \delta_{min} - 1] \leftarrow$ fAtivas$[t, \ldots, t + \delta_{min} - 1] + 1$;

fim

fim

fim

5.3 VNS

A segunda abordagem heurística proposta para o *leasing k-median* foi o VNS. O algoritmo implementado foi estruturado conforme o Algoritmo 3, introduzido na Seção 2.4.3. Ao longo desta seção são descritos uma representação para soluções do *leasing k-median*, as estruturas de vizinhança empregadas e o método utilizado para gerar a solução inicial.

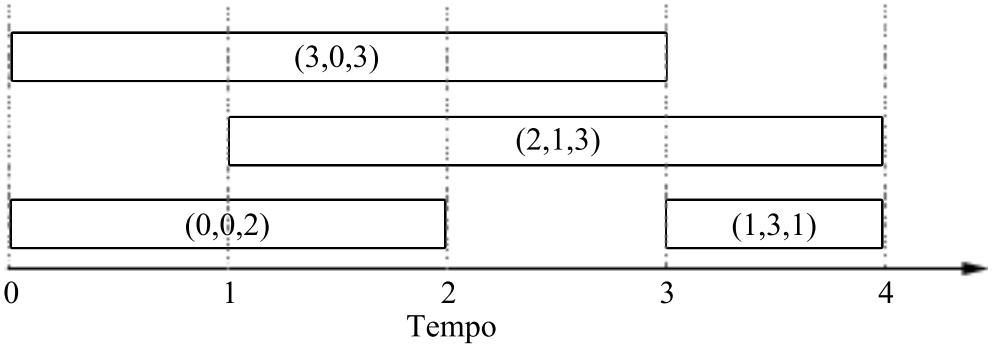
5.3.1 Representação das Soluções

Uma solução para o problema *leasing k-median* pode ser representada por um conjunto C composto por triplas (i, t, δ), em que cada tripla indica que a facilidade i deve ser aberta no instante de tempo t com uma duração de δ instantes, para $i \in V, t \in T$ e $\delta \in \{\delta_1, \ldots, \delta_{l_{max}}\}$. Note que para representar uma solução, não é necessário incluir a atribuição dos clientes às facilidades que os servem, uma vez que essa alocação é feita de forma trivial, designando cada cliente à facilidade ativa mais próxima.

Para facilitar a realização de movimentos simples, o conjunto de triplas C foi dividido em k vetores de triplas $v_1, \ldots, v_k$ com tamanhos não necessariamente iguais, de modo que os intervalos de duração de quaisquer duas facilidades representadas por triplas distintas de um mesmo vetor não se intersectem. Em outras palavras, a divisão das triplas é feita de modo que cada vetor, isoladamente, represente uma solução parcial para uma instância do problema em que $k = 1$.

Considere, por exemplo, a solução de uma instância com $t_{max} = 4$ e $k = 3$ ilustrada pela Figura 7. Cada retângulo na ilustração indica o intervalo de duração da facilidade i aberta no instante t com duração δ, denotado pela tripla (i, t, δ) no interior do retângulo.

Figura 7 – Exemplo de solução para instância com $t_{max} = 4$ e $k = 3$



Fonte: elaborada pelos autores.

Uma forma válida de representar a solução da Figura 7 consiste em utilizar os três seguintes vetores: $v_1 = \{(0, 0, 2), (1, 3, 1)\}, v_2 = \{(2, 1, 3)\}$ e $v_3 = \{(3, 0, 3)\}$. Note que caso as triplas $(3, 0, 3)$ e $(0, 0, 2)$ fossem colocadas no mesmo vetor, haveria intersecção entre os intervalos de duração das facilidades 3 e 0, e portanto, a representação não seria válida.

5.3.2 Estruturas de Vizinhança

Nesta seção dois tipos de movimentos são propostos para possibilitar que, praticamente, qualquer solução S possa ser obtida através de um número finito de movimentos aplicados em uma solução inicial.

O primeiro tipo de movimento m_1 consiste em trocar uma facilidade i de qualquer vetor de triplas por outra facilidade i', tal que $i' \in V$ e $i' \neq i$.

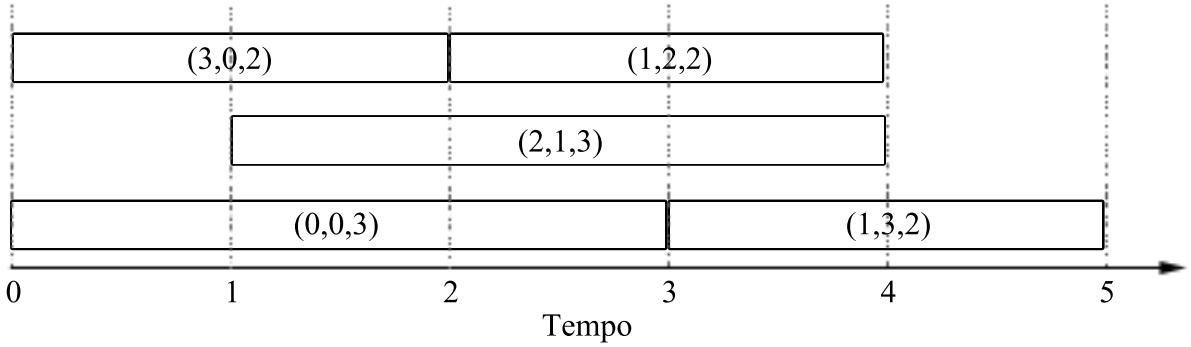
O segundo tipo de movimento m_2 seleciona r vetores de triplas e altera a duração δ de qualquer tripla do q-ésimo vetor selecionado para outra duração δ_q, tal que $\delta_q \in \{\delta_1, \ldots, \delta_{l_{max}}\}$ e $\delta_q \neq \delta$. Perceba que ao alterar a duração de uma facilidade, a solução pode acabar tornando-se inviável e os intervalos de duração de duas facilidades em um mesmo vetor de triplas podem acabar se intersectando. Para contornar esses problemas, dois casos são tratados.

Primeiramente, caso a nova duração seja maior do que a duração anterior, o instante de abertura t das triplas do vetor alterado é adiado o mínimo possível para evitar a sobreposição de intervalos de durações. Caso a nova duração seja menor do que a anterior,

uma nova tripla é adicionada com o instante de abertura imediatamente após o instante em que o intervalo de duração da facilidade alterada é encerrado e, em seguida, caso existam sobreposições os instantes de abertura das triplas do vetor são adiados conforme o caso anterior.

Suponha, por exemplo, que na solução da Figura 8 os vetores de triplas sejam organizados da seguinte forma: $v_1 = \{(3,0,2),(1,2,2)\}$, $v_2 = \{(2,1,3)\}$ e $v_3 = \{(0,0,3),(1,3,2)\}$. Considere também que em uma execução do movimento m_2 a duração da tripla $(3,0,2)$ do vetor v_1 tenha aumentado para 3 e a duração da tripla $(0,0,3)$ do vetor v_3 tenha diminuído para 2.

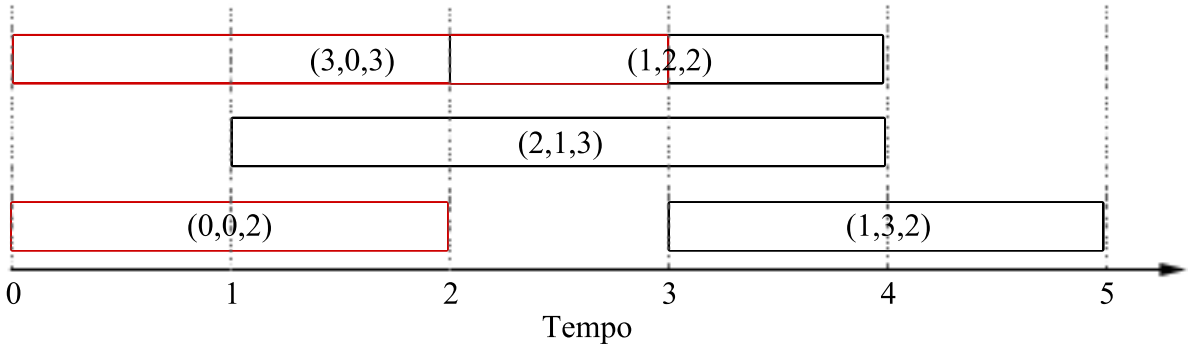
Figura 8 – Exemplo de solução para instância com $t_{max} = 5, k = 3$ e durações de 2 e 3 instantes



Fonte: elaborada pelos autores.

A Figura 9 retrata a solução após a execução do movimento m_2 sem tratar os possíveis conflitos de representação gerados. As triplas alteradas pelo movimento estão representadas em vermelho. A Figura 10 exibe a solução após a realização das alterações para evitar que a solução se torne inviável ou não obedeça às restrições da representação adotada.

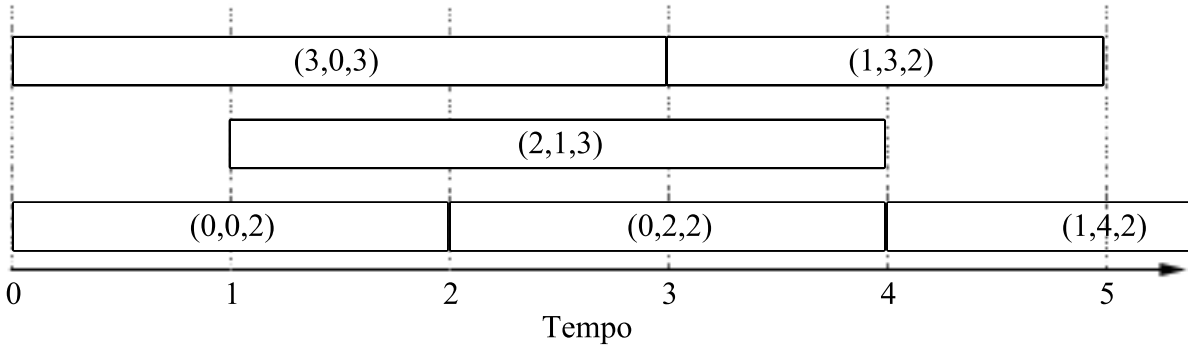
Figura 9 – Solução após execução do movimento m_2 sem tratamento



Fonte: elaborada pelos autores.

Na primeira etapa do movimento, a tripla $(3,0,2)$ é alterada para $(3,0,3)$, ocasionando uma sobreposição de duração com a tripla $(1,2,2)$ (Figura 9). O conflito é corrigido ao adiar a abertura da tripla $(1,2,2)$ em um instante de tempo (Figura 10). Na segunda etapa, a tripla $(0,0,3)$ é alterada para $(0,0,2)$, deixando uma lacuna de 1 instante de tempo. A lacuna é inicialmente preenchida ao adicionar uma nova tripla com a duração

Figura 10 – Solução após execução do movimento m_2 com tratamento



Fonte: elaborada pelos autores.

que mais se aproxime do tamanho da lacuna, o que resulta na adição da tripla $(0, 2, 2)$ no vetor v_3. Após a adição da nova tripla, a sobreposição de durações é corrigida ao adiar a abertura da tripla $(1, 3, 2)$ em 1 instante.

5.3.3 Solução Inicial

É comum que em instâncias práticas do problema *leasing k-median*, uma das possíveis durações de abertura de facilidades a ser considerada seja de 1 instante de tempo. É o que acontece, por exemplo, no arrendamento de instalações por 1 dia, 1 mês, 1 ano, etc. Para instâncias desse tipo, uma solução ótima pode ser encontrada utilizando apenas facilidades que durem 1 instante. Essa afirmação pode ser constatada ao notar que qualquer facilidade com duração δ^*, em que $\delta^* \neq 1$, pode ser convertida em δ^* facilidades com duração de 1 instante de tempo.

Outra observação a respeito desse tipo de instância, é que ao utilizar apenas durações de 1 instante, resolver uma instância do problema *leasing k-median* equivale a resolver $|T|$ instâncias do problema *k-median*, em que a t-ésima instância possui o conjunto D_t como o conjunto de clientes, onde $t \in T$. Além do mais, tratar as instâncias mais genéricas do *leasing k-median* com algoritmos já existentes para o *k-median*, pode ser uma boa pista para encontrar facilidades que ajudem a construir um bom valor de solução.

Valendo-se dessas observações, uma solução inicial para o VNS é construída em duas etapas: resolução de $|T|$ instâncias do *k-median* e adaptação das soluções para o problema *leasing k-median*.

5.3.3.1 Etapa 1

Na primeira etapa da construção da solução inicial, a instância do problema *leasing k-median* é convertida em $|T|$ instâncias do problema *k-median*, em que a t-ésima instância possui o conjunto V como o conjunto de pontos para instalar facilidades, o conjunto D_t como o conjunto de clientes e k como o limite de facilidades a serem abertas.

Para resolver cada uma das $|T|$ instâncias, optou-se por utilizar o algoritmo de

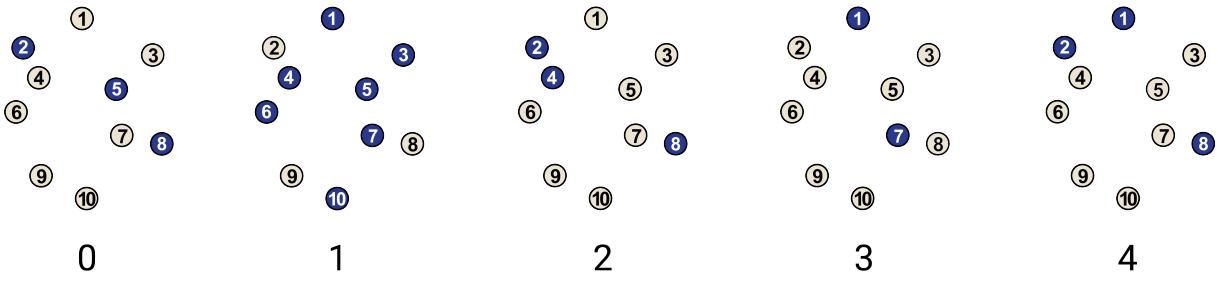
busca local proposto por ARYA et al. (2003), introduzido no Capítulo 4, uma vez que se trata de um algoritmo simples, responsável por um dos melhores fatores de aproximação disponíveis na literatura. A cada iteração, o algoritmo considera como movimento a substituição de uma única facilidade. O pseudocódigo do algoritmo é descrito pelo Algoritmo 5.

Algoritmo 5: Busca Local de Arya et al. (2003)

Entrada: V, D_t, k
Saída: melhor solução encontrada S^*
início
  Inicialize S' com um conjunto qualquer de k facilidades distintas;
  repita
    $S^* \leftarrow S'$;
    para $i^* \in S^*$ **faça**
      para $i' \in V - S^*$ **faça**
        se $val(S^* - \{i^*\} \cup \{i'\}) < val(S')$ **então**
          $S' \leftarrow S^* - \{i^*\} \cup \{i'\}$;
        fim
      fim
    fim
  até *que* $S^* = S'$;
fim

Considere como um exemplo a instância do problema *leasing k-median* ilustrada pela Figura 11. Essa instância é composta por 5 instantes de tempo, de 0 a 4, e por 10 pontos possíveis para abertura de facilidades, cada um identificado por um número de 1 a 10. Os pontos com clientes que precisam ser servidos são destacados em azul.

Figura 11 – Instância de exemplo com 5 instantes de tempo

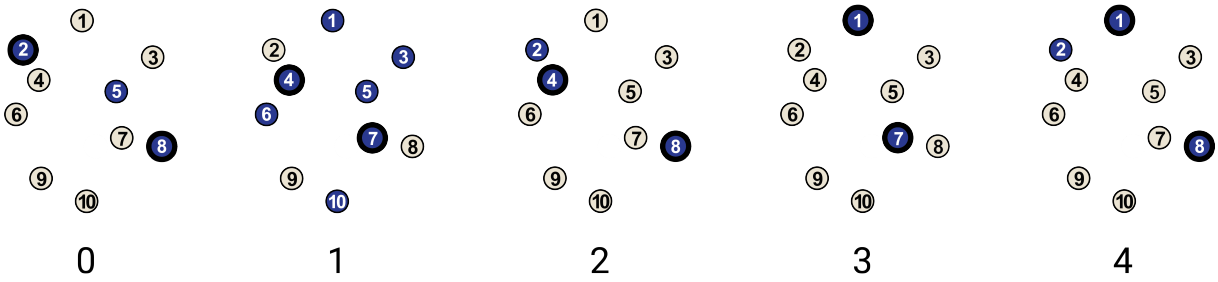


Fonte: elaborada pelos autores.

No instante de tempo 0 os clientes 2,5 e 8 precisam ser servidos. Analogamente, os clientes 1,3,4,5,6,7 e 10 precisam ser atendidos no instante de tempo 1, e assim por diante. Para a construção de uma solução, considere que, no máximo, duas facilidades possam estar ativas ao mesmo tempo, i.e. $k = 2$, e que as durações possíveis para uma abertura sejam de 2 e 3 instantes. Suponha também que os pontos estejam no $\mathbb{R}^2$ e que a métrica de distância utilizada seja a euclidiana.

Ao longo da Etapa 1 o conjunto de pontos de cada instante de tempo é tratado separademente utilizando o Algoritmo 5. As facilidades escolhidas para serem abertas em cada instante de tempo são expostas na Figura 12. Na ilustração uma facilidade aberta é representada por um ponto contornado em negrito.

Figura 12 – Soluções do *k-median* para a instância de exemplo (Etapa 1)



Fonte: elaborada pelos autores.

No instante de tempo 0 os pontos 2 e 3 foram selecionados para abrir facilidades. De modo semelhante, no instante 1 foram abertas facilidades nos pontos 4 e 7. Nos instantes 2, 3 e 4, as facilidades escolhidas foram, respectivamente, 4 e 8, 1 e 7, 1 e 8.

5.3.3.2 Etapa 2

Uma vez que as $|T|$ instâncias do *k-median* estão resolvidas, basta convertê-las em uma solução para o problema *leasing k-median*. A solução é convertida ao tentar manter aberto o maior número possível de facilidades que foram selecionadas na Etapa 1 em seus respectivos instantes de tempo, utilizando, desta vez, as durações da instância original.

Para realizar a conversão foi proposto, neste trabalho, um algoritmo de memorização. A memorização é uma técnica de *design* de algoritmos utilizada para evitar que subproblemas de um problema maior sejam calculados mais do que uma vez (KLEINBERG; TARDOS, 2005).

O algoritmo proposto utiliza três tipos de operação para auxiliar na decisão de quais facilidades e durações devem ser utilizados para maximizar o número de facilidades da Etapa 1 mantidas abertas. As operações são:

- ConsultaQuantidade(i, t, δ): devolve a quantidade de vezes que a facilidade i foi aberta na Etapa 1 durante o intervalo $[t, t')$, em que $t' = \min(t + \delta, |T|)$;
- ExcluiDaContagem(i, t, δ): exclui a facilidade i da contagem durante o intervalo $[t, t')$, em que $t' = \min(t + \delta, |T|)$. Exemplo: suponha que a facilidade i foi aberta nos instantes 0, 2 e 3 na Etapa 1. Uma operação ConsultaQuantidade(i, 0, 4) deve devolver o valor 3, i.e., a quantidade de vezes que i foi aberta durante o intervalo. Após uma execução da operação ExcluiDaContagem(i, 0, 3), o resultado da operação ConsultaQuantidade(i, 0, 4) deve ser 1;
- RecuperaContagem(i, t, δ): desfaz as alterações provocadas pela última operação ExcluiDaContagem(i, t, δ) aplicada à facilidade i no intervalo $[t, t')$, em que $t' =$

$\min(t + \delta, |T|)$.

O Algoritmo 6 apresenta o pseudocódigo do algoritmo de conversão. O algoritmo proposto utiliza uma matriz de memorização $memo_{k \times |T|}$ para armazenar o resultado dos subproblemas calculados. A cada chamada recursiva $converte(k', t)$ é decidido qual combinação facilidade-duração deve ser aberta no instante de tempo t para maximizar a quantidade de facilidades da Etapa 1 mantidas abertas, considerando que no máximo k' facilidades possam estar ativas ao mesmo tempo. O problema de maximização é resolvido através da chamada inicial $converte(k, 0)$.

Algoritmo 6: Converte Solução: converte(k', t)

Entrada: limite de facilidades abertas k', instante de abertura atual t
Saída: quantidade de facilidades da Etapa 1 mantidas abertas
início
  se $k' = 0$ **então** // Limite de facilidades abertas nulo
    retorna 0;
  fim
  se $t \geq |T|$ **então**
    retorna converte$(k' - 1,\ 0)$; // Decrementa o limite de facilidades abertas
  fim
  se $memo[k'][t]$ *foi inicializado* **então** // Verifica se o estado atual foi computado
    retorna $memo[k'][t]$;
  fim
  $memo[k'][t] \leftarrow 0$;
  // Testa a abertura de todas as possíveis combinações de facilidades e durações
  para $i \in V$ **faça**
    para $l \in L$ **faça**
      $qt \leftarrow$ `ConsultaQuantidade(`i`,` t`,` δ_l`)`;
      `ExcluiDaContagem(`i`,` t`,` δ_l`)`;
      // Escolhe a opção que maximiza o nº de facilidades mantidas abertas
      $memo[k'][t] = \max(qt + \text{converte}(k', t + \delta_l), memo[k'][t])$;
      `RecuperaContagem(`i`,` t`,` δ_l`)`;
    fim
  fim
  retorna $memo[k'][t]$;
fim

Para implementar as três operações auxiliares utilizadas no algoritmo de conversão pode-se utilizar $|V|$ árvores de segmentos (HALIM; HALIM, 2013; BARCAROLI, 2016), uma para cada possível facilidade, possibilitando realizar qualquer uma das três operações em tempo $\mathcal{O}(log_2|T|)$.

Sendo assim, a complexidade de tempo do Algoritmo 6 é $\mathcal{O}(k*|T|*|V|*|L|*log_2|T|)$.

O fator $k * |T|$ advém da quantidade máxima de estados calculados pelo algoritmo de memorização. O fator $|V| * |L|$ provém da necessidade de testar cada uma das $|V|$ possíveis facilidades com cada uma das $|L|$ possíveis durações para computar cada estado da memorização. O fator $log_2|T|$ advém do uso da estrutura de dados sugerida.

Após a execução do Algoritmo 6, os k vetores de triplas utilizados para gerar a solução inicial podem ser obtidos analisando a matriz de memorização. O pseudocódigo para obter os k vetores segue o Algoritmo 7. A cada chamada $salvaSolucao(k', t)$ é obtida a combinação facilidade-duração que gerou a melhor resposta para o estado (k', t). Note que a chamada à função $converte(k', t + \delta_l)$ na linha 11 é executada em tempo constante, uma vez que o estado $(k', t + \delta_l)$ já foi computado e armazenado na matriz de memorização.

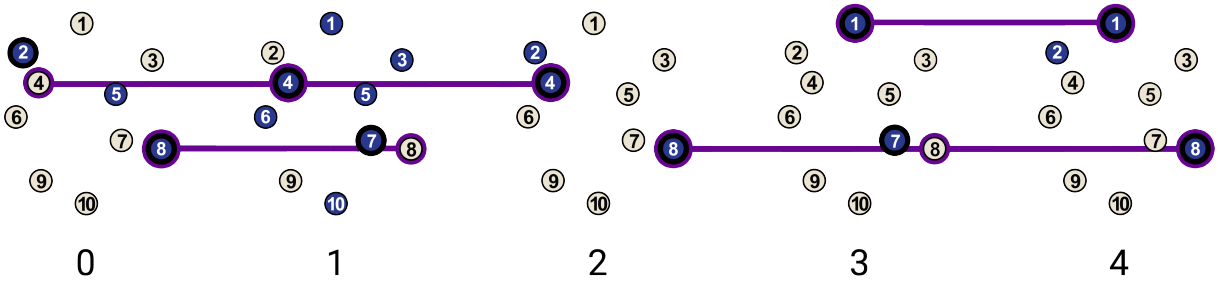
Algoritmo 7: Salva Solução: salvaSolucao(k', t)

```
   Entrada: limite de facilidades abertas k', instante de abertura atual t
 1 início
 2     se k' = 0 então // Limite de facilidades abertas nulo
 3         retorna;
 4     fim
 5     se t ≥ |T| então
 6         salvaSolucao(k' − 1, 0);
 7         retorna;
 8     fim
       // Testa a abertura de todas as possíveis combinações de facilidades e durações
 9     para i ∈ V faça
10         para l ∈ L faça
11             qt ← ConsultaQuantidade(i, t, δ_l);
               // Verifica se a melhor resposta para o estado (k', t) foi obtida utilizando a
                  facilidade i com duração δ_l
12             se memo[k'][t] = qt + converte(k', t + δ_l) então
13                 Adicione a tripla (i, t, δ_l) no vetor v_{k'};
14                 ExcluiDaContagem(i, t, δ_l);
15                 salvaSolucao(k', t + δ_l);
16                 retorna;
17             fim
18         fim
19     fim
20 fim
```

A Figura 13 exibe os resultados da execução da Etapa 2 para a instância de exemplo da Figura 11. As facilidades do *leasing k-median* abertas estão contornadas na cor

roxa. Uma linha roxa unindo um mesmo ponto em diferentes instantes indica o intervalo de duração da facilidade aberta naquele ponto.

Figura 13 – Solução do *leasing k-median* para a instância de exemplo (Etapa 2)



Fonte: elaborada pelos autores.

A facilidade 4 foi aberta no tempo 0 com uma duração de 3 instantes, preservando assim a abertura da facilidade 4 nos instantes 1 e 2 realizada na Etapa 1. A facilidade 8 também foi aberta no instante de tempo 0 e durou até o instante 1, mantendo aberta uma facilidade da Etapa 1, i.e, a facilidade 8 no instante 0.

No instante de tempo 2, a facilidade 8 é novamente aberta, desta vez, com uma duração de 3 instantes, mantendo abertas 2 facilidades da Etapa 1 (nos instantes 2 e 4). Por fim, a facilidade 1 foi alocada no instante 3 e durou 2 instantes, preservando 2 escolhas da Etapa 1.

5.3.4 Pseudocódigo do Algoritmo

Reescrevendo o Algoritmo 3, a abordagem proposta utilizando o VNS é descrita pelo Algoritmo 8.

Algoritmo 8: VNS

Entrada: quantidade de estruturas de vizinhança R

Saída: melhor solução encontrada S^*

início

Gere uma solução inicial S^* utilizando as etapas 1 e 2;

enquanto *condição de parada não é satisfeita* **faça**

$r \leftarrow 1$;

enquanto $r \leq R$ **faça**

Gere uma solução S' executando, de forma pseudoaleatória, o movimento m_2 em S^* com base em r vetores de triplas;

Realize uma busca local em S' com base no movimento m_1: $S'' \leftarrow BuscaLocal(S')$;

se $val(S'') < val(S^*)$ **então**

$S^* \leftarrow S''$;

$r \leftarrow 1$;

senão

$r \leftarrow r + 1$;

fim

fim

fim

fim

6 EXPERIMENTOS COMPUTACIONAIS

Neste capítulo são descritos os experimentos computacionais realizados para comparar as três abordagens utilizadas.

Os algoritmos propostos foram implementados na linguagem C++ com o compilador GCC na versão 5.4.0. Para a formulação de programação linear inteira foi utilizada a API de C++ do Gurobi na versão 8.1.0.

Os experimentos foram executados em um computador com 4GB de memória RAM e com processador Intel Pentium Dual Core de $2,60$ GHz sobre o sistema operacional Ubuntu 16.04.6 LTS (64 *bits*). Foi considerado um tempo limite de 10 minutos para a execução de cada abordagem em cada instância, sendo utilizada, para fins de comparação, a melhor solução obtida dentro desse prazo.

6.1 Elaboração das Instâncias

Para realizar os testes computacionais foram utilizadas 30 instâncias do problema *leasing k-median*. As instâncias foram obtidas de duas formas: gerando novas instâncias e adaptando instâncias do problema *k-median* já existentes.

Cada uma das novas instâncias foi gerada escolhendo de forma pseudoaleatória a quantidade de instantes de tempo $|T|$ no intervalo $[1, 150]$, a quantidade de pontos de facilidades $|V|$ no intervalo $[100, 550]$, o limite de facilidades abertas k no intervalo $[5, 30]$ a quantidade de durações $|L|$ no intervalo $[1, 40]$ e a duração δ_l no intervalo $[1, |T|]$, para cada $l \in L$. O tamanho do conjunto D_t foi selecionado no intervalo $[0, |V|]$ e os clientes desse conjunto foram sorteados a partir do conjunto V, para cada $t \in T$. As distâncias entre as facilidades foram obtidas através da construção de um grafo com arestas ponderadas, em que cada vértice é um dos $|V|$ pontos de facilidade e os pesos são pseudoaleatórios. A distância mínima entre cada par de facilidades foi encontrada aplicando o algoritmo de Floyd-Warshall (CORMEN et al., 2009) no grafo construído.

As instâncias adaptadas foram geradas convertendo algumas das instâncias do *k-median* disponíveis na OR-Library (BEASLEY, 1990). O processo de conversão mantém o conjunto de pontos de facilidades V, as distâncias $d(i, j)$ e o valor da constante k originais da instância. Os demais conjuntos, i.e. os instantes de tempo T, os clientes $\mathcal{D}$ e as durações $\{\delta_1, \ldots, \delta_{l_{max}}\}$, foram obtidas da mesma forma que as instâncias geradas.

As instâncias utilizadas foram separadas em três grupos, de acordo com os resultados obtidos pelo resolvedor Gurobi. No primeiro grupo estão as instâncias para as quais o resolvedor encontrou uma solução ótima dentro do tempo limite de 10 minutos. No segundo grupo estão as instâncias em que o resolvedor não encontrou uma solução ótima no tempo limite, porém apresentou algumas soluções viáveis, sendo utilizada a solução

incumbente para fins de comparação. Por fim, para as instâncias do terceiro grupo, o resolvedor sequer conseguiu obter uma solução viável dentro do prazo de 10 minutos.

As Tabelas 1-3 apresentam o volume dos dados e a origem das instâncias de cada grupo.

Tabela 1 – Descrição das instâncias do Grupo I

Instância	$\|V\|$	$\|T\|$	$\sum_{t\in T} \|D_t\|$	$\|L\|$	k	Origem
1	100	7	420	4	5	Adaptada da OR-Library
2	100	10	491	2	5	Adaptada da OR-Library
3	100	1	53	1	5	Adaptada da OR-Library
4	100	2	90	2	5	Adaptada da OR-Library
5	100	4	125	2	5	Adaptada da OR-Library
6	100	12	725	3	10	Adaptada da OR-Library
7	100	8	262	5	10	Adaptada da OR-Library
8	100	5	193	1	10	Adaptada da OR-Library
9	100	15	958	1	10	Adaptada da OR-Library
10	100	11	479	2	10	Adaptada da OR-Library

Fonte: elaborada pelos autores.

Tabela 2 – Descrição das instâncias do Grupo II

Instância	$\|V\|$	$\|T\|$	$\sum_{t\in T} \|D_t\|$	$\|L\|$	k	Origem
1	200	18	2029	10	10	Adaptada da OR-Library
2	200	17	1440	7	10	Adaptada da OR-Library
3	170	25	1923	8	15	Gerada pseudoaleatoriamente
4	170	36	2990	5	27	Gerada pseudoaleatoriamente
5	172	33	3037	8	13	Gerada pseudoaleatoriamente
6	183	25	2049	12	9	Gerada pseudoaleatoriamente
7	127	31	1963	18	14	Gerada pseudoaleatoriamente
8	193	19	2069	11	8	Gerada pseudoaleatoriamente
9	201	16	1427	8	12	Gerada pseudoaleatoriamente
10	230	8	1104	4	9	Gerada pseudoaleatoriamente

Fonte: elaborada pelos autores.

Tabela 3 – Descrição das instâncias do Grupo III

Instância	$\|V\|$	$\|T\|$	$\sum_{t\in T} \|D_t\|$	$\|L\|$	k	Origem
1	218	86	8751	23	25	Gerada pseudoaleatoriamente
2	268	89	12711	8	21	Gerada pseudoaleatoriamente
3	205	101	10754	32	10	Gerada pseudoaleatoriamente
4	322	31	4546	8	8	Gerada pseudoaleatoriamente
5	312	141	22980	10	16	Gerada pseudoaleatoriamente
6	348	42	7128	19	21	Gerada pseudoaleatoriamente
7	333	38	6350	19	9	Gerada pseudoaleatoriamente
8	493	50	13268	7	9	Gerada pseudoaleatoriamente
9	232	92	9870	29	25	Gerada pseudoaleatoriamente
10	503	30	7566	11	7	Gerada pseudoaleatoriamente

Fonte: elaborada pelos autores.

6.2 Definição dos Parâmetros

A fim de melhorar os resultados obtidos com as abordagens heurísticas, o pacote *irace* na versão 3.3 foi utilizado para auxiliar na definição dos parâmetros. O pacote *irace* faz uso de um algoritmo estatístico, chamado *iterated racing*, para selecionar, dentro de um conjunto de parâmetros, as configurações mais adequadas para que os algortimos gerem bons resultados (LÓPEZ-IBÁÑEZ; CÁCERES; DUBOIS-LACOSTE, 2019).

Para utilizar o *irace* devem ser especificados os conjuntos de instâncias de treino e de teste, o conjunto de parâmetros válidos e o tempo total disponível para realizar o ajuste de parâmetros. Os conjuntos de instâncias foram determinados separando 7 instâncias de cada grupo para treinamento e 3 para testes. O tempo total disponível para as execuções foi fixado em 72 horas. Para o BRKGA os parâmetros ajustados foram: o tamanho da população p entre 100 e 800, o tamanho do conjunto elite p_e entre $0,01 * p$ e $0,45 * p$, o tamanho do conjunto de mutantes p_m entre $0,01 * p$ e $0,45 * p$, e a probabilidade p_a de um gene ser herdado do pai elite entre $0,51$ e $0,95$. Para o VNS a quantidade de estruturas de vizinhança R foi ajustada entre 1 e 100.

Os valores obtidos para os parâmetros através do *irace* foram: $p = 496$, $p_e = 198$, $p_m = 38$, $p_a = 0,638$ e $R = 10$.

Quanto à configuração do Gurobi, além de manter os valores de parâmetros *default*, o resolvedor foi configurado para utilizar o método Dual-Simplex nas relaxações de problemas de programação linear inteira, visto que, em termos de memória, é a opção mais econômica dentre as disponíveis, possibilitando o tratamento de instâncias maiores.

6.3 Resultados

A métrica utilizada na comparação entre as soluções obtidas com as abordagens utilizadas foi o *gap*, calculado a partir da Fórmula 7, em que $val(S)$ é o valor da solução obtida pela abordagem e $val(S^*)$ é o valor ótimo ou um limitante inferior desse valor.

$$gap = \frac{[val(S) - val(S^*)] * 100}{val(S)} \tag{7}$$

A apresentação dos resultados foi dividida em três partes, uma para cada grupo de instâncias.

6.3.1 Grupo I

Para as instâncias de tamanho menor, representadas pelo Grupo I, o resolvedor encontrou soluções ótimas sem muita dificuldade, resolvendo todas as instâncias antes mesmo dos primeiros 30 segundos de execução.

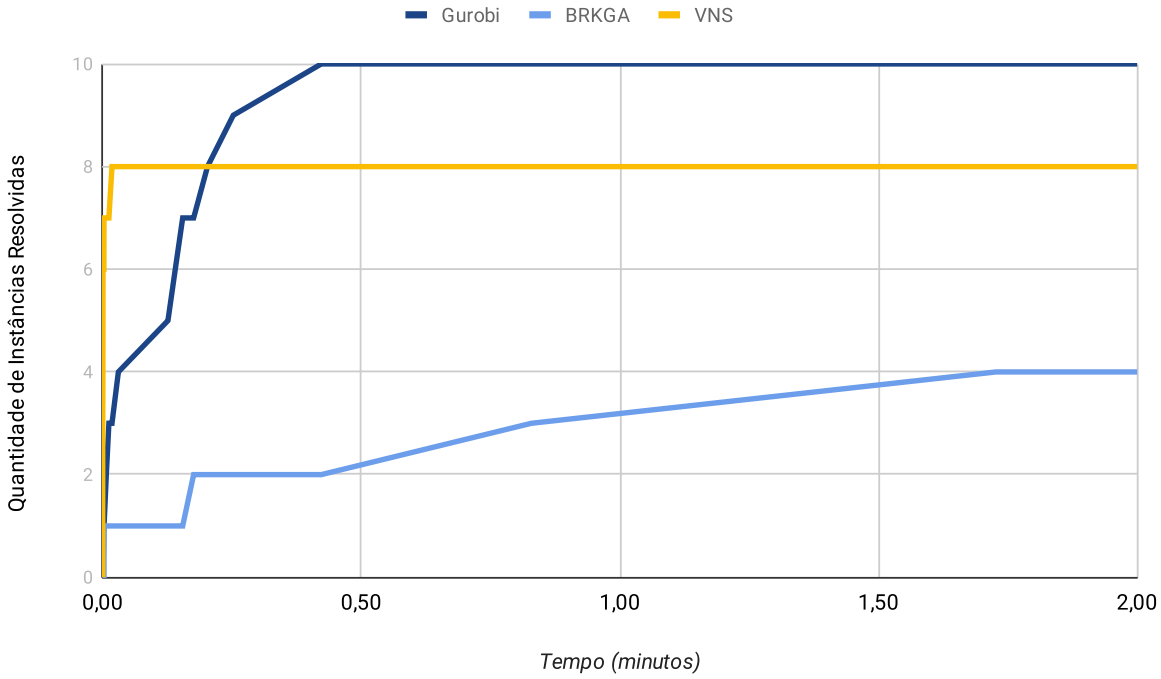
O BRKGA, no geral, obteve soluções com valores bem próximos ao valor ótimo, apresentando um *gap* médio de $0,45\%$. Contudo, somente em 4 das 10 instâncias do

Grupo I, a heurística encontrou uma solução ótima. Após 3 minutos de execução, observou-se uma certa convergência na população do algoritmo genético, o que torna interessante a exploração, em trabalhos futuros, de taxas de mutação mais elevadas ou de operadores de extermínio para reinicializar a população.

A abordagem baseada no VNS encontrou soluções ótimas para 8 das 10 instâncias, sendo que a última solução ótima foi obtida dentro do prazo de 1 segundo. Nas duas instâncias restantes, o algoritmo apresentou um *gap* médio de $0,07\%$, computando a última solução incumbente no prazo de 20 segundos.

A Figura 14 exibe de forma gráfica a quantidade de instâncias resolvidas por cada abordagem ao longo do tempo. Todos os algoritmos foram executados por 10 minutos, contudo, o gráfico expõe somente os primeiros 2 minutos, visto que após esse período não houve alteração na quantidade de instâncias resolvidas.

Figura 14 – Gráfico de desempenho para o Grupo I



Fonte: elaborada pelos autores.

A Tabela 4 apresenta o valor das soluções obtidas pelas três abordagens. Os valores ótimos estão destacados na cor verde.

6.3.2 Grupo II

O Grupo II é formado pelas instâncias em que o resolvedor não obteve soluçoes ótimas dentro do tempo limite de 10 minutos. Mesmo após executar por mais de 24 horas, uma solução ótima não foi encontrada para qualquer uma das três primeiras instâncias.

A fim de possibilitar uma análise através do *gap*, um limitante inferior do valor ótimo foi utilizado. Esse limitante foi obtido através da resolução da relaxação da formulação

Tabela 4 – Valores das soluções para o Grupo I

Instância	Gurobi	BRKGA	VNS
1	23168	23533	23168
2	27953	27987	27953
3	3025	3025	3025
4	4801	4801	4801
5	6785	6785	6785
6	28087	28099	28087
7	9497	9712	9520
8	6568	6568	6568
9	38773	38773	38773
10	19147	19263	19238

Fonte: elaborada pelos autores.

introduzida na Seção 3.2. A Tabela 5 apresenta os valores das soluções obtidas pelas três abordagens e o limitante inferior do valor ótimo. Para cada instância, a melhor solução apresentada pelas três abordagens está destacada na cor verde. O *gap* médio para as soluções incumbentes geradas pelo Gurobi foi de $20,51\%$.

Tabela 5 – Valores das soluções para o Grupo II

Instância	Gurobi		BRKGA		VNS		Limitante Inferior
	$val(S)$	$gap(\%)$	$val(S)$	$gap(\%)$	$val(S)$	$gap(\%)$	
1	55221	0,32	56748	3,00	55336	0,52	55046,00
2	58086	32,54	40684	3,69	39701	1,31	39182,74
3	216618	7,38	230816	13,08	205280	2,26	200630,73
4	225868	1,47	279688	20,43	225390	1,26	222539,84
5	476682	25,55	398302	10,9	365034	2,78	354896,15
6	352198	24,03	296664	9,81	274656	2,59	267551,63
7	345736	41,96	229194	12,44	203708	1,49	200678,65
8	411944	28,58	314184	6,36	302536	2,75	294201,28
9	220166	24,95	183314	9,87	169936	2,77	165228,04
10	187872	18,34	162034	5,32	158424	3,16	153410,81

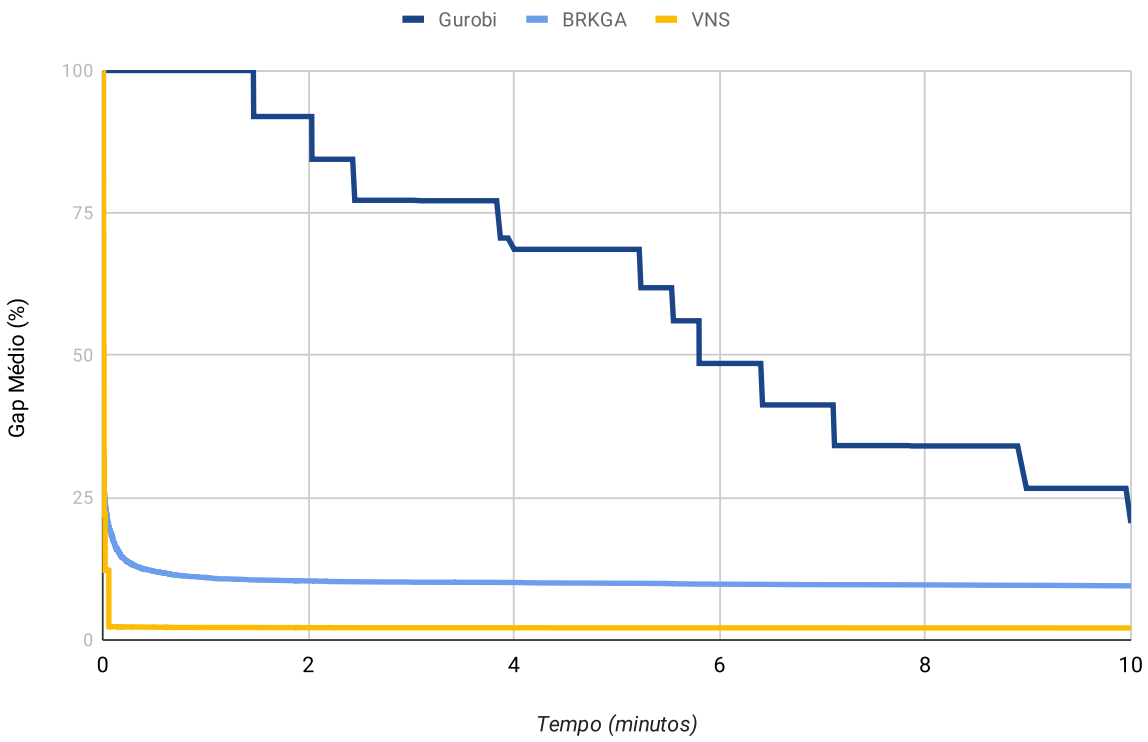
Fonte: elaborada pelos autores.

O VNS obteve a melhor solução dentre as três abordagens em 9 das 10 instâncias, apresentando um *gap* médio de $2,08\%$. Logo nos primeiros 2 minutos de execução a abordagem em questão obteve um *gap* de $2,1\%$. Note que é possível que, para alguma instância do Grupo II, o VNS tenha obtido alguma solução ótima, contudo, como o valor ótimo não pôde ser computado em tempo hábil, não é possível fornecer garantia de otimalidade.

A Figura 15 exibe um gráfico de desempenho relacionando o *gap* médio com o tempo de execução em minutos.

Diferente do que ocorreu no Grupo I, as soluções incumbentes do BRKGA continuaram melhorando mesmo com o tempo de execução se aproximando do tempo limite. De forma geral, o *gap* médio foi de $9,49\%$ e a solução mais distante do ótimo foi a da instância de número 4. Para essa instância a heurística obteve um *gap* de 30% nos primeiros

Figura 15 – Gráfico de desempenho para o Grupo II



Fonte: elaborada pelos autores.

segundos de execuçao e, em média, a cada 4 gerações a solução incumbente melhorava.

6.3.3 Grupo III

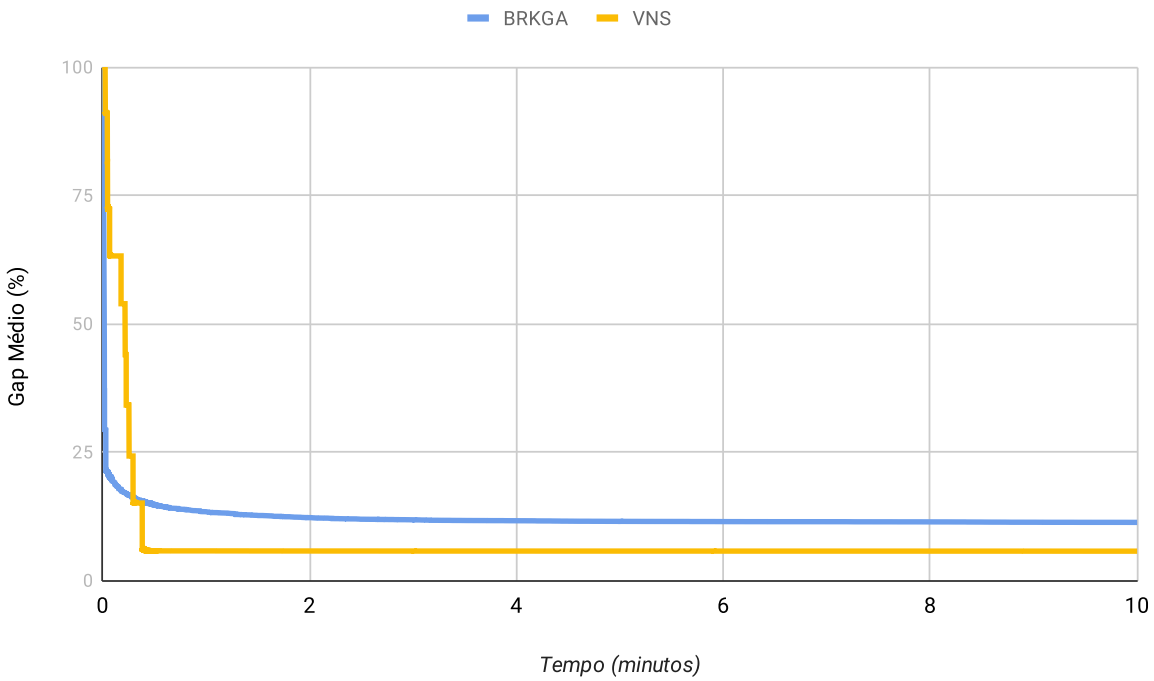
Para as instâncias de tamanho maior, representadas pelo Grupo III, o resolvedor comercial não encontrou soluções viáveis dentro do tempo limite. Nem sequer, para a relaxação da formulação de programação linear inteira foi possível obter uma solução ótima, por conta do tempo de execução necessário ser muito elevado ou por causa da quantidade de variáveis geradas pela formulação exceder o limite de memória da máquina de testes.

Um limitante inferior do valor ótimo foi obtido tratando cada instante de tempo como uma instância separada do problema *k-median*, semelhante ao que foi feito na Seção 5.3.3.1. Os limitantes inferiores de cada instância do *k-median* foram, então, computados, resolvendo a relaxação da formulação proposta por AHN et al. (1988), e somados.

A Figura 16 exibe de forma gráfica o comportamento do *gap* médio para as instâncias do Grupo III ao longo do tempo. Nos primeiros segundos de execução, o BRKGA apresentou soluções melhores do que as do VNS. Após concluir o cálculo da última solução inicial, aproximadamente em 15 segundos de execução, o VNS obteve um *gap* médio de $15,30\%$, alcançando o BRKGA.

A Tabela 6 apresenta os limites inferiores, os valores de solução e os *gaps* obtidos com o BRKGA e o VNS. Em média, o BRKGA alcançou um *gap* de $11,31\%$, ao passo

Figura 16 – Gráfico de desempenho para o Grupo III



Fonte: elaborada pelos autores.

que o VNS obteve um *gap* de $5,70\%$. Note que o limitante inferior utilizado, além de ser proveniente de um modelo com variáveis contínuas, foi obtido removendo as durações, um dos elementos principais do problema *leasing k-median*, o que pode tornar a comparação imprecisa.

Tabela 6 – Valores das soluções para o Grupo III

Instância	BRKGA		VNS		Limitante Inferior
	$val(S)$	$gap(\%)$	$val(S)$	$gap(\%)$	
1	799712	16,31	675047	0,85	669297,34
2	1215812	10,58	1184862	8,24	1087204,41
3	1170448	10,56	1120546	6,58	1046815,84
4	575898	12,27	562649	10,20	505247,11
5	2421770	10,11	2336756	6,84	2177014,39
6	688221	8,82	638856	1,77	627518,62
7	740766	10,78	700088	5,60	660895,74
8	1644154	10,23	1586917	6,99	1476018,14
9	887275	12,97	778329	0,79	772193,86
10	953561	10,52	940041	9,23	853237,37

Fonte: elaborada pelos autores.

7 CONCLUSÃO

Neste trabalho de conclusão de curso foi abordado um problema de otimização combinatória, chamado *leasing k-median*. Para esse problema são dados um conjunto de pontos onde facilidades podem ser abertas, um conjunto de instantes de tempo discretos, conjuntos de clientes que precisam ser servidos para cada instante de tempo, as durações disponíveis para a abertura de uma facilidade e um limite de facilidades abertas k. O objetivo do problema é determinar quais, quando e por quanto tempo as facilidades devem ser abertas para que a soma total das distâncias de cada cliente à facilidade ativa mais próxima seja mínima, respeitando a restrição de que no máximo k facilidades estejam abertas ao mesmo tempo.

Para tratar o problema foram utilizadas três abordagens: uma exata e duas heurísticas. A abordagem exata consistiu em utilizar uma formulação de programação linear inteira, adaptada da literatura, junto a um resolvedor comercial. Através dos experimentos computacionais realizados, foi possível perceber que o método exato é bastante útil na resolução de instâncias pequenas, encontrando soluções ótimas rapidamente. Contudo, quando foram utilizadas instâncias de médio e de grande porte, o resolvedor apresentou dificuldades para encontrar boas soluções, requisitando recursos computacionais e tempo de execução elevados.

A primeira abordagem heurística utilizada foi o BRKGA. Ao longo do trabalho foram propostos uma representação para soluções e um algoritmo decodificador. Nos experimentos computacionais, o BRKGA, dentre as três abordagens, obteve o pior desempenho para as instâncias de tamanho menor ao não conseguir melhorar a solução incumbente para 6 de 10 dessas instâncias após 2 minutos de execução. Para trabalhos futuros envolvendo essa meta-heurística, podem ser analisados uma taxa de mutação mais elevada ou operadores de extermínio para reinicializar a população. Além disso, um pré-processamento para excluir pontos de facilidades que não contribuam tanto para um bom valor de solução, pode ser uma boa alternativa para reduzir o espaço de busca e melhorar o desempenho.

A segunda meta-heurística utilizada foi o VNS. Para as instâncias de tamanho menor, o VNS solucionou 8 das 10 instâncias em tempo inferior ao das outras abordagens. Além disso, para as instâncias de médio e grande porte a heurística apresentou os melhores resultados dentre as três abordagens, com *gaps* médios, em relação a limitantes inferiores do valor ótimo, de $2,08\%$ e $5,70\%$, respectivamente.

Para trabalhos futuros podem ser abordadas variações do problema *leasing k-median* como, por exemplo, uma versão capacitada em que cada ponto de facilidade possui um limite máximo para a quantidade de clientes que podem ser servidos. Outra possibilidade, é incluir uma restrição secundária para que a quantidade de trocas de facilidades seja a menor possivel, reduzindo, na prática, custos com a mudança de instalações.

REFERÊNCIAS

AHN, S.; COOPER, C.; CORNUÉJOLS, G.; FRIEZE, A. Probabilistic Analysis of a Relaxation for the k-Median Problem. **Mathematics of Operations Research**, v. 13, n. 1, p. 1–31, 1988.

ARYA, V.; GARG, N.; KHANDEKAR, R.; MEYERSON, A.; MUNAGALA, K.; PANDIT, V. Local search heuristics for k -median and facility location problems. **SIAM Journal on Computing**, v. 33, 01 2003.

BARCAROLI, V. Consulta em vetores com segment tree e lazy propagation. **Revista Tecnológica**, v. 4, n. 1, p. 71 – 85, 2016.

BEASLEY, J. E. Or-library: Distributing test problems by electronic mail. **The Journal of the Operational Research Society**, Palgrave Macmillan Journals, v. 41, n. 11, p. 1069–1072, 1990.

BELFIORE, P. P.; COSTA, O. L. do V.; FÁVERO, L. P. L. Decomposição de benders e suas aplicações em modelos de programação mista e em problemas de estoque e roteirização. In: **XXXVII Simpósio Brasileiro de Pesquisa Operacional**. Gramado, RS, Brasil: [s.n.], 2005. p. 2453–2464.

BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. **Information Sciences**, v. 237, p. 82 – 117, 2013.

BYRKA, J.; PENSYL, T.; RYBICKI, B.; SRINIVASAN, A.; TRINH, K. An improved approximation for k -median and positive correlation in budgeted optimization. **ACM Transactions on Algorithms**, v. 13, p. 1–31, 03 2017.

CARVALHO, M.; CERIOLI, M.; DAHAB, R.; FEOFILOFF, P.; FERNANDES, C.; FERREIRA, C.; GUIMARãES, K.; MIYAZAWA, F.; JR, J.; SOARES, J.; WAKABAYASHI, Y. **Uma Introdução Sucinta a Algoritmos de Aproximação**. [S.l.: s.n.], 2001.

CHARIKAR, M.; GUHA, S.; TARDOS, ; SHMOYS, D. A constant-factor approximation algorithm for the k-median problem. **Proceedings of 31st Annual ACM Symposium on Theory of Computing**, p. 1–10, 01 1999.

CHEN, Y.-Y.; CHENG, C.-Y.; WANG, L.-C.; CHEN, T.-L. A hybrid approach based on the variable neighborhood search and particle swarm optimization for parallel machine scheduling problems—a case study for solar cell industry. **International Journal of Production Economics**, v. 141, n. 1, p. 66 – 78, 2013.

CONFORTI, M.; CORNUÉJOLS, G.; ZAMBELLI, G. **Integer Programming**. [S.l.]: Springer, 2014.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. [S.l.]: The MIT Press, 2009.

FELDMAN, E.; LEHRER, F. A.; RAY, T. L. Warehouse location under continuous economies of scale. **Management Science**, v. 12, n. 9, p. 670–684, 1966.

FERLAND, J. A.; COSTA, D. Heuristic search methods for combinatorial programming problems. 03 2001.

FÜGENSCHUH, A.; MARTIN, A. Computational integer programming and cutting planes. **Discrete Optimization**, v. 12, p. 69–121, 12 2005.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman & Co., 1990.

GENDREAU, M.; POTVIN, J.-Y. **Handbook of Metaheuristics**. [S.l.]: Springer, 2010.

GOLDBARG, M.; LUNA, H. **Otimização Combinatória e Programação Linear**. [S.l.]: Elsevier, 2005.

GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, v. 17, n. 5, p. 487–525, 2011.

Gurobi Optimization. **The Fastest Solver**. 2019. <https://www.gurobi.com/>. Acessado em: 01-11-2019.

HALIM, S.; HALIM, F. **Competitive Programming 3**. [S.l.]: Lulu, 2013.

IBM. **CPLEX Optimizer**. 2019. <https://www.ibm.com/br-pt/analytics/cplex-optimizer>. Acessado em: 01-11-2019.

KARIV, O.; HAKIMI, S. L. An Algorithmic Approach to Network Location Problems. II: The p-Medians. **SIAM Journal on Applied Mathematics**, v. 37, n. 3, p. 539–560, 1979.

KARMARKAR, N. A new polynomial-time algorithm for linear programming. In: **Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: [s.n.], 1984. (STOC '84), p. 302–311.

KHACHIYAN, L. Polynomial algorithms in linear programming. **USSR Computational Mathematics and Mathematical Physics**, v. 20, n. 1, p. 53 – 72, 1980.

KLEINBERG, J.; TARDOS, E. **Algorithm Design**. [S.l.]: Pearson, 2005.

KUEHN, A. A.; HAMBURGER, M. J. A heuristic program for locating warehouses. In: _____. **Mathematical Models in Marketing: A Collection of Abstracts**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1976. p. 406–407.

LI, S.; SVENSSON, O. Approximating k-median via pseudo-approximation. **SIAM Journal on Computing**, v. 45, p. 530–547, 01 2016.

LINTZMAYER, C. N.; MOTA, G. O. Caderno de problemas. In: **1º Workshop Paulista em Otimização, Combinatória e Algoritmos**. [S.l.: s.n.], 2017.

LÓPEZ-IBÁÑEZ, M.; CÁCERES, L. P.; DUBOIS-LACOSTE, J. The irace package: User guide. 04 2019.

MAKHORIN, A. **GLPK (GNU Linear Programming Kit)**. 2012. <https://www.gnu.org/software/glpk/>. Acessado em: 01-11-2019.

MEINDL, B.; TEMPL, M. Analysis of commercial and free and open source solvers for linear optimization problems. 02 2012.

MICHALEWICZ, Z.; FOGEL, D. B. **How to Solve It: Modern Heuristics**. [S.l.]: Springer, 2004.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers & Operations Research**, v. 24, n. 11, p. 1097 – 1100, 1997.

MRV Engenharia. **MRV Engenharia**. 2019. <https://www.mrv.com.br>. Acessado em: 30-04-2019.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization: Algorithms and Complexity**. [S.l.]: Dover, 1998.

RESENDE, M. G. C. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. In: **XVL Simpósio Brasileiro de Pesquisa Operacional**. Natal, RN, Brasil: [s.n.], 2013. p. 3680–3691.

SCHMIDT, C. E.; OLIVEIRA, C. R. V. de; SILVA, A. C. L. da. Performance de dois solvers na resolução do problema de transporte com custo linear por partes. In: **XLIX Simpósio Brasileiro de Pesquisa Operacional**. Blumenau, SC, Brasil: [s.n.], 2017.

SOUZA, A. S. de; PÉCORA, J. E.; LOCH, G. V.; FRESSATO, A. A. Performance de dois solvers na resolução da metaheurística fix and optimize aplicado ao problema de high school timetabling. In: . [S.l.: s.n.], 2017.

WILLIAMSON, D. P.; SHMOYS, D. The design of approximation algorithms. **The Design of Approximation Algorithms**, 01 2011.