GUILHERME PEREIRA PORTO LONDE

HEURÍSTICAS PARA O PROBLEMA *LEASING K-CENTER*

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: Alexandre Ribeiro
Coorientador: Welverton Rodrigues da Silva

GOIÂNIA
2019

Altere este texto inserindo a dedicatória do seu trabalho.

AGRADECIMENTOS

Agradeço, acima de tudo, a Deus. Reconheço que eu não chegaria onde estou se não fosse por Ele e para Ele. Agradeço imensamente o meu coorientador, que apesar das diversas responsabilidades, voluntariamente dedicou grande parte do seu tempo para esta orientação, fornecendo sugestões valiosas e me preparando para a vida de pesquisador. Agradeço imensamente o meu orientador pelas sugestões valiosas e também por ter providenciado o ambiente e os recursos para que este trabalho fosse realizado. Por fim, também agradeço o meu colega Jorge M. dos Santos pela coparticipação na produção do resumo expandido que foi o precursor deste trabalho de conclusão de curso e pelas dicas na utilização de várias das ferramentas que foram utilizadas aqui.

RESUMO

O problema *k-center* é um problema de otimização combinatória bastante conhecido na literatura. Este texto apresenta uma generalização deste problema, conhecida como *leasing k-center*. Neste texto o problema *leasing k-center* é apresentado formalmente e informalmente, assim como são apresentados os principais conceitos envolvidos. Possíveis aplicações também são apresentadas. Mostramos que o problema *k-center* é bastante trabalhado na literatura e que várias abordagens foram desenvolvidas ao longo dos anos para solucioná-lo e, com isso, também afirmamos que o problema *leasing k-center* possui relevância. Acreditamos que o problema *leasing k-center* não tenha sido trabalhado com experimentos computacionais, portanto, este trabalho trabalho de conclusão de curso vem com o objetivo de servir como base para futuros trabalhos na área. Aqui é proposto uma formulação em Programação Linear, uma heurística baseada na meta-heurística BRKGA e uma outra heurística baseada na meta-heurística *Local Search* para encontrar soluções viáveis ao *leasing k-center*, assim como é feita uma comparação entre esses algoritmos experimentalmente.

Palavras-chave: Otimização Combinatória; Localização de Facilidades; Programação Linear; Heurísticas; *Leasing k-center*.

ABSTRACT

The k-center problem is a well known combinatorial optimization problem. This text presents a generalization of this problem, known as leasing k-center. In this text we present the leasing k-center problem formally and informally, as well as the main concepts involved. Possible applications are also presented. We show that the k-center problem is extensively worked on literature and several approaches have been proposed over the years to solve it and so we also states that the problem leasing k-center has relevance. We believe that the problem leasing k-center was not worked with computational experiments, so this conclusion of course work takes the purpose to serve as a basis to future works of the area. Here we propose a formulation on Linear Programming, a heuristic based on the metaheuristic BRKGA and another heuristic based on the metaheuristic Local Search to obtain solutions for the leasing k-center, as well as we compare these algorithms experimentally.
Keywords: Combinatorial Optimization; Facility Location; Linear Programming; Heuristics; Leasing k-center.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O problema *k-center* (HAKIMI, 1964; HAKIMI, 1965) e variantes são problemas de otimização combinatória frequentemente abordados na literatura. No problema *k-center* clássico, temos um grafo completo com distância associada a cada aresta e queremos selecionar k vértices de tal forma que minimize a maior distância entre qualquer um dos vértices selecionados ao vértice mais próximo. Conceitos sobre grafos são apresentados no Capítulo 3.

Este trabalho de conclusão de curso apresenta resultados para uma variante do *k-center*, denominada *leasing k-center* (LKC). Neste trabalho propomos uma formulação de programação Linear e duas heurísticas para a resolução do LKC.

Considere, a título de problema motivacional, o problema prático a seguir que envolve alocação de antenas para atender estações de pesquisa científica.

Suponha que uma organização possui estações de pesquisa científica em uma determinada região e são conhecidos de antemão os períodos de tempo em que cada estação estará ativa ou desativada. Essa organização precisa alocar antenas de telecomunicação ao longo de determinados períodos de tempo para estabelecer uma rede de comunicação entre todas as estações ativas. Cada estação ativa é servida pela antena em operação mais próxima e uma antena em operação pode servir mais de uma estação ativa ao mesmo tempo.

A organização possui k antenas que podem ser utilizadas para servir a região. Por questões de logística, há um tempo mínimo de alocação de uma antena, e cada antena deve estar localizada em alguma das estações da organização.

A distância de uma estação ativa à antena em operação que a serve deve ser a menor possível visto que a comunicação entre elas ocorre via rádio e por equipamentos pouco eficientes, que enviam o sinal próximo ao solo e, por isso, está sujeita a ruídos e à rápida atenuação do sinal. Por esse motivo considera-se que a maior distância entre uma estação ativa e a antena em operação mais próxima determina a qualidade de serviço de toda a rede de comunicação.

A comunicação entre antenas é pouco afetada por ruído porque é realizada por equipamentos mais eficientes e, talvez, por um canal de comunicação diferente. Por esse motivo considera-se que a distância entre as antenas em operação não interfere na qualidade de serviço da rede de comunicação. Dito isso, a organização pretende alocar as antenas de forma que a maior distância de uma estação ativa à antena em operação mais próxima seja minimizada.

A Figura 1 ilustra uma instância específica do problema de alocação de antenas, bem como uma solução que minimiza a maior distância entre uma estação ativa à antena em operação mais próxima. Nessa mesma instância são considerados cinco períodos de

tempo sequencialmente numerados a partir de 0, a organização possui duas antenas que podem ser utilizadas, e cada antena deve ser alocada por não menos que dois períodos consecutivos. Por este motivo consideramos dois tipos de alocação: de dois e de três períodos. As estações de pesquisa são distribuídas no espaço conforme os vértices de um polígono regular. Quando uma antena serve uma estação que não está na mesma localidade um canal de comunicação via rádio é mostrado entre elas.

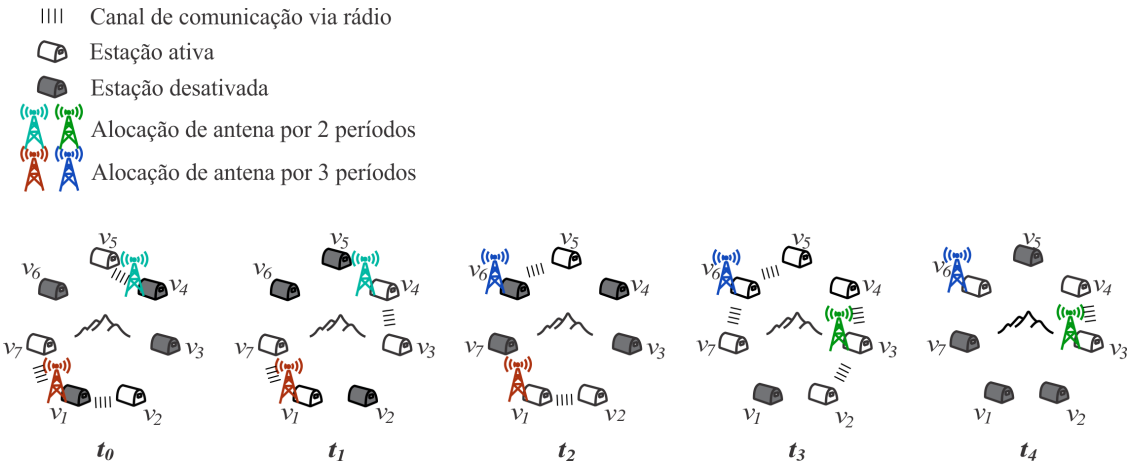

Figura 1 – Instância do problema da alocação de antenas.

Na instância ilustrada pela Figura 1, cada uma das sete estações se alternam entre ativa ou inativa ao longo dos cinco períodos de tempo. A solução ótima representada consiste em realizar quatro alocações: uma antena por três períodos de tempo na estação v_1 e uma antena por dois períodos de tempo na estação v_4, ambos a partir do período t_0; uma antena por três períodos de tempo na estação v_6 a partir do período t_2; e uma antena por dois períodos de tempo na estação v_3 a partir do período t_3. O custo da solução é igual a maior distância entre uma estação ativa e a antena em operação mais próxima, que neste caso é a distância entre duas estações adjacentes. Este custo também pode ser visto como o raio da circunferência de cobertura válida para todos os períodos, conforme pode ser visto na Figura 2.

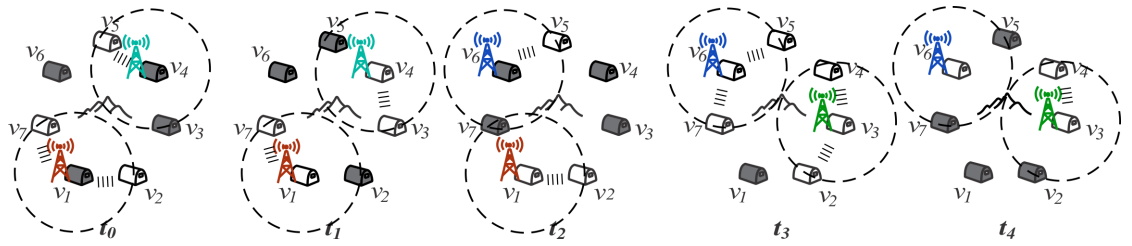

Figura 2 – Circunferência de cobertura.

Uma tendência em logística é a utilização de aluguéis e terceirizações de serviços, isto é, organizações que não possuem um determinado serviço e preferem utilizar os

serviços de terceiros, minimizando o custo operacional. Em algumas dessas aplicações há um limitante de quantos serviços podem ser realizados em um mesmo período, e o problema de alocação de antenas também pode ser visto como uma aplicação desse tipo, uma vez que podemos considerar que a organização não possui os serviços mas recebe uma quantia fixa de período em período para alugá-los.

O problema da alocação de antenas também pode ser visto como o problema de terceirizar funcionários via contrato por prazo determinado para servir diferentes departamentos de uma organização, como um problema de alocar pacotes de dados pré-pagos a pontos de roteamento, como um problema de alugar residências ou depósitos, entre outros, pois em tais casos cada tipo alocação é previamente definido. Estes problemas são aplicações práticas do problema *leasing k-center* (LKC), proposto inicialmente em (LINTZMAYER; MOTA, 2017).

Informalmente, o problema pode ser definido da seguinte maneira. Sejam: um conjunto V de localidades, no qual os clientes e as facilidades podem estar instalados; um conjunto $T = \{0, \ldots, t_{max}\}$ de períodos de tempo, onde t_{max} é o período máximo considerado; uma coleção $\mathcal{D}$, onde cada subconjunto D_t de $\mathcal{D}$ são as localidades dos clientes que permanecem ativos no período t, e as localidades em $V - D_t$ são os clientes que permanecem inativos no período t; o conjunto $L = \{1, \ldots, l_{max}\}$ dos tipos de alocações e outro conjunto $\Delta = \{\delta_1, \ldots, \delta_{l_{max}}\}$ em que δ_l é a duração da alocação do tipo l. Ademais, é dado um inteiro positivo k que indica o número máximo permitido de facilidades ativas em cada período de tempo.

Uma solução para o problema consiste em um conjunto $\mathcal{M}$, onde cada elemento é uma tupla (i, l, t), para i em V, l em L e t em T, que indica que a facilidade i é aberta no período t e permanece ativa nos períodos entre t e $t + \delta_l - 1$. O objetivo consiste em encontrar um conjunto $\mathcal{M}$ que minimize a maior distância para atender cada cliente em D_t a uma facilidade ativa mais próxima, restrito a não haver mais do que k facilidades ativas no período t.

Na instância para o problema motivação que é ilustrada na Figura 1, o conjunto V de clientes consiste em $V = \{v_1, v_2, \ldots, v_7\}$, o valor t_{max} é igual a 4, cada estação v_i que é ativa em um período t pertence ao conjunto D_t, $\Delta = \{2, 3\}$ ou, de outra forma, $\delta_{L_1} = 2$ e $\delta_{L_2} = 3$ e, por fim, $k = 2$. A solução ótima ilustrada é $\mathcal{M} = \{(v_1, 2, 0), (v_4, 1, 0), (v_6, 2, 2), (v_3, 1, 3)\}$.

O problema *k-center* é NP-difícil (KARIV; HAKIMI, 1979). O *k-center* é o caso particular do LKC quando há apenas um período de tempo a ser considerado, e portanto, é correto afirmar que o LKC também é NP-difícil. Assim sendo, é pouco provável que exista um algoritmo de tempo polinomial para resolvê-lo. Neste contexto, obter uma solução de forma exata para instâncias reais pode demandar um tempo computacional inviável, sendo necessário a utilização de heurísticas para encontrar soluções, isto é, algoritmos que devolvem uma solução viável mas sem a garantia da qualidade da solução em relação à

ótima.

Para obtenção de soluções viáveis para o LKC duas heurísticas foram propostas neste trabalho. Uma heurística utiliza a meta-heurística BRKGA e é capaz de apresentar soluções em um tempo relativamente curto, e pode ter sua duração extendida a fim de melhorar a qualidade da solução, utilizando não muita memória. A outra heurística proposta neste trabalho utiliza a meta-heurística *Local Search* e apresenta soluções tão boas quanto as soluções da heurística de BRKGA para o LKC, levando em média menos tempo, porém utiliza bastante memória e pode, nos piores casos, demandar um tempo computacional excessivamente longo.

O restante do texto está estruturado como segue. O capítulo 2 apresenta uma revisão da literatura a respeito de problemas relacionados ao *leasing k-center*. O capítulo 3 contém um glossário detalhado sobre os principais termos e notações relacionados a esse trabalho, mas que não pertencem unicamente ao escopo do problema que é analizado aqui. O capítulo 4 contém a descrição formal, uma formulação de programação linear inteira mista e duas heurísticas para o *leasing k-center*. O capítulo 5 apresenta os resultados obtidos, mostrando também a forma como foram realizados os experimentos computacionais.

2 REVISÃO BIBLIOGRÁFICA

No problema de localização de facilidades (*facility location problem*) é dado um conjunto de potenciais localizações para a instalação de facilidades e um conjunto de clientes que devem ser atendidos por estas facilidades. Nesses problemas queremos instalar as facilidades em algumas das localizações, de tal forma que o custo para atender os clientes seja o menor possível, restrito ao número de facilidades instaladas não ser maior do que um valor definido.

O *k-center* é um problema de localização de facilidades que tornou possível modelar uma série de aplicações que não eram passíveis de serem modeladas com outras abordagens até então. Quando o problema *k-center* foi proposto em (HAKIMI, 1964), um caso considerado como possível aplicação era o de instalar pontos de serviço de emergência (tais como estações policiais ou hospitais) em rodovias para servir um conjunto de cidades. Nesse caso é importante minimizar a maior distância entre um cliente e o serviço de emergência mais próximo. O problema *k-center* é modelado como um problema em grafos, no qual o conjunto de vértices pode ou não satisfazer a desigualdade triangular (ver propriedade 3 da definição de espaço métrico na Seção 3.2). Consideramos a versão do problema onde facilidades devem ser instaladas nos vértices do grafo (*vertex k-center*). As Figuras 3 e 4 ilustram um cenário do *vertex k-center* para o problema da instalação de hospitais.

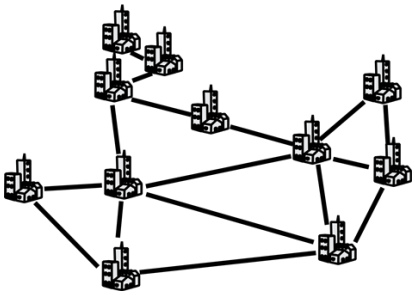

Figura 3 – Conjunto de cidades.

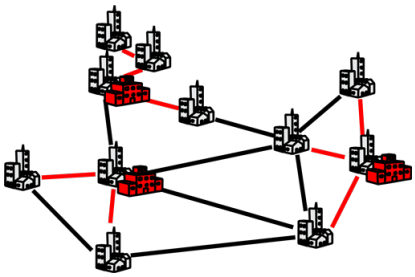

Figura 4 – Pontos de emergência: $k = 3$.

Várias abordagens exatas foram propostas para o *k-center*. Para encontrar a solução ótima no *k-center*, destacamos o algoritmo de (MINIEKA, 1970), que consiste em resolver uma série de problemas de conjunto de cobertura mínimo (*minimum set-cover*), e que é utilizado como referência para a construção da heurística da Seção 4.4. Este mesmo algoritmo toma por um diferente conjunto a união de um centro e os clientes alcançados por tal centro em uma distância não maior do que um limiar. Este limiar é decrementado se o número de conjuntos necessários para cobrir todo o grafo for menor do que ou igual a k, ou é incrementado caso contrário. O valor final dessa distância de cobertura é o valor da função objetivo de uma solução ótima para o problema. Mais tarde estratégias

bem elaboradas foram propostas para resolver o problema de forma exata. Dentre estas destacamos a formulação de (DASKIN, 1995) em programação linear inteira mista, que é utilizada como referência para a formulação da Seção 4.2.

Algoritmos de aproximação também têm sido utilizados para encontrar soluções viáveis para o problema *k-center*. Em (HSU; NEMHAUSER, 1979) é demonstrado que uma α-aproximação para o *vertex k-center* é NP-difícil, para $\alpha < 2$, se o grafo satisfaz a desigualdade triangular, caso contrário P=NP. Algoritmos de aproximação com fator igual a 2 para o *k-center* têm sido propostos. Um desses algoritmos foi proposto em (HOCHBAUM; SHMOYS, 1985). Esse último (HS) é um algoritmo guloso que encontra soluções para o *k-center* em tempo $O(|E|log|E|)$, onde E é o conjunto de arestas de um grafo completo $G = (V, E)$, utilizando a técnica *parametric prunning*.

Um dos primeiros trabalhos a propor heurísticas com base em meta-heurísticas foi (MLADENOVIC; HANSEN, 2003). Neste mesmo trabalho as meta-heurísticas *Tabu Search*, *Variable Neighborhood Search* (VNS) e *Multistart Local Search* foram utilizadas para encontrar soluções para o *k-center*. Através de experimentos, tais heurísticas obtiveram um desempenho melhor do que outras heurísticas clássicas até então, como o HS.

Uma classe de problemas semelhantes ao LKC, os quais chamamos de *facility leasing problems*, foi introduzida em (ANTHONY; GUPTA, 2007). Em tais problemas, diferentemente do LKC, a restrição de não haver mais do que k facilidades ativas em um mesmo período de tempo é ignorada, dando espaço para outra restrição: um custo associado a manter uma facilidade ativa. Dessa forma, o objetivo do problema é minimizar a soma do custo de servir os clientes nos períodos considerados com o custo de manter as facilidades ativas. Este problema já foi estudado na literatura e abordagens interessantes já foram propostas. Destacamos em especial um algoritmo de aproximação com fator igual a 3 proposto em (NAGARAJAN; WILLIAMSON, 2013), que utiliza uma variante do algoritmo conhecido como *primal-dual facility location*.

Acreditamos que outros problemas de localização de facilidades da mesma categoria do LKC, isto é, que possuem a restrição de não haver mais do que k facilidades abertas em um mesmo período, não tenham sido estudados até o início deste trabalho. Uma literatura preliminar, que é parte desse mesmo trabalho de conclusão de curso, foi publicada em (SANTOS et al., 2019). Neste resumo expandido o LKC é introduzido juntamente com outro problema da mesma categoria, o *leasing k-median*. Ambos os problemas foram descritos formalmente e formulados em programação linear inteira, uma heurística baseada no BRKGA foi construida para ambos os problemas, incluindo experimentos computacionais. Os algoritmos e resultados deste mesmo artigo referentes ao LKC foram replicados e estendidos neste trabalho.

3 CONCEITOS

Os conceitos e definições apresentados nesta seção são necessários para a compreensão do problema *leasing k-center* e dos algoritmos que serão analisados ao longo do trabalho de conclusão de curso.

3.1 Conceitos de Teoria dos Grafos

Um *grafo* G é um par ordenado (V_G, E_G), onde V_G é um conjunto (finito) e não vazio de objetos, chamados *vértices*, e E_G é um conjunto (finito) de *arestas*, onde cada aresta e de E_G é uma associação entre dois vértices u e v de V_G na forma de um par ordenado (u, v). Por simplicidade, caso não haja ambiguidade, denotaremos os conjuntos de vértices e arestas por V e E, respectivamente.

Consideramos o *custo* de uma aresta e em E, denotado por $w(e)$, como uma função $w : E \to \mathbb{R}_{\geq 0}$. Um *caminho* entre u e v, para u e v em V, é uma cadeia de arestas $C = (u, r_1)(r_1, r_2) \dots (r_s, v)$, tal que para todo e em C, e em E, e o custo de um caminho C é definido como $\sum w(e)$, para todo e em C. Denotaremos por $w(C)$ o custo de um caminho C.

Um *caminho mínimo* entre u e v, para u e v em V, é definido como um caminho C^* entre u e v tal que $w(C^*)$ é o menor possível, e o custo do caminho mínimo entre u e v é denotado por $d(u, v)$.

3.2 Espaço Métrico

Um *espaço métrico* é um par ordenado (V, d), onde V é um conjunto não vazio de pontos no espaço n-dimensional, e d é uma função métrica $d\colon V \times V \to \mathbb{R}_{\geq 0}$ que representa o custo do caminho mínimo entre quaisquer dois elementos de V. Por definição, conforme em (SHIRALI; VASUDEVA, 2006), uma função métrica d deve satisfazer as seguintes propriedades:

1. *não-negativa* e *separa pontos distintos*: para quaisquer $(i, j) \in V \times V$, $d(i, j) = 0$ se e somente se $i = j$;
2. *simetria*: para quaisquer $(i, j) \in V \times V$, $d(i, j) = d(j, i)$;
3. *desigualdade triangular*: para quaisquer $i, j, k \in V$, $d(i, j) \leq d(i, k) + d(k, j)$.

3.3 O problema *Minimum Set Cover*

Seja uma coleção $S = \{S_1, \dots, S_m\}$ e seja U a união de todos os subconjuntos de S. No problema *Minimum Set Cover* queremos encontrar uma cobertura C para S, isto é,

$C \subseteq S$ tal que $U = C_1 \cup \cdots \cup C_n$, de forma que o tamanho de C seja o menor possível. Tal problema é NP-Completo (KARP, 1972).

3.4 O problema *Lower-Upper Bounded Cover*

O *Lower-Upper Bounded Cover* (LUBC) é um problema que acreditamos não ter sido definido na literatura. Tal problema encontra aplicação no *leasing k-center*.

Seja um conjunto (finito) universo U, equipado com as funções demanda $req : U \to \mathbb{Z}^{\geq}$ e capacidade $cap : U \to \mathbb{Z}^{+}$, onde $req(u) \leq cap(u)$ para todo u de U, e seja uma coleção $S = \{S_1, \ldots, S_m\}$, onde $S_j \subseteq U$ para $1 \leq j \leq m$. No problema de decisão é dado U, S e um natural n; queremos então verificar se há um conjunto S^* de tamanho n ou menor, tal que $S^* \subseteq S$ e $req(u) \leq |\{S_j^* \in S^* : S_j^* \ni u\}| \leq cap(u)$ para todo u de U. No problema de otimização são dados U e S; queremos então encontrar um conjunto S^* de tamanho mínimo, tal que $S^* \subseteq S$ e $req(u) \leq |\{S_j^* \in S^* : S_j^* \ni u\}| \leq cap(u)$ para todo u de U.

Lema 3.4.1. *O problema Minimum Set Cover (ver Seção 3.3) é um caso particular do Lower-Upper Bounded Cover.*

Demonstração. Mostramos a seguir como o problema *Minimum Set Cover* (MSC) pode ser reduzido a um caso do LUBC. Considere como uma entrada para o LUBC um conjunto U com os valores $req(u) = 1$ e $cap(u) = \infty$ para todo u de U, e um conjunto S tal que $S_1 \cup \cdots \cup S_m = U$, para $m = |S|$. Considere como uma entrada para o MSC um conjunto U' e um conjunto S'. Dessa forma, o problema de encontrar o conjunto de cobertura C para este caso do MSC se torna o mesmo problema de encontrar o conjunto de cobertura S^* para este caso do LUBC: no MSC queremos encontrar $C \subseteq S'$ de tamanho mínimo tal que, $C_1 \cup \cdots \cup C_m = U'$, para $m = |C|$, que, por propriedade de conjuntos, é o mesmo que $1 \leq |C_j \in C : u \in C_j| \leq \infty$ para todo u de U'; no LUBC, queremos encontrar $S^* \subseteq S$ de tamanho mínimo tal que, para todo u de U, $req(u) \leq |\{S_j^* \in S^* : S_j^* \ni u\}| \leq cap(u)$, e a partir da entrada que foi considerada, $1 \leq |\{S_j^* \in S^* : u \in S_j^*\}| \leq \infty$, que é o mesmo objetivo do MSC. □

3.5 Heurísticas e meta-heurísticas

Uma *heurística* é um algoritmo bastante útil em problemas de otimização combinatória, que encontra uma solução viável relativamente boa, mas que não garante encontrar uma solução ótima. É desejável que uma heurística seja capaz de obter soluções de "boa qualidade", isto é, próximas de uma solução ótima, enquanto minimiza o esforço computacional (HERTZ; WIDMER, 2003; PEARL, 1984).

Uma *meta-heurística* é uma heurística estruturada e genérica, independente de um problema, que direciona ao desenvolvimento de heurísticas (HERTZ; WIDMER, 2003). Citamos como exemplos a meta-heurística BRKGA (*Biased Random-Key Genetic*

Algorithm), que é apresentada na Seção 3.7, e a meta-heurística *Local Search*, que é apresentada na Seção 3.8.

3.6 Algoritmos de aproximação

Um *algoritmo de aproximação* é um algoritmo de tempo polinomial que, para qualquer instância do problema, garante encontrar uma solução viável com um fator de até α vezes em relação ao valor de uma solução ótima. Na literatura, o fator α é comumente chamado de fator de aproximação (WILLIAMSON; SCHMOYS, 2011). Por convenção, considera-se $\alpha < 1$ em problemas de maximização, e $\alpha > 1$ em problemas de minimização. Um algoritmo de aproximação pode ser visto como uma heurística simples para a qual se provou a existência de um fator α.

3.7 Meta-heurística BRKGA

A classe de heurísticas BRKGA (Biased random-key genetic algorithm) foi introduzida em (GONÇALVES; RESENDE, 2010). O BRKGA é uma variação da classe de heurísticas RKGA (Random-key genetic algorithm), que usa o conceito tradicional em algoritmos genéticos de preservar os indivíduos de maior aptidão de uma população para as próximas gerações.

Um indivíduo é representado por um vetor de chaves aleatórias no intervalo $[0, 1]$ denominado cromossomo, que é uma solução codificada para um problema de otimização combinatória. Cada indivíduo tem seu próprio fator de aptidão, que é derivado do cálculo da função objetivo da solução que este mesmo indivíduo codifica, e é usado como critério de comparação entre indivíduos. O cálculo do fator de aptidão é intermediado por uma função decodificadora, esta que tem por papel tomar o cromossomo e convertê-lo em uma solução para o problema.

No escopo do BRKGA, uma população é um conjunto de p indivíduos. Tomemos uma geração como uma diferente versão do conjunto população. A primeira geração é formada por p indivíduos gerados aleatoriamente. Para cada geração (incluindo a primeira), em primeiro momento divide-se a população em dois grupos: um grupo que contém os p_e indivíduos com maior fator de aptidão, o qual chamamos de Elite, e um grupo com os $p - p_e$ indivíduos restantes, o qual chamamos de Não-elite.

A próxima geração é formada pelos seguintes grupos: os p_e indivíduos da classe elite da geração atual; um grupo de p_m indivíduos gerados aleatoriamente, o qual chamamos de mutantes; e por fim um grupo de $p - (p_e + p_m)$ indivíduos gerados por cruzamentos entre indivíduos do grupo elite com indivíduos do grupo não-elite, ambos selecionados aleatoriamente. Um cruzamento entre um indivíduo a pertencente a elite e um indivíduo b de não-elite é um *crossover* e gera um indivíduo c. Tal indivíduo c é gerado de forma

que o *alelo* c_i seja selecionado aleatoriamente de: a_i, com probabilidade ρ; ou b_i, com probabilidade $1 - \rho$.

Após várias gerações é esperado que os indivíduos da última geração possuam melhores fatores de aptidão do que os indivíduos das primeiras gerações e, por fim, basta que a única solução a ser considerada seja uma das soluções de melhor fator de aptidão da última geração.

3.8 Meta-heurística *Local Search*

Seja um conjunto $\mathcal{S}$ de todas as soluções viáveis para um problema, uma função custo $f : \mathcal{S} \to \mathbb{R}$ e uma estrutura de vizinhança $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$. Definimos um mínimo local como uma solução s de $\mathcal{S}$ em que $f(s) \leq f(s')$ para todo s' de $\mathcal{N}(s)$. Em um problema de minimização, uma Busca Local (*Local Search*) é um procedimento que parte de uma solução inicial s de $\mathcal{S}$ e navega entre soluções, passando sempre de uma solução s' a uma outra solução s'' vizinha a s' através de um movimento m, procurando minimizar o custo da solução final, até encontrar um mínimo local (ARIYA et al., 2004). Uma busca local genérica é mostrada no Algoritmo 1.

Algoritmo 1: *Local Search.*

Saída: a solução ótima local s'
início
  $s' \leftarrow$ uma solução arbitrária em $\mathcal{S}$
  enquanto existir s'' **em** $\mathcal{N}(s')$ **tal que** $c(s'') < c(s')$ **faça**
    $s' \leftarrow s''$
  fim
fim

4 O PROBLEMA *LEASING k-CENTER*

Este capítulo apresenta conceitos, definições e algoritmos específicos para o *leasing k-center*.

4.1 Descrição formal

Uma descrição formal para o *leasing k-center* é dada da seguinte maneira. Sejam (V, d) um espaço métrico e $T = \{0, \ldots, t_{max}\}$ o conjunto dos instantes de tempo, onde t_{max} é o instante máximo considerado. Os clientes que precisam ser servidos no instante t, para t em T, são agrupados no subconjunto D_t de V. Seja $\mathcal{D} = \{D_0, \ldots, D_{t_{max}}\}$ uma coleção de clientes a serem atendidos. Sejam $L = \{1, \ldots, l_{max}\}$ o conjunto dos tipos de facilidades, onde l_{max} é o último tipo considerado, e $\Delta = \{\delta_1, \ldots, \delta_{l_{max}}\}$ o conjunto de possíveis durações para a abertura de uma facilidade, de modo que uma facilidade do tipo l aberta no instante t permanece ativa durante o intervalo fechado $[t, t + \delta_l)$.

O objetivo do problema *leasing k-center* é encontrar um conjunto $\mathcal{M}$ em $V \times L \times T$ tal que $|\{(i, l, t') \in \mathcal{M} : t \in [t', t' + \delta_l)\}| \leq k$ para todo t em T, e que minimiza a distância máxima entre um cliente e a facilidade ativa mais próxima, dada por

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

4.2 Formulação

A formulação de programação linear inteira mista do *leasing k-center* é uma adaptação da formulação de (DASKIN, 1995) para o *k-center*. Tal formulação foi proposta por nós em (SANTOS et al., 2019).

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. A variável z representa a distância máxima entre um cliente ativo e uma facilidade ativa mais próxima. Por simplicidade, considere p_l^t o menor período em que uma possível facilidade de tipo l aberta no período t pode ter iniciado sua abertura, dado por $p_l^t = t - \delta_l + 1$, se $\delta_l \leq t$; ou $p_l^t = 1$, caso contrário.

$$\text{minimizar} \quad z \tag{1}$$

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

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

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

$$\sum_{i \in V} x_{ij}^{t} d(i,j) \leq z \qquad \forall t \in T,\ \forall j \in D_t \tag{5}$$

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

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

$$z \in \mathbb{R}$$

A função objetivo (1) minimiza a distância máxima entre cada cliente e a 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. As restrições em (5) asseguram para cada instante t que a maior distância entre um cliente j em D_t e a facilidade ativa mais próxima seja no máximo z.

4.3 Heurística BRKGA para o LKC

A heurística descrita nesta seção foi proposta como parte deste mesmo trabalho em (SANTOS et al., 2019). Esta heurística utiliza a meta-heurística BRKGA conforme descrita na Seção 3.7 para gerar os indivíduos, e um algoritmo decodificador específica para o *leasing k-center* que avalia tais indivíduos. Esta função decodificadora é uma heurística que devolve um valor real: o custo da solução associada a um indivíduo da entrada. Este valor também é utilizado como o fator de aptidão do indivíduo da entrada.

Um cromossomo é modelado como uma matriz de chaves aleatórias A de dimensões $|T| \times |V|$ onde $0 \leq A_{ti} \leq 1$. A função decodificadora converte a matriz A em uma solução viável $\mathcal{M}$ por associar cada elemento A_{ti} à possibilidade de abertura da facilidade i no instante t. De antemão é calculado um valor ρ que define a probabilidade de um valor A_{ti} corresponder a abertura de uma facilidade. Dessa forma a chave aleatória A_{ti} é uma abertura se e somente se $A_{ti} < \rho$. O cálculo de ρ toma como base a duração média δ_μ de uma facilidade, dada por

$$\delta_\mu = \tfrac{1}{l_{max}} \sum_{l \in L} \delta_l.$$

Tomando como hipótese que toda facilidade seja do tipo de δ_μ períodos, então o número máximo de facilidades abertas possível é aproximadamente $\frac{|T|k}{\delta_\mu}$ e o valor de ρ é

$$\rho = \frac{\frac{|T|k}{\delta_\mu}}{|T||V|} = \frac{k}{\delta_\mu |V|}.$$

Se $A_{ti} < \rho$, então A_{ti} representa que a facilidade i no período t seja aberta com tipo $l = 1 + \lfloor \frac{A_{ti} * l_{max}}{v} \rfloor$. Seja a função $f(x)$ descrita a seguir:

$$f(x) = \begin{cases} 1 + \lfloor \frac{x * l_{max}}{\rho} \rfloor, \text{ se } x < \rho, \\ 0, \text{ caso contrário.} \end{cases}$$

Seja também *ativas* um vetor de dimensão $|T|$ que inicialmente possui todos os valores igual a zero. O algoritmo decodificador é mostrado no Algoritmo 2.

Algoritmo 2: Decodificador.

Entrada: matriz de chaves aleatórias A
Saída: o valor da solução $\mathcal{M}$
início
  $l_{min} = arg\min_{l \in L} \delta_l$
  $\mathcal{M} \leftarrow \emptyset$
  para $t \leftarrow 0$ **até** t_{max} **faça**
    para $i \leftarrow 1$ **até** $|V|$ **faça**
      $l \leftarrow f(A_{ti})$
      se $l > 0$ **e** $ativas(t) < k$ **então**
        $\mathcal{M} = \mathcal{M} + (i, l, t)$;
        para $o \leftarrow t$ **até** $min(t + \delta_l - 1, t_{max})$ **faça**
          $ativas(o) \leftarrow ativas(o) + 1$
        fim
      fim
    fim
    se $ativas(t) = 0$ **então**
      $i^t_{min} = arg\min_{i \in V} \max_{j \in D_t} d(i, j)$;
      $\mathcal{M} = \mathcal{M} + (i^t_{min}, l_{min}, t)$;
      para $o \leftarrow t$ **até** $min(t + l_{min} - 1, t_{max})$ **faça**
        $ativas(o) \leftarrow ativas(o) + 1$
      fim
    fim
  fim
fim

A função decodificadora itera entre todas as $|T|$ linhas da matriz A. Em cada iteração as $|V|$ colunas são percorridas fazendo sucessivas consultas ao vetor *ativas* e,

nos casos onde há clientes não servidos, é realizado cálculo de i^t_{min}, que possui custo $O(|V|^2)$ para cada t, seguido por uma atualização do vetor *ativas*. Cada valor i^t_{min} pode ser pré-processado antes mesmo do início da execução do BRKGA, para evitar cálculo desnecessário. Cada consulta e atualização ao vetor *ativas* pode ser realizada em tempo $O(log|T|)$ utilizando uma estrutura de dados como *Segment Tree*. Cada adição de uma nova facilidade em $\mathcal{M}$ é feita em $\Theta(1)$, entretanto, após o término das iterações é necessário verificar o valor da solução $\mathcal{M}$, uma tarefa que possui complexidade $O(|T| \times |V|^2)$. O algoritmo implementado dessa forma possui complexidade $O(|T| \times |V| \times log|T| + |T| \times |V|^2)$ e demanda um pré-processamento de complexidade $O(|V|^2)$.

4.4 Heurística LS-LUBC para o LKC

Nesta seção propomos a heurística *Local Search in Lower-Upper Bounded Cover* para encontrar soluções viáveis ao *leasing k-center*, que é semelhante ao algoritmo proposto em (MINIEKA, 1970) para o *k-center*. Este procedimento toma por um diferente conjunto de cobertura a união dos seguintes itens: uma abertura de facilidade; os clientes ativos que seriam servidos caso tal abertura ocorra e que estão a uma distância para com a localidade de tal facilidade não maior do que um limiar; e elementos que garantem a restrição de não haver mais do que k facilidades abertas em um mesmo período. Basicamente, o procedimento consiste em uma busca pelo menor limiar que, quando utilizado para construir os conjuntos de cobertura, permite gerar um problema de LUBC que pode ser resolvido por uma heurística *Local Search*, devolvendo uma solução $\mathcal{M}$. Detalhamos o procedimento a seguir. Os algoritmos não mostrados nesta Seção se encontram no Apêndice A.

Tomemos um elemento $u = (i, l, t)$, para u em $V \times L \times T$, como a abertura de uma facilidade em i de tipo l no período t. Tomemos um elemento $u = (f, t)$, para u em $D_t \times T$, como o cliente ativo D^t_f.

A primeira etapa do procedimento é a criação dos conjuntos C, U, S e S'. O conjunto C é a união dos valores de distâncias $C = \{d(u, v) : u, v \in V\}$. A complexidade de espaço deste conjunto é $\Theta(|V|^2)$.

O conjunto universo U é constituído por elementos x de $V \times L \times T + D_t \times T + T$. Os elementos de U são particionados em períodos $T_0, \ldots, T_{t_{max}}$, e cada elemento se enquadra em uma das seguintes classes: virtual (ε), localidade (V) ou cliente ativo (D). Um elemento virtual x é oriundo de T, possui demanda zero e capacidade k. Um elemento localidade x é oriundo de $V \times L \times T$, possui demanda zero, capacidade infinita, e corresponde ao vértice *pai* de algum conjunto de cobertura. Um elemento cliente ativo x é oriundo de $D_t \times T$, possui demanda um e capacidade infinita. A função de construção de U é mostrado no Algoritmo 4. A complexidade de tempo da função, assim como a complexidade de espaço do conjunto U é $O(|V| \times |L| \times |T|)$.

Tomemos como exemplo para o restante desta Seção o seguinte cenário em que há quatro localidades e cujas distâncias entre si são mostradas no grafo completo da Figura 5. Neste cenário são considerados três períodos de tempo, aluguéis dos tipos dois ou três períodos, e que os clientes ativos são distribuídos no tempo da seguinte forma: $D_0 = \{V_2, V_4\}$, $D_1 = \{V_1, V_2, V_3, V_4\}$ e $D_2 = \{V_1, V_4\}$. Para este cenário o conjunto U a ser criado é ilustrado na Figura 6, no qual podemos observar as partições T_0, T_1 e T_2, e também a separação dos elementos pelos grupos ε, V e D.

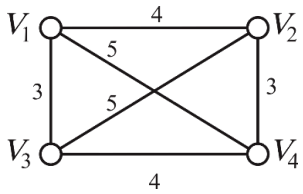

Figura 5 – Exemplo de grafo de localidades.

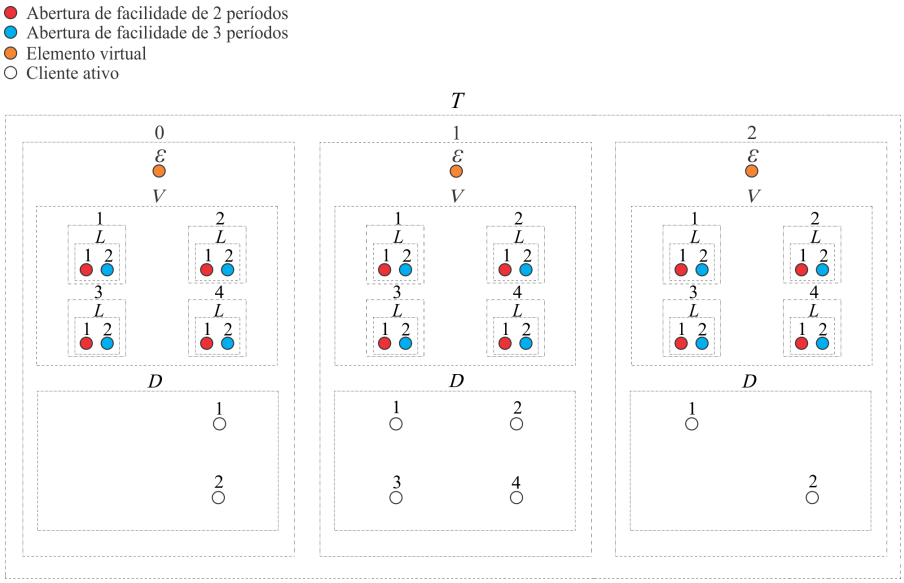

Figura 6 – Exemplo de representação do conjunto U.

A coleção S contém todos os conjuntos de cobertura. Cada conjunto de cobertura possui apenas um elemento *pai*. A coleção S é construída contendo, em cada conjunto de cobertura, apenas o elemento *pai*, conforme a função mostrada no Algoritmo 5. Os elementos do grupo ε não são adicionados nos conjuntos de cobertura de S, mas são facilmente gerados conhecendo apenas o elemento *pai*. Os elementos do grupo D são adicionados através de outra função. A complexidade de tempo da função é $O(|V| \times |L| \times |T|)$.

O conjunto S' é a união das associações entre aberturas de facilidades u e clientes ativos v, onde cada associação é uma tupla (u, v, w), para u em $V \times L \times T$, v em $D_t \times T$ e w em $\mathbb{R}$. A função de construção de S' é mostrada no Algoritmo 6. A complexidade de tempo da função, assim como a coplexidade de memória do conjunto S' é $O(|V|^2 \times |T|^2 \times |L|)$.

Uma solução viável $\mathcal{M}'$ é uma solução cujo valor de penalidade é zero. Apesar de um elemento $\mathcal{M}'_s$ representar um conjunto de cobertura s, apenas o valor $pai(s)$ é armazenado. A cada adição ou remoção de uma cobertura em $\mathcal{M}'$ os elementos globais de contagem $\mathcal{C} = (D, \varepsilon)$, σ e τ, para $\mathcal{C}_i^D$, $\mathcal{C}_i^\varepsilon$, σ e τ em $\mathbb{Z}$, são alterados para serem utilizados no cálculo da penalidade. As funções de adicionar e remover um conjunto de cobertura s em $\mathcal{M}'$ são mostradas através dos Algoritmos 7 e 8, respectivamente. Quando um conjunto de cobertura s é adicionado ou removido em $\mathcal{M}'$, o elemento $pai(s)$ é adicionado ou removido em $\mathcal{M}'$, enquanto que os outros elementos de s alteram apenas os elementos de contagem. A função $pen+(s, \mathcal{M}')$ calcula o valor de penalidade caso uma cobertura s seja adicionada a $\mathcal{M}'$, e é mostrada pelo Algoritmo 9. A função $pen\text{-}(s, \mathcal{M}')$ calcula o valor de penalidade caso uma cobertura seja removida de $\mathcal{M}'$, que é similar à função $pen+$, mas com as operações *adicionar* e *remover* invertidas. A função $pen+\text{-}(s', s'', \mathcal{M}')$ calcula o valor de penalidade caso a cobertura s' seja adicionada e a cobertura s'' removida de $\mathcal{M}'$ ao mesmo tempo. Todas as funções *pen* utilizam duas constantes α e β. As funções *adicionar*, *remover* e *pen* possuem complexidade $O(|V| + |T|)$.

O procedimento utiliza outras três funções: redução ao *Lower-Upper Bounded Cover* (*reduçãoLUBC*), solução inicial (*solInicial*) e refinamento (*solRefinada*).

A função *reduçãoLUBC* adiciona outros elementos a conjuntos de cobertura de S. Um parâmetro especial desta função é um valor c de C, que restringe quais elementos serão adicionados a S. A função é descrita no Algoritmo 10.

A partir do cenário anterior e considerando, por simplicidade, apenas os conjuntos de cobertura que possuem como *pai* as aberturas de facilidades (V_1, L_0, T_0) e (V_1, L_1, T_0), quando $c = 3$ a função retorna os conjuntos que são ilustrados na Figura 7 e, quando $c = 5$ retorna os conjuntos ilustrados na Figura 8. Um conjunto de cobertura é ilustrado como uma rede interligada por linhas de uma determinada cor, tendo o elemento *pai* como o centróide.

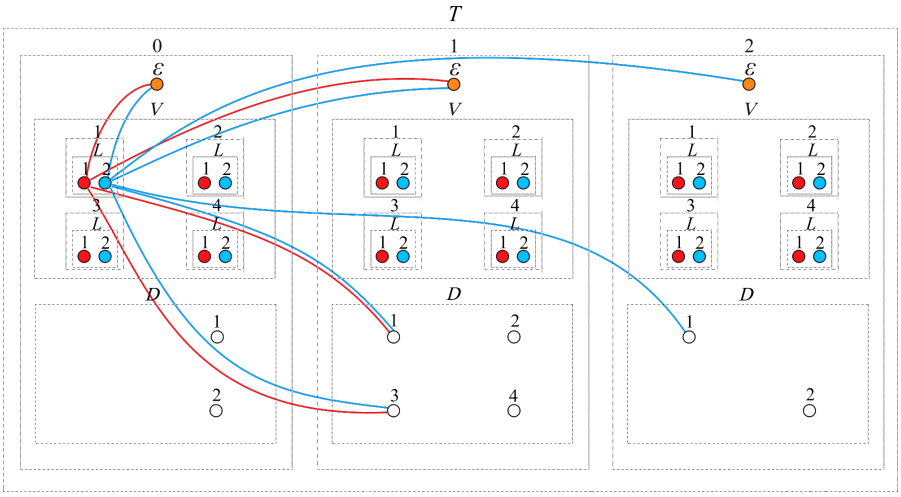

Figura 7 – Conjunto S para o cenário de exemplo com $c = 3$.

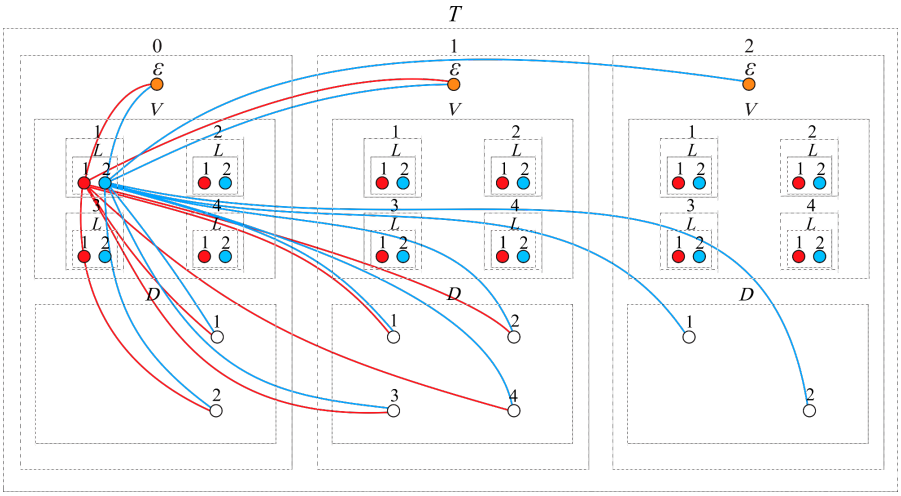

Figura 8 – Conjunto S para o cenário de exemplo com $c = 5$.

A complexidade de tempo da função *reduçãoLUBC* é feita por análise amortizada. Esta função é chamada até que todos os valores c de C tenham sido verificados, isto é, é chamada $O(|V|^2)$ vezes. Ao final da última execução o número de instruções já realizadas é $O(|S'|)$. Desta forma, por análise amortizada, o custo da função *reduçãoLUBC* é

$$O(\Pi) : \Pi = \frac{|S'|}{|V|^2} = \frac{|V|^2 \times |T|^2 \times |L|}{|V|^2} = |T|^2 \times |L|$$

O objetivo da função *solInicial* é gerar uma solução (possivelmente não viável) $\mathcal{M}_{ini}$, de tal forma que pelo menos a restrição de demanda seja satisfeita para todo u de U. A função consiste em um laço que, para cada u de U, adiciona a $\mathcal{M}$ o *pai* de um conjunto s de S, onde u esteja em s, se a restrição de demanda de u ainda não tiver sido atendida. Caso haja mais do que um conjunto s possível é escolhido aquele que minimiza o valor de $pen+(s)$, neste caso considerando $\beta = 0$, pois estamos interessados em apenas satisfazer a restrição de demanda. Os elementos u de U oriundos de partições inferiores (de menores índices) são verificados primeiro em relação aos elementos de períodos de tempo superiores. O algoritmo 11 mostra a função.

O comportamento da função *solInicial* é ilustrado nas Figuras de 9 a 11. Para o cenário ilustrado é fornecido o valor de cobertura $c = 3$. Conforme o Algoritmo 11, os elementos não servidos oriundos de partições inferiores são verificados primeiro. Cada elemento não servido é colorido de preto, cada elemento servido é colorido de branco e cada elemento sendo verificado é destacado por uma circunferência vermelha. As figuras mostram o estado da função em cada iteração na mesma ordem em que são mostradas e a Figura 11 é o estado final.

O segundo laço de repetição da função *solInicial* acessa apenas os elementos de U do grupo D em D_t. Cada grupo D_t possui no máximo V elementos e, por isso, o segundo laço realiza $O(|V|)$ iterações. A coleção S possui, no máximo, $|V| \times |L| \times |T|$ conjuntos de cobertura. Verificar se um cliente ativo u está em um conjunto de cobertura s pode ser feito em $O(1)$ conhecendo apenas $pai(s)$. As funções $pen+$ e *adicionar* possuem

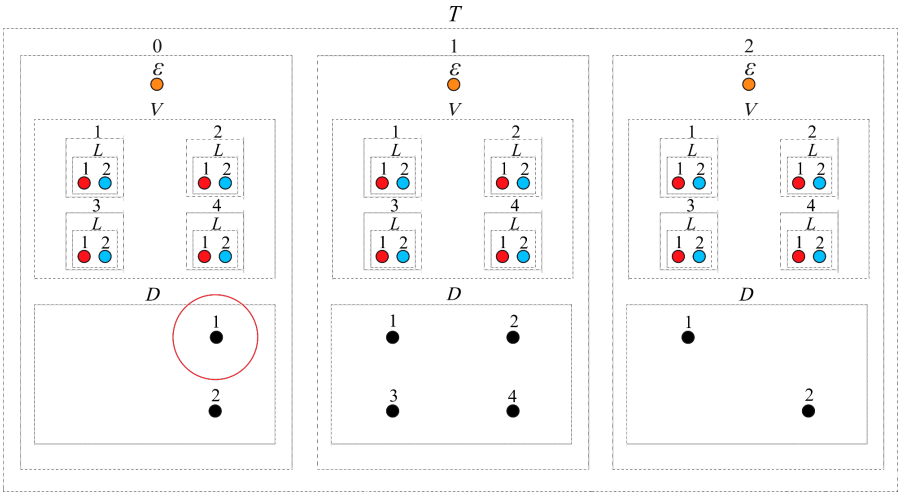

Figura 9 – Primeira iteração da função *solInicial* no cenário de exemplo.

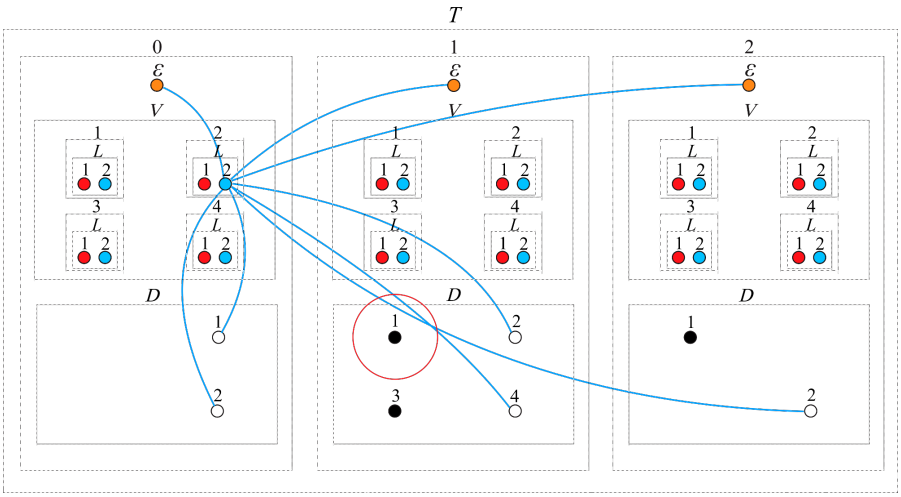

Figura 10 – Segunda iteração da função *solInicial* no cenário de exemplo.

complexidade $O(|V| + |T|)$, conforme discutido anteriormente, mas como neste caso não estamos interessados em satisfazer a restrição de capacidade, então o cálculo de *pen+* pode ser realizado em $O(|V|)$. Quando implementada dessa forma, esta função possui complexidade $O(|T| \times |V| \times [\{|V| \times |L| \times |T|\} \times |V| + \{|V| + |T|\}]) \in O(|V|^3 \times |T|^2 \times |L|)$.

A função *solRefinada* utiliza a meta-heurística *Local Search* para navegar entre soluções, partindo da solução $\mathcal{M}_{ini}$ e até alcançar uma solução ótima localmente $\mathcal{M}_{ref}$. Para este contexto, a vizinhança de uma solução $\mathcal{M}'$ é um dos seguintes casos:

1. $\mathcal{M}' + s'$, onde $s' \notin \mathcal{M}'$;
2. $\mathcal{M}'$ - s'', onde $s'' \in \mathcal{M}'$;
3. $\mathcal{M}' + s'$ - s'', onde $s' \notin \mathcal{M}'$ e $s'' \in \mathcal{M}'$.

Normalmente a vizinhança em 2 é muito menor do que a vizinhança em 1, que é muito menor do que a vizinhança em 3. Por este motivo, a função busca primeiro a solução de menor penalidade da vizinhança em 2. Se tal solução possuir penalidade menor do

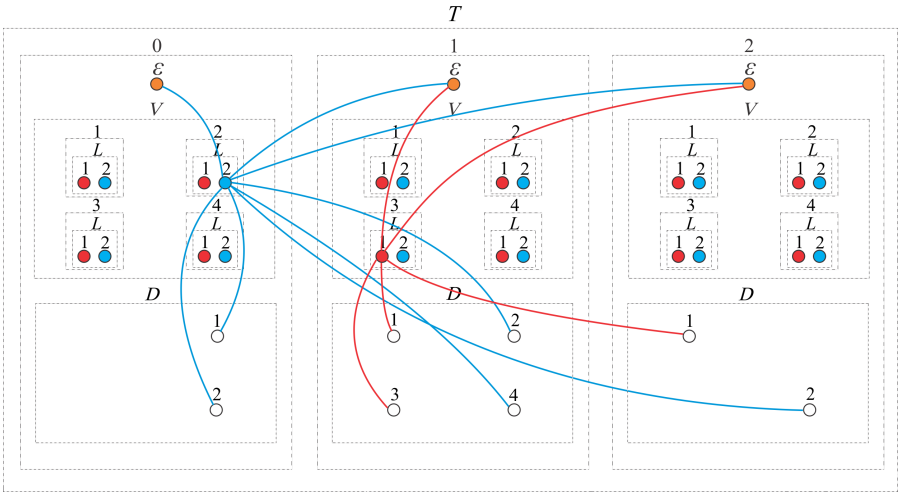

Figura 11 – Estado final da função *solInicial* no cenário de exemplo.

que a solução corrente esta é a próxima solução vizinha a se explorar. Caso contrário o mesmo é feito para a vizinhança em 1. Se ainda assim não for encontrada uma solução de penalidade menor, a vizinhança em 3 é verificada e a primeira solução de penalidade menor do que a solução corrente é escolhida como a próxima a ser explorada. Caso não tenha sido encontrada tal solução, então a função não convergiu e termina sua execução.

Outros dois critérios também são utilizados para terminar a execução: limite de tempo ou limite de iterações. O limite de tempo se baseia em uma constante $limT$ fornecida pelo programador. O limite de iterações se baseia em uma constante $limI$ fornecida pelo programador. O número de iterações de uma chamada da função sempre é estimado com base nas penalidades das quatro últimas soluções exploradas até a iteração corrente, quando houver. Tais penalidades são mantidas em uma fila *solAnteriores*. Caso o número estimado seja maior do que o limite de iterações, a função também para. A função devolve a solução de menor penalidade encontrada até o término. Seu funcionamento é mostrado através do Algoritmo 12.

O tempo levado para a execução da função *solRefinada* é geralmente limitado pela constante $limT$ mas, no pior caso, a saber, quando o tempo limite é atingido mesmo quando as vizinhanças de M ainda são verificadas, tal verificação deverá ser concluída para então terminar a execução. No pior caso a vizinhança 3 é visitada, que é a maior das vizinhanças. O custo para verificar todas as aberturas de facilidade em $\mathcal{M}_{ref}$ é $O(|V| \times |T|)$, que pode ocorrer quando $k = |V|$ e há um $\delta_l = 1$, pois cada V_i poderia ser uma abertura de facilidade do tipo $\delta_l = 1$ em cada período t de T. O custo para verificar as soluções em $\mathcal{M}_{not}$ é $O(|V| \times |L| \times |T|)$. As complexidades das funções *pen+*, *pen-* e *pen+-* são $O(|V| + |T|)$, conforme discutido anteriormente. Dessa forma, a complexidade de tempo da verificação da vizinhança 3 é $O(\{|V|\times|T|\}\times\{|V|\times|L|\times|T|\}\times\{|V|+|T|\}) = O(|V|^2\times|T|^2\times|L|\times\{|V|+|T|\})$. A complexidade desta função, considerando que $I = limT \times o$ seja o número de operações computacionais realizadas em $limT$ segundos, é $O(I + |V|^2 \times |T|^2 \times |L| \times \{|V| + |T|\})$.

O procedimento LS-LUBC é descrito no Algoritmo 3.

Algoritmo 3: Heurística LS-LUBC.

Saída: o custo c da solução viável $\mathcal{M}$
início
  $C \leftarrow \{d(u,v) : u,v \in V\}$
  $U \leftarrow \text{Construção}U()$
  $S \leftarrow \text{inicializa}S(V, t_{max}, \Delta, S', c)$
  $S' \leftarrow \text{Construção}S'()$
  $\sigma' \leftarrow \sum_{t \in T} |D_t|$
  $c \leftarrow \min(C)$;
  $solução \leftarrow -1$;
  enquanto $solução = -1$ **faça**
    $reduçãoLUBC(c)$;
    $\mathcal{C} \leftarrow (0,0)$
    $\sigma \leftarrow \sigma'$
    $\tau \leftarrow 0$
    $\mathcal{M}_{ini} \leftarrow solInicial()$;
    $\mathcal{M}_{ref} \leftarrow solRefinada(\mathcal{M}_{ini})$;
    se $pen+(\emptyset, \mathcal{M}_{ref}) = 0$ **então**
      $\mathcal{M} \leftarrow \mathcal{M}_{ref}$;
      $solução \leftarrow c$;
    fim
    $C \leftarrow C - c$;
    $c \leftarrow \min(C)$;
  fim
fim

A soma das complexidades das funções *reduçãoLUBC*, *solInicial* e *solRefinada* é:

$$\begin{aligned}\Phi &\in O(|T|^2 \times |L|) + O(|V|^3 \times |T|^2 \times |L|) + O(I + |V|^2 \times |T|^2 \times |L| \times \{|V| + |T|\}) \\ &\in O(I + |T|^2 \times |L| \times \{1 + 2|V|^3 + |V|^2 \times |T|\}) \\ &\in O(I + |V|^2 \times |T|^2 \times |L| \times \{|V| + |T|\})\end{aligned}$$

A soma das complexidades das funções de criação dos conjuntos C, U, S e S' é:

$$\begin{aligned}\Psi &\in O(|V|^2) + O(|V| \times |L| \times |T|) + O(|V| \times |L| \times |T|) + O(|V|^2 \times |T|^2 \times |L|) \\ &\in O(|V|^2 \times |T|^2 \times |L|)\end{aligned}$$

Por fim, a complexidade do procedimento LS-LUBC é:

$$\begin{aligned}&O(\Psi + |V|^2 \times \Phi) \\ \in\ &O(|V|^2 \times |T|^2 \times |L| + |V|^2 \times \{I + |V|^2 \times |T|^2 \times |L| \times \{|V| + |T|\}\}) \\ \in\ &O(|V|^2 \times I + |V|^4 \times |T|^2 \times |L| \times \{|V| + |T|\})\end{aligned}$$

Experimentos mostram que, apesar da complexidade computacionalmente inviável, a heurística alcançou, em instâncias de tamanho razoáveis de caso médio, soluções boas em tempo razoável.

5 EXPERIMENTOS COMPUTACIONAIS

Neste capítulo comparamos experimentalmente os resultados apresentados pela formulação de PLIM quando utilizada em um resolvedor e os resultados apresentados por cada uma das heurísticas do Capítulo 4.1.

A formulação de PLIM foi utilizada no resolvedor comercial GUROBI, versão 8.1.0. O resolvedor GUROBI implementa heurísticas próprias e métodos exatos. A primeira solução apresentada por tal resolvedor é uma heurística, em seguida o algoritmo realiza iterações até encontrar uma solução ótima. Nos casos em que uma solução ótima não foi encontrada dentro de um prazo foi considerada apenas a última solução viável encontrada pelo resolvedor.

Através dos experimentos foi observado que o resolvedor demandava um menor custo computacional, em relação a outras configurações testadas, quando configurado para utilizar apenas o método *Simplex Dual*. Dessa forma todos os experimentos foram realizados apenas com essa configuração. Não foram configuradas outras variáveis globais.

O algoritmo decodificador do BRKGA foi implementado conforme é descrito na Seção 4.3 e, para os demais procedimentos do BRKGA, foi utilizado o *framework* em C++ proposto por (TOSO; RESENDE, 2015). As variáveis globais do BRKGA foram definidas automaticamente pelo *irace* (IBÁNEZ et al., 2011), um software designado para tunagem de parâmetros. A configuração de melhor desempenho encontrada pelo *irace* é mostrada na Tabela 1.

Variável	Descrição	Valor
p	tamanho da população	943
p_e	tamanho da partição elite	322
p_m	tamanho da partição não-elite	90
ρ	probabilidade de um indivíduo herdar um alelo do pai elite	55,1%
K	número de populações independentes	1
MAXT	número de threads para decodificação paralela	2
X_INTVL	intervalo para haver trocas de melhores indivíduos	298
X_NUMBER	número de indivíduos envolvidos em cada troca	3

Tabela 1 – Variáveis para a heurística de BRKGA

A semente para o gerador de números aleatórios não foi configurada como um parâmetro a ser gerado pelo *irace* mas foi fixada como zero. Isso também possibilita que os experimentos aqui realizados sejam replicados.

Para a heurística LS-LUBC foram implementados procedimentos conforme a Seção 4.4. As variáveis globais foram definidas conforme a Tabela 2.

Variável	Descrição	Valor
α	pontos de penalidade para cada requisição não atendida	1
β	pontos de penalidade para cada capacidade ultrapassada	2
$limI$	limite de iterações para a Busca Local	40
$limT$	limite de tempo para a Busca Local	60 segundos

Tabela 2 – Variáveis globais para a heurística LS-LUBC

Esta heurística está sujeita a nem sequer apresentar uma solução viável dentro de um prazo. Entretanto, não houve tal caso durante os experimentos aqui realizados.

5.1 Instâncias

Algumas das instâncias utilizadas são adaptações de instâncias para o problema *k-median* da base *OR-Library* publicada em (BEASLEY, 1990). As outras instâncias foram criadas aleatoriamente. A partir de experimentos foi possível particionar as instâncias em três categorias:

1. foi possível a execução em cada um dos algoritmos e o resolvedor GUROBI foi capaz de encontrar a solução ótima dentro de um prazo;
2. foi possível em cada um dos algoritmos, mas o resolvedor GUROBI não apresentou uma solução ótima dentro de um prazo e por isso foi considerada a melhor solução viável fornecida;
3. tanto pelo resolvedor GUROBI quanto pela heurística LS-LUBC não foi possível a execução devido a limitações de memória da máquina de teste.

A Tabela 3 contém as informações básicas a respeito das instâncias usadas. A coluna *Id* é a identificação da instância conforme é referenciada ao longo deste capítulo. A coluna *Cat* indica a qual das categorias mencionadas anteriormente a instância pertence. A coluna *Origem* informa se a instância foi adaptada da *OR-Library* ou se foi gerada aleatoriamente. A coluna *cliAt* contém o valor de

$$cliAt = \sum_{t \in T} |D_t|.$$

As demais colunas representam as informações nomeadas conforme a Seção 4.1.

Informações gerais			Tamanho da instancia					
Id.	Cat.	Origem	$\|V\|$	$\|T\|$	$\|L\|$	k	$\|V\| \times \|T\| \times \|L\|$	$cliAt$
1	1	adaptada	100	7	4	5	2800	420
2	1	adaptada	100	10	2	5	2000	491
3	1	adaptada	100	1	1	5	100	53
4	1	adaptada	100	2	2	5	400	90
5	1	adaptada	100	4	2	5	800	125
6	1	adaptada	100	12	3	10	3600	725
7	1	adaptada	100	8	5	10	4000	262
8	1	adaptada	100	5	1	10	500	193
9	1	adaptada	100	15	1	10	1500	958
10	1	adaptada	100	11	2	10	2200	479
11	2	aleatória	170	25	8	15	34000	1923
12	2	aleatória	170	36	5	27	30600	2990
13	2	aleatória	172	33	8	13	45408	3037
14	2	aleatória	183	25	12	9	54900	2049
15	2	aleatória	127	31	18	14	70866	1963
16	2	aleatória	193	19	11	8	40337	2069
17	2	aleatória	201	16	8	12	25728	1427
18	2	aleatória	230	8	4	9	7360	1104
19	2	adaptada	200	18	10	10	36000	2029
20	2	adaptada	200	17	7	10	23800	1440
21	3	aleatória	218	86	23	25	431204	8751
22	3	aleatória	268	133	9	23	320796	19250
23	3	aleatória	252	187	1	76	47124	22852
24	3	aleatória	322	31	8	8	79856	4546
25	3	aleatória	312	141	10	16	439920	22980
26	3	aleatória	348	152	30	19	1586880	25315
27	3	aleatória	333	168	16	69	895104	28531
28	3	aleatória	493	50	7	9	172550	13268
29	3	aleatória	451	153	12	68	828036	33875
30	3	aleatória	500	159	23	29	1828500	41368

Tabela 3 – Quadro de instâncias

Como apenas a heurística de BRKGA pôde executar as instâncias da categoria 3 na máquina de teste, então apenas os resultados dos experimentos com as instâncias das categorias 1 e 2 são considerados ao longo deste capítulo.

5.2 Comparação dos resultados

Foi escolhida como máquina de teste um computador de 4GB de memória RAM e processador pentium dual core 2,60 GHz sob um sistema operacional Linux versão 16.04. Todas as implementações foram feitas em C++. Na máquina de teste, cada experimento foi limitado a um prazo de vinte e cinco minutos e foi considerada apenas a melhor solução encontrada nesse período.

Para todos os experimentos com a categoria de instâncias 1 o resolvedor GUROBI encontrou soluções ótimas dentro do prazo de 25 minutos. Dessa forma o valor de cada solução ótima foi utilizado como referência na comparação através do cálculo do GAP entre os valores das soluções dos demais algoritmos. Para todos os experimentos com a categoria de instâncias 2 o GUROBI não apresentou a solução ótima dentro do prazo, mas também foi observado que o GUROBI não apresentou soluções incumbentes além das soluções geradas por heurísticas. Nas instâncias 12 e 15 o GUROBI finalizou a etapa de relaxação de raízes antes do prazo e um limitante inferior pôde ser devolvido, enquanto que o mesmo não ocorreu para as outras instâncias da categoria 2. Dessa forma apenas os valores das

soluções geradas pelas heurísticas do GUROBI foram utilizados como parâmetros para o cálculo do GAP.

A Tabela 4 mostra a comparação dos resultados para experimentos com instâncias da categoria 1 e a Tabela 5 para os experimentos com instâncias da categoria 2. Em cada tabela, para cada algoritmo há uma coluna *valor* que contém o valor da solução encontrada. Para as heurísticas BRKGA e LS-LUBC há também uma coluna adicional GAP, que é um cálculo obtido através de:

$$\text{GAP} = \frac{|\text{valorALGORITMO - valorGUROBI}|}{\text{valorALGORITMO}} \times 100\%.$$

Para cada um dos algoritmos também há uma coluna *tempo*. Esta coluna informa o tempo (em segundos) que o algoritmo levou para apresentar a solução cujo valor é mostrado na tabela. Na tabela dos resultados para experimentos da segunda categoria há uma coluna adicional ao GUROBI que informa, quando há, o limitante inferior que foi devolvido imediatamente após a etapa de relaxação de raizes.

	GUROBI		BRKGA			LS-LUBC		
Id.	valor	tempo	valor	GAP	tempo	valor	GAP	tempo
1	116	306	127	8,661%	27	127	8,661%	34
2	127	1252	133	4,511%	12	128	0,781%	28
3	112	2	112	0%	333	123	8,943%	1
4	112	8	116	3,448%	1	116	3,448%	2
5	115	19	115	0%	403	125	8%	4
6	98	855	108	9,259%	41	102	3,922%	77
7	96	72	102	5,822%	105	103	6,796%	34
8	95	36	100	5%	47	101	8,654%	11
9	98	712	111	11,712%	315	106	7,547%	85
10	98	496	109	10,092%	110	128	23,438%	48

Tabela 4 – Valores das soluções para experimentos da categoria 1.

	GUROBI			BRKGA			LS-LUBC		
Id.	LimInf	valor	tempo	valor	GAP	tempo	valor	GAP	tempo
11	-	442	48	236	87,288%	1367	226	95,575%	288
12	99,393	238	125	212	12,264%	1417	198	16,667%	866
13	-	408	185	240	70%	843	242	67,213%	558
14	-	392	188	256	53,125%	562	250	51,938%	1274
15	134,13	460	224	234	96,581%	392	230	100%	540
16	-	380	108	256	48,438%	242	240	47,287%	938
17	-	300	26	234	28,205%	949	220	36,364%	401
18	-	378	13	240	57,5%	298	236	60,169%	137
19	-	97	184	78	24,359%	206	67	44,776%	845
20	-	97	49	69	40,58%	363	68	42,647%	466

Tabela 5 – Valores das soluções para experimentos da categoria 2.

Para as instâncias da categoria 1, o GAP médio para a heurística de BRKGA foi $\approx 6,562\%$, enquanto que para a heurística LS-LUBC foi $\approx 8,1\%$. O tempo médio para a execução do GUROBI foi $\approx$ 376s, para a heurística de BRKGA foi $\approx$ 140s e para a heurística de LS-LUBC foi $\approx$ 32s.

Para as instâncias da categoria 2, todas as soluções devolvidas pelas heurísticas de BRKGA e LS-LUBC foram melhores do que as soluções devolvidas pelo GUROBI. O valor GAP neste caso é semanticamente invertido. O GAP médio para a heurística BRKGA

foi $\approx 51,833\%$, enquanto que para a heurística LS-LUBC foi $\approx 58,346\%$. O tempo médio para a execução do GUROBI foi ≈ 115s, para a heurística de BRKGA foi ≈ 664s e para a heurística de LS-LUBC foi ≈ 631s. A seguir observamos como as soluções geradas pelos algoritmos melhoraram ao longo do tempo.

A Figura 12 mostra o desempenho do GUROBI para as instâncias da categoria 1. As primeiras soluções são geradas a partir de heurísticas. É possível observar, através dos gráficos, que tais soluções apresentam uma grande discrepância em relação às próximas soluções incumbentes. A partir do mesmo gráfico é possível analisar como os valores das soluções incumbentes melhoram ao longo do tempo até atingir a solução ótima. Os experimentos com as instâncias 1, 3, 8 e 10 evidenciam que, em algum momento do tempo, a magnitude das soluções se correlacionam com uma reta.

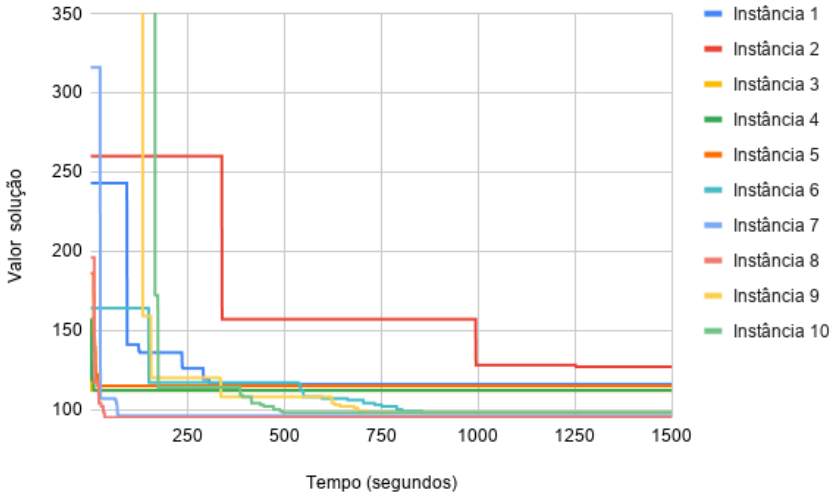

Figura 12 – Desempenho para categoria 1: GUROBI.

Em todas as instâncias da categoria 2 o GUROBI apresentou uma solução a partir de uma heurística e, ao longo dos 25 minutos, não foi encontrada outra solução incumbente. Dessa forma o gráfico de desempenho para as instâncias da categoria 2 não é mostrado uma vez que os valores permanecem estacionários ao longo do tempo.

A Figura 13 mostra o desempenho da heurística BRKGA para a categoria 1 e a Figura 14 para a categoria 2. Nestes gráficos são mostrados como o valor das melhores soluções mudam ao longo do tempo. É possível observar que, para cada um dos gráficos, o comportamento das magnitudes das soluções se correlacionam com uma função $f(1/t)$, onde t é o tempo decorrido e $f(1/t)$ o valor.

A Figura 15 mostra o desempenho da heurística LS-LUBC para as instâncias da categoria 1 e a Figura 15 para instâncias da categoria 2. Neste gráfico, diferentemente dos gráficos anteriores, é mostrado como as penalidades das soluções mudam ao longo do tempo, até que alguma solução tenha penalidade igual a zero. Podemos também utilizar tal gráfico para mostrar como as penalidades mudam à medida que o valor do raio da circunferência

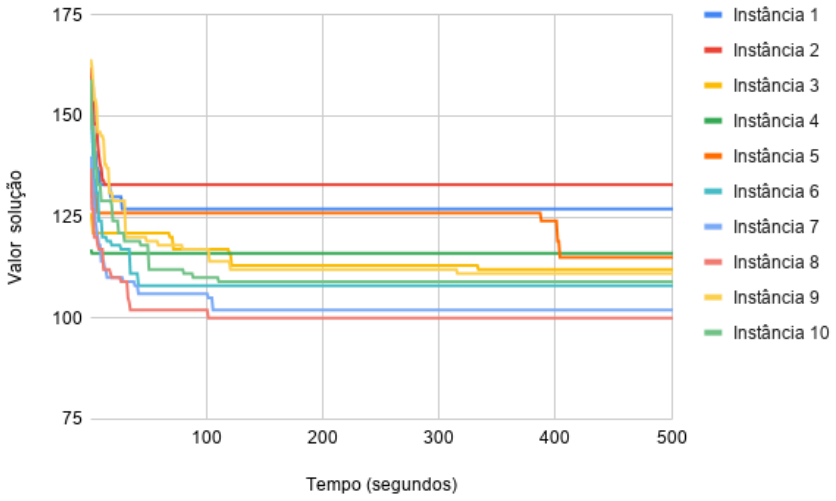

Figura 13 – Desempenho para categoria 1: heurística BRKGA.

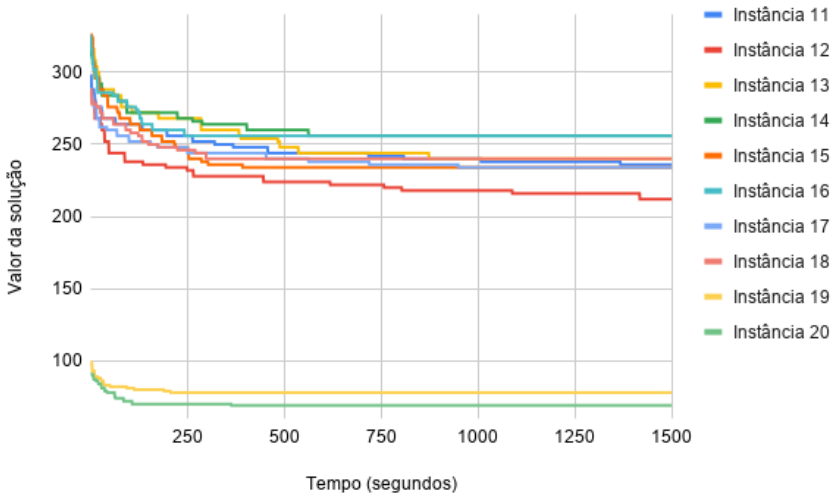

Figura 14 – Desempenho para categoria 2: heurística BRKGA.

de cobertura é incrementado. Em cada gráfico é evidenciado que as magnitudes das penalidades das soluções estão fracamente correlacionadas a uma função $f(1/t)$, onde t é o tempo decorrido e $f(1/t)$ o valor da penalidade. Essa fraca correlação impede a utilização de um algoritmo de busca na ordem $O(log(n))$, como por exemplo uma busca binária. É evidenciado também que nas primeiras soluções a magnitude das penalidades é muito alta, mas em um tempo razoável a magnitude alcança valores absolutamente baixos. A principal razão desta característica é que o algoritmo de refinamento ignora parte das soluções apenas por estimar se o número de iterações preditas é maior do que $limT$.

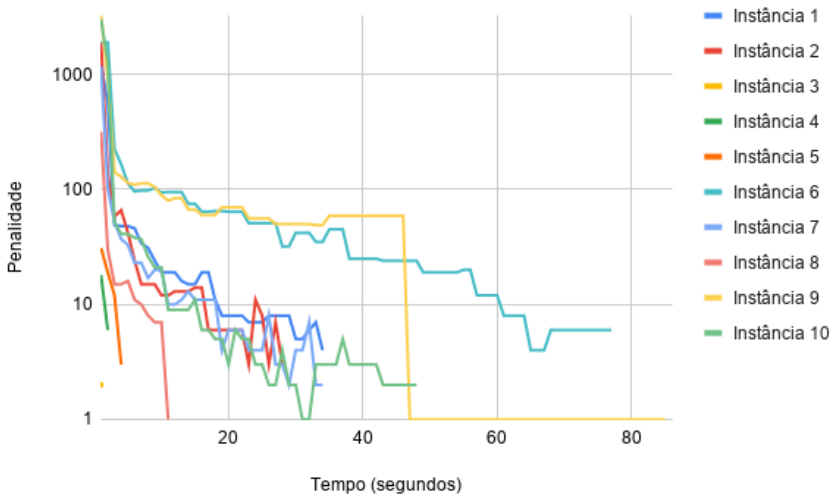

Figura 15 – Desempenho para categoria 1: heurística LS-LUBC.

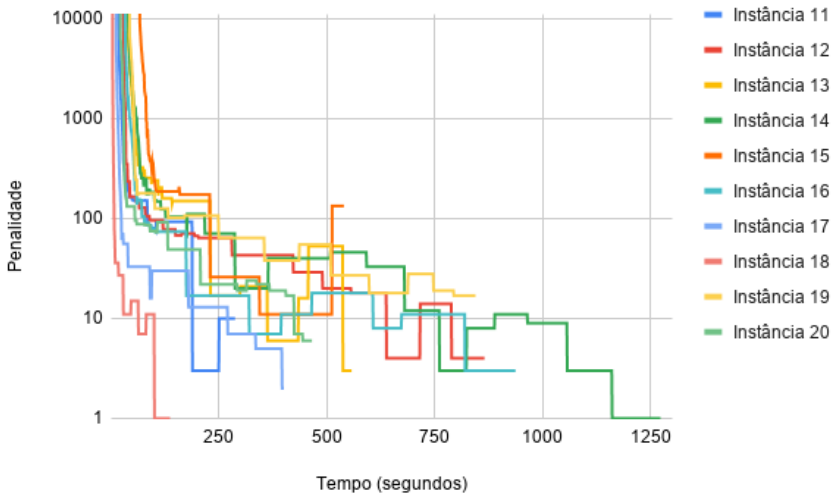

Figura 16 – Desempenho para categoria 2: heurística LS-LUBC.

6 CONSIDERAÇÕES FINAIS

Neste trabalho o problema *leasing k-center* foi introduzido e algoritmos foram propostos para resolvê-lo. O problema em si possui mais aplicações do que o problema *k-center* e tem grande potencial para ser utilizado em aplicações reais, por sua caracteristica de lidar com diferentes períodos de tempo e aluguéis.

Foram realizados experimentos computacionais para testar os algoritmos. A estratégia proposta para resolver o problema de forma exata consiste em uma formulação de Programação Linear Inteira Mista. Ao utilizar um resolvedor foi possível, através dos experimentos realizados, e considerando apenas um conjunto de vinte instâncias, obter soluções ótimas para as dez menores instâncias, assim como soluções viáveis não ótimas para as dez maiores instâncias. Uma heurística foi construída na forma de um algoritmo decodificador para a meta-heurística BRKGA. A outra heurística construida se baseia em resolver uma generalização do problema *Minimum Set Cover* através da meta-heurística *Local Search*.

Os resultados obtidos através de experimentos mostram que, nos casos onde o resolvedor apresentou uma solução ótima, as soluções encontradas pelas heurísticas foram boas, isto é, apresentaram um baixo valor GAP quando comparadas à ótima. Nos casos onde o resolvedor não foi capaz de apresentar uma solução em um prazo as soluções encontradas pelas heurísticas foram melhores do que as apresentadas pelo resolvedor.

Este trabalho serve de inspiração para trabalhos futuros, uma vez que não foram encontrados trabalhos relacionados. Como trabalho futuro, pretende-se demonstrar a aproximabilidade do problema, assim como propor heurísticas mais eficientes do que as que foram aqui propostas.

REFERÊNCIAS

ANTHONY, B. M.; GUPTA, A. Infrastructure leasing problems. In: **ICPO 2007: Integer Programming and Combinatorial Optimization. Lecture Notes on Computer Science**. [S.l.]: Springer, 2007. v. 4513, p. 424–438. Citado na página 15.

ARIYA, V. et al. Local search heuristics for k-median and facility location problems. **SIAM J. Comput.**, v. 33, n. 3, p. 544–562, 2004. Citado na página 19.

BEASLEY, J. E. Or-library: Distributing test problems by electronic mail. **The Journal of the Operational Research Society**, v. 41, n. 11, p. 1069–1072, 1990. Citado na página 31.

DASKIN, M. S. **Network and Discrete Locations: Models, algorithms, and applications**. [S.l.]: John Wiley & Sons, 1995. Citado 2 vezes nas páginas 15 e 20.

GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, 2010. Citado na página 18.

HAKIMI, S. Optimum location of switching centers and the absolute centers and medians of a graph. **Operations Research**, v. 12, p. 450–459, 1964. Citado 2 vezes nas páginas 10 e 14.

HAKIMI, S. Optimum location of switching centers in a communications network and some related graph theoretic problems. **Operations Research**, v. 13, p. 462–475, 1965. Citado na página 10.

HERTZ, A.; WIDMER, M. Guidelines for the use of meta-heuristics in combinatorial optimization. **Journal of Operational Research**, v. 151, p. 247–252, 2003. Citado na página 17.

HOCHBAUM, D. S.; SHMOYS, D. B. A best possible heuristic for the k-center problem. **Mathematics of Operations Research**, v. 10, n. 2, p. 180–184, 1985. Citado na página 15.

HSU, W.; NEMHAUSER, G. L. Easy and hard bottleneck location problems. **Discrete Applied Mathematics**, v. 1, p. 209–215, 1979. Citado na página 15.

IBÁNEZ, M. L. et al. The irace package, iterated race for automatic algorithm configuration. **Technical Report TR/IRIDIA/2011-004**, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. Citado na página 30.

KARIV, O.; HAKIMI, S. L. An Algorithmic Approach to Network Location Problems. I: The p-Centers. **SIAM Journal on Applied Mathematics**, v. 37, n. 3, p. 513–538, 1979. Citado na página 12.

KARP, R. M. Reducibility among combinatorial problems. In: **Complexity of Computer Computations**. [S.l.]: Springer, Boston, MA, 1972. p. 85–103. Citado na página 17.

LINTZMAYER, C. N.; MOTA, G. O. Caderno de problemas. In: **1º Workshop Paulista em Otimização, Combinatória e Algoritmos**. [S.l.: s.n.], 2017. Citado na página 12.

MINIEKA, E. The m-center problem. **SIAM Rev.**, v. 12, p. 138–139, 1970. Citado 2 vezes nas páginas 14 e 23.

MLADENOVIC, M. L. N.; HANSEN, P. Solving the p-center problem with tabu search and variable neighborhood search. **Networks**, v. 42, p. 48–64, 2003. Citado na página 15.

NAGARAJAN, C.; WILLIAMSON, D. P. Offline and online facility leasing. **Discrete Optimization**, v. 10, p. 361–370, 2013. Citado na página 15.

PEARL, J. **Heuristics: intelligent search strategies for computer problem solving**. [S.l.]: Addison-Wesley, 1984. Citado na página 17.

SANTOS, J. dos et al. 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, 2019. ISSN 2595-6116. Disponível em: <https://sol.sbc.org.br/index.php/etc/article/view/6397>. Citado 3 vezes nas páginas 15, 20 e 21.

SHIRALI, S.; VASUDEVA, H. L. **Metric Spaces**. [S.l.]: Springer, 2006. Citado na página 16.

TOSO, R.; RESENDE, M. A c++ application programming interface for biased random-key genetic algorithms. **Optimization Methods and Software**, 01 2015. Citado na página 30.

WILLIAMSON, D. P.; SCHMOYS, D. B. **The design of approximation algorithms**. [S.l.]: Cambridge University Press, 2011. Citado na página 18.

Apêndices

APÊNDICE A – Algoritmos da heurística LS-LUBC para o LKC

Algoritmo 4: *ConstruçãoU*

```
Saída: o conjunto U
início
    U ← ∅
    para cada t de T faça
        U ← U + t
        para cada f de D_t faça
            U ← U + (f, t)
        fim
        para cada l de L faça
            para cada i de V faça
                U ← U + (i, l, t)
            fim
        fim
    fim
fim
```

Algoritmo 5: *inicializaS*

```
Saída: o conjunto S
início
    S ← ∅
    para cada t de T faça
        para cada l de L faça
            para cada i de V faça
                S ← S + (i, l, t)
            fim
        fim
    fim
fim
```

Algoritmo 6: *ConstruçãoS′*

Saída: o conjunto S'
início
  $S' \leftarrow \emptyset$
  para cada t **de** T **faça**
    para cada l **de** L **faça**
      para cada i **de** V **faça**
        para $o \leftarrow t$ **até** $min(t + \delta_l - 1, t_{max})$ **faça**
          para cada f **de** D_t **faça**
            $S' \leftarrow S' + ((i, l, t), (f, t), d(i, f))$
          fim
        fim
      fim
    fim
  fim
fim

Algoritmo 7: *adicionar*

Entrada: $s, \mathcal{M}'$
Saída: $\mathcal{M}'$
início
  para cada f **de** s **onde** f **pertence a** D **faça**
    se $\mathcal{C}_f^D = 0$ **então**
      $\sigma \leftarrow \sigma - 1$
    fim
    $\mathcal{C}_f^D \leftarrow \mathcal{C}_f^D + 1$
  fim
  $p \leftarrow pai(s)$
  para $o \leftarrow p_t$ **até** $min(p_t + \delta_{p_l} - 1, t_{max})$ **faça**
    $\mathcal{C}_o^\varepsilon \leftarrow \mathcal{C}_o^\varepsilon + 1$
    se $\mathcal{C}_o^\varepsilon > k$ **então**
      $\tau \leftarrow \tau + 1$
    fim
  fim
  $\mathcal{M}' \leftarrow \mathcal{M}' + p$
fim

Algoritmo 8: *remover*

Entrada: $s, \mathcal{M}'$
Saída: $\mathcal{M}'$
início
  para cada f **de** s **onde** f **pertence a** D **faça**
    $\mathcal{C}_f^D \leftarrow \mathcal{C}_f^D - 1$
    se $\mathcal{C}_f^D = 0$ **então**
      $\sigma \leftarrow \sigma + 1$
    fim
  fim
  $p \leftarrow pai(s)$
  para $o \leftarrow p_t$ **até** $min(p_t + \delta_{p_l} - 1, t_{max})$ **faça**
    se $\mathcal{C}_o^\varepsilon > k$ **então**
      $\tau \leftarrow \tau - 1$
    fim
    $\mathcal{C}_o^\varepsilon \leftarrow \mathcal{C}_o^\varepsilon - 1$
  fim
  $\mathcal{M}' \leftarrow \mathcal{M}' - p$
fim

Algoritmo 9: *pen+*

Entrada: $s, \mathcal{M}'$
Saída: um valor inteiro *contador*
início
  $\mathcal{M}' \leftarrow adicionar(s, \mathcal{M}')$
  $contador \leftarrow \sigma \times \alpha + \tau \times \beta$
  $\mathcal{M}' \leftarrow remover(s, \mathcal{M}')$
fim

Algoritmo 10: *reduçãoLUBC*

Entrada: c
início
  para cada $x \in S'$ **onde** $x_w = c$ **faça**
    adicionar x_v no conjunto s de S onde x_u é o *pai*
  fim
fim

Algoritmo 11: *solInicial*

Saída: $\mathcal{M}_{ini}$
início
  $\mathcal{M}_{ini} \leftarrow \emptyset$
  para cada t **de** T **faça**
    para cada u de U **da partição** T_t **e do grupo** D **faça**
      se $\mathcal{C}_u^D = 0$ **então**
        $id \leftarrow -1$
        para $i \leftarrow 1$ **até** $|S|$ **faça**
          se u está em S_i **então**
            se $id \neq -1$ **então**
              se $pen{+}(S_i, \mathcal{M}_{ini}) < pen{+}(S_{id}, \mathcal{M}_{ini})$ **então**
                $id \leftarrow i$
              fim
            fim
            senão
              $id \leftarrow i$
            fim
          fim
        fim
        se $id \neq -1$ **então**
          $\mathcal{M}_{ini} \leftarrow adicionar(S_{id}, \mathcal{M}_{ini})$
        fim
      fim
    fim
  fim
fim

Algoritmo 12: *solRefinada*

Entrada: $\mathcal{M}_{ini}$
Saída: $\mathcal{M}_{ref}$
início
  $\mathcal{M}_{ref} \leftarrow \mathcal{M}_{ini}$
  $\mathcal{M}_{not} \leftarrow V \times L \times T - \mathcal{M}_{ini}$
  solAnteriores.enfileirar($pen+(\emptyset, \mathcal{M}_{ini})$)
  convergiu $\leftarrow 1$
  $i \leftarrow 1$
  $t_0 \leftarrow tempo()$
  enquanto *convergiu* $= 1$ **e** $i \leq limI$ **e** $tempo()$ - $t_0 \leq limT$ **faça**
    $a, b \leftarrow \emptyset$
    para cada $s'' \in \mathcal{M}_{ref}$ **faça**
      se $pen\text{-}(s'', \mathcal{M}_{ref}) < pen\text{-}(b, \mathcal{M}_{ref})$ **então**
        $b \leftarrow s''$
      fim
    fim
    se $b = \emptyset$ **então**
      para cada $s' \in \mathcal{M}_{not}$ **faça**
        se $pen+(s', \mathcal{M}_{ref}) < pen+(a, \mathcal{M}_{ref})$ **então**
          $a \leftarrow s'$
        fim
      fim
    fim
    para cada $s' \in \mathcal{M}_{not}$ **enquanto** $a, b = \emptyset$ **faça**
      para cada $s'' \in \mathcal{M}_{ref}$ **enquanto** $a, b = \emptyset$ **faça**
        se $pen+\text{-}(s', s'', \mathcal{M}_{ref}) < pen+(\emptyset, \mathcal{M}_{ini})$ **então**
          $a \leftarrow s'$
          $b \leftarrow s''$
        fim
      fim
    fim
    se $a = b = \emptyset$ **então**
      convergiu $\leftarrow 0$
    fim
    $adicionar(a, \mathcal{M}_{ref})$
    $remover(b, \mathcal{M}_{ref})$
    $\mathcal{M}_{not} \leftarrow \mathcal{M}_{not} - a + b$
    se $i \geq 3$ **então**
      ganhoMedio $\leftarrow$ (*solAnteriores*.desenfileirar() - $pen+(\emptyset, \mathcal{M}_{ref})$) / 3
      estimativa $\leftarrow i +$ $pen+(\emptyset, \mathcal{M}_{ref})$ / *ganhoMedio*
      se *estimativa* $> limI$ **então**
        $i \leftarrow limI$
      fim
    fim
    $i \leftarrow i + 1$
    solAnteriores.enfileirar($pen+(\emptyset, \mathcal{M}_{ref})$)
  fim
fim