UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Nichene Joslin Verçosa

Modelo e algoritmos para seleção de sensores como serviço

Recife

2019

Nichene Joslin Verçosa

Modelo e algoritmos para seleção de sensores como serviço

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Glauco Estácio Gonçalves

Coorientador: Victor Wanderley Costa de Medeiros

Recife

2019

Agradecimentos

Agradeço Emilia Joslin e Eduardo Scalia por permanecerem sempre ao meu lado e acreditarem em mim.

Agradeço à FACEPE e todo o JuaLabs por me possibilitarem e apoiarem a realização deste trabalho, com um ambiente de trabalho acolhedor e inspirador.

Agradeço, por fim, ao Professor Glauco, pelas grandes lições aprendidas, por ser sempre um grande exemplo de profissionalismo e dedicação com seus alunos.

“In a dark place we find ourselves, and a little more knowledge lights our way.”
(Yoda)

Resumo

A Internet das coisas (IoT) desempenha um papel fundamental no futuro da Internet, pois integrará bilhões de dispositivos inteligentes que apresentarão capacidades de detecção, atuação e processamento de dados. Assim, cada dispositivo integrado pode ter um ou mais sensores incorporados que potencialmente gerarão enormes quantidades de dados. Este cenário levanta o desafio de eficientemente procurar e selecionar os sensores mais adequados dentro de um conjunto que possa apresentar funcionalidades e recursos similares. Neste contexto, este trabalho apresenta um modelo matemático para seleção de sensores capaz de maximizar o atendimento aos requisitos de entrada do usuário, como precisão, robustez e disponibilidade para diferentes tipos de sensores (como temperatura, pressão, umidade, velocidade do vento) em um orçamento limitado. Este modelo foi testado através de dois algoritmos, sendo o primeiro um algoritmo de otimização com solução exata e o segundo uma heurística gulosa. Estas soluções foram avaliadas e comparadas em termos de três critérios: o tempo de otimização, a quantidade de orçamento a ser utilizada e limites do objetivo ótimo. As melhores soluções foram encontradas pelo algoritmo de otimização no entanto, a heusrística gulosa obteve resultados próximos com tempo de solução, 10x mais rápido que o algoritmo de otimização, em alguns casos.

Palavras-chave: Modelagem Matemática, Algorithms, Sensoriamento como Serviço.

Abstract

The Internet of Things (IoT) plays a key role in the future of the Internet, as it will integrate billions of smart devices which will present sensing, actuation and data processing capabilities. Thus, each integrated device may have one or more built-in sensors that will potentially generate huge amounts of data. This scenario raises the challenge of efficiently searching and selecting the most appropriate sensors within a set that can present similar functionalities and capabilities. In this context, this work presents a mathematical model for sensor selection able to maximize the attendance to user input requirements, such as accuracy, robustness, and availability for different types of sensors (such as temperature, pressure, humidity, wind speed, and so on) in a limited budget. This model was tested through two algorithms, the first being an optimization algorithm and the second a greedy heuristic. These solutions were evaluated and compared in terms of three criteria: the optimization time, the amount of budget being utilized, and the optimal value. The best solutions were found by the optimization algorithm but, the greedy heuristic found close results. In some cases, the greedy heuristic found solutions more than 10x faster when compared to the Optimal Algorithm.

Keywords: Sensing as a Service, Mathematical Modeling, Algorithms.

Lista de ilustrações

Lista de abreviaturas e siglas

IoT	Internet of Things
LP	Linear Programming
MMC	Método Monte Carlo
MIP	Mixed-Integer Programming
PM	Problema da Mochila
S^2aaS	Sensing as a Service

Sumário

1 Introdução

A coleta de parâmetros ambientais é essencial para que pesquisadores possam monitorar, analisar e compreender diversos processos e fenômenos naturais (BENGHANEM, 2009) (MORAIS et al., 2008). No entanto, ainda não é comum o uso de um serviço flexível que colete estes dados e os disponibilize para os usuários finais de maneira eficiente.

Uma forma de criar um mecanismo para o problema da coleta de dados e monitoramento de sensores é o emprego de técnicas presentes em sistemas de Internet das Coisas (IoT - *Internet of Things*) aliados à serviços em nuvem. Chamado de *Sensing as a Service* (Sensoriamento como um serviço - S²aaS (PERERA et al., 2014a), este tipo de serviço é composto por diversas estações de coleta com múltiplos sensores que se conectam, por meio de módulos de transmissão sem-fio, a serviços de armazenamento, processamento e visualização centralizados na Nuvem para enviar periodicamente os dados coletados em seus sensores.

O S²aaS difere dos mecanismos tradicionais de telemetria por Internet utilizados há anos em soluções de sensoriamento remoto (MORAIS et al., 2008), por conta da possibilidade de compartilhar a infraestrutura de estações de coleta entre diversos usuários que dinamicamente reconfiguram a infraestrutura para atender seus interesses de coleta. Assim, além de permitir a coleta eficiente dos dados e a possibilidade do monitoramento da saúde das estações de coleta, o S²aaS permite compartilhar todo o conjunto de sensores instalados entre múltiplos usuários que estejam interessados em observar fenômenos diferentes a partir dos dados dos mesmos sensores.

No entanto, o S²aaS ainda apresenta uma gama de configurações que podem ser otimizadas, como por exemplo, a seleção dos sensores a serem utilizados, dentro de um orçamento previsto. O número de sensores implementados no mundo está aumentando a um ritmo muito rápido e quanto mais sensores se conectam à Internet mais torna-se difícil a pesquisa de um sensor com características específicas. A diferença nas características podem ser em tipos (sensor de temperatura, pressão, umidade, velocidade do vento etc) ou nos parâmetros de coleta escolhido por cada usuário (taxa de coleta, custo, frequência de envio, precisão das amostras, nível de tolerância com faltas de dados etc). (PERERA et al., 2014a)

Por isto, quando quantidades significativas de sensores estão disponíveis para escolha, a seleção de sensores e estações de coleta torna-se uma tarefa bastante demorada. Por exemplo, um usuário que queira escolher um sensor localizado no agreste de Pernambuco pode ter, como necessidade primordial, a obtenção dos da-

dos de temperatura e umidade com um período máximo de uma hora além de um custo pré-determinado, e, por outro lado, pode aceitar alguma variação maior no que concerne à precisão destes. Todos estes requisitos e suas respectivas flexibilidades, bem como seus custos inerentes, devem ser levados em consideração durante este processo de seleção.

Dentro deste contexto, este trabalho de conclusão de curso teve como principal objetivo conceber um modelo matemático e algoritmos capazes de selecionar as sensores em estações de coleta de acordo com as necessidades dos usuários.

Este processo foi iniciado pela avaliação dos trabalhos relacionados ao tema, a fim de se buscar quais modelos já haviam sido aplicados, suas características e lições aprendidas. Desta forma, foi possível identificar que o problema de seleção de sensores tinha forte relação com o famoso Problema da Mochila e que esta relação ainda não havia sido abordada na literatura. Após análise mais profunda deste problema, identificou-se como possibilidade a adaptação de solução para o problema da mochila ao problema estudado neste trabalho. Neste sentido, criou-se um modelo matemático do problema de seleção de estações de coleta inspirado no Problema da Mochila, com suas devidas adaptações ao cenário proposto. A partir do entendimento do problema, foram propostos dois algoritmos que solucionam o modelo e estes foram avaliados por meio de simulações.

1.1 Objetivos

O presente trabalho teve como objetivo a investigação e o desenvolvimento de algoritmos para seleção de estações de coleta, baseado em parâmetros definidos por usuário de um sistema S²aaS. De forma específica, pretendeu-se:

- Criar um modelo matemático do problema de seleção de estações de coleta, com o intuito de estudar sua complexidade e classifica-lo por meio de comparação de problemas existentes;
- Propor algoritmos para seleção das estações de coleta mais adequadas para cada usuário de acordo com os requisitos fornecidos e custos inerentes;
- Avaliar o desempenho da soluções propostas através de simulações.

1.2 Estrutura do Trabalho

Na sequência desta introdução, o presente trabalho está estruturado da seguinte forma: No capítulo 2, é realizado um levantamento do estado da arte a respeito do tema com uma revisão da literatura e algumas descrições de outras soluções.

Neste capítulo discute-se ainda o Problema da Mochila, apresentando suas principais soluções. No capítulo 3, o modelo matemático criado é apresentado em detalhes e sua categoria de complexidade é determinada. Em seguida, os experimentos realizados, suas avaliações e discussões são tecidas no capítulo 4. Finalmente, o Capítulo 5 apresenta uma conclusão e perspectivas para pesquisas futuras.

2 Fundamentação Teórica

O presente capítulo pretende, primeiramente, apresentar uma visão geral da área de sensoriamento como serviço com foco no problema da alocação de sensores (Seção 2.1). Desta forma, pretende-se situar o leitor a respeito do estado da arte na área. Este capítulo apresenta ainda uma visão global do Problema da Mochila (Seção 2.2) com sua história, formulação, complexidade e principais soluções.

2.1 Trabalhos Relacionados

A IoT tem um papel fundamental para o futuro da Internet, uma vez que irá integrar bilhões de dispositivos de comunicação inteligentes que terão capacidades de sensoriamento, atuação e processamento de dados. Sendo assim cada dispositivo integrado poderá ter um ou mais sensores incorporados para captação de dados. Este cenário levanta o desafio de pesquisar e selecionar, de forma eficiente e eficaz, os sensores certos para uma consulta (PERERA et al., 2014a).

Semelhante a muitas outras tecnologias distribuídas em larga escala, a integração e a escalabilidade são os desafios mais proeminentes a serem resolvidos. A proposta de S²aaS visa enfrentar esses desafios. Segundo o *Internet Business Solutions Group* da Cisco, haverá 50 bilhões de dispositivos conectados à Internet até 2020, já de cordo com a Comissão Europeia este número é de 100 bilhões de dispositivos conectados. (WANG et al., 2015)

O crescimento das implantações de sensores aumentou expressivamente nos últimos 5 anos. Isso se deve, entre outros fatores, aos avanços na tecnologia de sensor e nuvem, na capacidade de processamento e armazenamento, à diminuição do custo de produção do sensor e do custo de aquisição de dados. Mesmo que, em algum momento, os custos iniciais sejam altos, a aquisição regular de dados se configura como muito barata e com bons retornos sobre os investimentos. De acordo com a BCC Research, o mercado global de sensores girava em torno US $ 91,5 bilhões em 2016 com uma taxa de crescimento anual composta de 7,8%. (ZASLAVSKY; PERERA; GEORGAKOPOULOS, 2013)

Segundo (PERERA et al., 2014b), o número de coisas conectadas à Internet excedeu o número de pessoas na Terra em 2008. Isto fica muito claro quando leva-se em consideração os telefones móveis. A maioria dos telemóveis atuais estão equipados com um conjunto de sensores incorporados, como câmara, GPS, rádios WiFi/ 3G/ 4G, acelerômetro, bússola digital, giroscópio, microfone e assim por diante. Além disso,

os sensores externos (como o sensor de batimentos cardíacos, o sensor de poluição do ar, etc.) também podem ser conectados a um telefone celular através de sua interface Bluetooth. Esses sensores podem permitir aplicações sensoriais atraentes em vários domínios, como monitoramento ambiental, redes sociais, saúde, transporte, etc. (SHENG et al., 2013)

A população mundial está crescendo em ritmo acelerado. As cidades e os municípios estão acomodando metade da população mundial, criando assim uma tremenda pressão sobre todos os aspectos da vida urbana. As cidades são conhecidas por terem grande concentração de recursos e instalações. No entanto, a atração sem precedentes tornou-se uma questão imensa para a governança e política da cidade. A enorme pressão para a gestão eficiente da cidade desencadeou várias iniciativas de cidades inteligentes, tanto do governo quanto do setor privado, para investir em tecnologias de informação e comunicação para encontrar soluções sustentáveis para os problemas crescentes. Resíduos, tráfego, energia, água, educação, desemprego, saúde e gestão da criminalidade são algumas das questões críticas. Espera-se que as cidades inteligentes lidem com esses desafios de maneira eficiente e eficaz usando as tecnologias da informação e comunicação. Por definição, as cidades inteligentes, através do uso de sensores e IoT acoplaram seis características: economia inteligente, pessoas inteligentes, governança inteligente, mobilidade inteligente, ambiente inteligente e vida inteligente. (PERERA et al., 2014b)

Toda esta relevância, fez com que a IoT tenha recebido uma atenção significativa tanto da academia como da indústria. No entanto, a pesquisa a respeito da seleção de sensores não foi estudada de maneira abrangente. Especificamente, as técnicas para busca e seleção de sensores usando informações de entrada dos usuários ainda não foram exploradas substancialmente. Neste contexto, quando um grande número de sensores está disponível para seleção, torna-se um desafio e uma tarefa demasiadamente complexa escolher o melhor conjunto de sensores que ajudarão os usuários a resolver seus problemas.

Ao passo que a IoT ajuda a aumentar consideravelmente a quantidade e a disponibilidade de dados coletados para diferentes propósitos, criando novas oportunidades interessantes, traz consigo também uma gama de desafios. Por exemplo, é muito difícil procurar e selecionar um sensor ou um grupo de sensores desejados quando não há descrição e precisão sobre os dados fornecidos. Neste contexto, a adequada descrição dos serviços desempenha um papel fundamental, uma vez que quanto mais rica a descrição do serviço, melhor será a interpretação e compreensão do próprio serviço, convergindo para a melhor escolha dos sensores necessários (NETO et al., 2015).

Alguns estudos vêm sendo realizados nesta área de pesquisa, abordando diversos aspectos do problema e criando soluções com abordagens heterogêneas.

O artigo de Truong (TRUONG; ROMER; CHEN, 2012) apresenta um serviço de busca na *Web of Things* (Teia das Coisas) que, de modo similar a uma busca tradicional na internet, permite encontrar sensores com certas propriedades. Seu algoritmo de comparação explora conjuntos fuzzy (conjuntos com graus de pertinência) para, de maneira eficiente, computar uma pontuação de similaridade para um par de sensores, convergindo para uma lista ordenada de sensores com a melhor correspondência às propriedades requeridas pelo usuário. Ao utilizar dados de sensores de implementações reais em sua pesquisa, foram obtidos resultados com elevada precisão. De maneira similar, o serviço da Microsoft SensorMap permite aos usuários selecionar sensores utilizando um mapa de localização, por tipo de sensor e por palavras-chave (PERERA et al., 2014c).

Ainda explorando a localização dos sensores, o artigo de (WANG et al., 2015) teve como objetivo projetar um método eficiente para a descoberta de serviços de sensores em grande escala com base em indexação geoespacial, técnicas semânticas e orientação à serviços. Este projeto contou com os seguintes recursos: (1) indexação geoespacial para reduzir eficaz e efetivamente o espaço de pesquisa; (2) resiliência ao dinamismo das operações de atualização, reduzindo seus impactos computacionais; (3) computação mais precisa das propriedades espaciais dos *gateways* de serviços de sensores usando técnicas geométricas computacionais; e (4) repositórios semânticos distribuídos e *matchmaking* de serviço semântico para fornecer resultados mais precisos.

O CASSARAM é um modelo de pesquisa, seleção e classificação de sensores de contexto, que aborda o desafio de selecionar eficientemente um subconjunto de sensores relevantes de um grande conjunto de sensores com funcionalidade e capacidades similares (PERERA et al., 2014c). Esta solução considera as preferências do usuário e uma ampla gama de características de sensores, como confiabilidade, precisão, localização, duração da bateria etc.

O CASSARAM foi desenvolvido para que o usuário através de uma interface possa definir seus requisitos. Em seguida, o algoritmo faz com que cada sensor seja modelado num espaço multidimensional em que cada dimensão representa uma propriedade de contexto (por exemplo, precisão, etc.). Cada dimensão é normalizada no intervalo [0,1]. Com isto, é gerado o *Comparative Priority Based Weighted Index* (Índice Ponderado com base na Prioridade Comparativa) para cada sensor, combinando as prioridades do usuário e os valores das propriedades de contexto. Os sensores são classificados usando o índice e o número de sensores requerido pelo usuário é selecionado no topo da lista.

O artigo de (EBRAHIMI et al., 2017), inspirado no algoritmo da otimização da colônia de formigas, propõe um método de reconhecimento de contexto para agrupar

sensores na forma de *Sensor Semantic Overlay Networks* (SSONs), em que sensores com informações de contexto semelhantes são reunidos em um cluster. Em primeiro lugar, os sensores são agrupados com base nos seus tipos para criar SSONs. Em seguida, o algoritmo meta-heurístico chamado AntClust agrupa os sensores utilizando suas informações de contexto. Além disso, foram desenvolvidos alguns métodos úteis para reduzir o custo do processo de busca de sensores. Suas experiências mostraram que o AntClust é significativamente mais rápido na pesquisa de sensores quando comparado com outras abordagens, como por exemplo, o CASSARAM.

A seleção de sensores é uma tarefa que necessita um maior amadurecimento, bem como o estudo de novas alternativas. Desta forma, a análise dos estudos apresentados acima inspirou a concepção do modelo matemático desenvolvido para este trabalho, a partir do desdobramento do Problema da Mochila, visto que esta abordagem ainda não havia sido testada.

2.2 O Problema da Mochila

2.2.1 Fundamentos

O problema da mochila (PM) é um problema de otimização combinatória. O nome dá-se devido a situação em que é necessário preencher uma mochila com objetos de diferentes pesos e valores. O objetivo é que se preencha a mochila com o maior valor possível, não ultrapassando seu peso máximo permitido.

O termo “problema de mochila” invoca a imagem do mochileiro que é restringido por uma mochila de tamanho fixo e, portanto, deve preenchê-la apenas com os itens mais úteis ou essenciais. Os problemas da mochila são, normalmente, representados por modelos de programação inteira, onde parte ou a totalidade das variáveis envolvidas no problema pertencem ao conjunto dos números inteiros.

A programação inteira, por sua vez, é uma extensão do problema de programação linear geral. Pode-se supor programação inteira como sendo um programa linear com requisitos de indivisibilidade, que, no caso binário, decide-se entre os valores 0 e 1 (SARKER; NEWTON, 2007).

Dentro do contexto da Programação Linear Inteira (PLI), o problema da mochila é um clássico extensamente estudado. Sua relevância histórica se deve ao fato de que: (i) este é o mais simples dos problemas de PLI; (ii) ele se apresenta como um subproblema de diversos outros mais complexos; e (iii) representa uma elevada gama de situações reais e práticas (SILVANO; PAOLO, 1990).

Dada sua abrangência conceitual, diversos problemas podem se correlacionar com o problema da mochila. Assim, o modelo em si pode ser aplicado no apoio à

tomada de decisões, em diversos casos como: investimento de capital, corte e empacotamento, carregamento de veículos e orçamento. Isto se deve ao fato de que, na realidade, o problema relata uma experiência bastante comum: “o que deve ser escolhido quando o espaço é limitado?” (BARTHOLDI, 2008).

Em uma abordagem mais simplificada, o problema da mochila é representado pela história de um ladrão, que possui uma mochila, vários objetos de diferentes valores e uma decisão a tomar: qual a seleção de objetos de maior valor. Todavia, quando o problema é estendido e adaptado para outros cenários, ampliando também sua complexidade, pode-se notar variações como por exemplo: o ladrão possuir mais de uma mochila; uma mochila com vários bolsos e vários objetos de cada tipo com o seu valor; ou ainda, possíveis objetos fracionáveis (em pó, por exemplo). Desta forma, sua complexidade pode variar de acordo com diferentes objetivos, restrições e requisitos de entrada.

2.2.2 História

O problema da mochila parece ter sido identificado pela primeira vez em 1957, em duas importantes publicações. Uma delas foi um artigo de George Dantzig (DANTZIG, 1957), um dos desenvolvedores de programação linear e criador do campo de Pesquisa Operacional. No mesmo ano, Richard Bellman, outro nome importante na pesquisa operacional, descreveu como usar a programação dinâmica para resolver o problema da mochila (BARTHOLDI, 2008).

Nos anos 60 do século passado, pesquisadores como Gilmore and Gomory investigaram a fundo o PM. Em 1967 Kolesar desenvolveu o algoritmo *branch-and-bound*, que se provou ser o único que pode resolver o problema com um grande número de variáveis. Este algoritmo, de forma aperfeiçoada, ainda hoje é utilizado por ferramentas de solução de problemas de programação matemática (*solvers*), tais como o Gurobi, que apresentam soluções exatas ótimas (BARTHOLDI, 2008).

Nos anos 70, o problema foi um alvo popular para o desenvolvimento de métodos de aproximação. De fato, um dos primeiros “esquemas de aproximação polinomial” foi desenvolvido para o problema da mochila por Sahni em 1975. Curiosamente, nesta época, o problema da mochila foi também a primeira base sugerida para um sistema de criptografia de chave pública. Este trabalho foi descrito por (DIFFIE; HELLMAN, 1976) e (MERKLE; HELLMAN, 1978). Deve-se notar que a abordagem foi posteriormente “quebrada” por criptógrafos e substituída por esquemas mais resistentes (SILVANO; PAOLO, 1990).

Ao longo da década de 1980, ainda foram publicados muitos trabalhos propondo técnicas de solução que não garantem a produção de soluções ótimas, pois elas es-

trategicamente perdem alguma qualidade para diminuir o tempo de solução.

Mais recentemente, pesquisadores como (KELLERER; PFERSCHY; PISINGER, 2004) trabalharam em maneiras de resolver instâncias cada vez maiores do problema da mochila. Muitos deles envolvem a solução de alguns ”núcleos”do problema e, em seguida, a construção dessa solução parcial para uma solução completa (BARTHOLDI, 2008).

A amplitude conceitual e aplicabilidade do PM faz com que seja amplamente utilizado e estudado em diferentes ocasiões. O estudo da *Stony Brook University Algorithm Repository* de 1998, denota sua relevância histórica ao apontá-lo como o décimo nono problema mais popular e o terceiro mais necessário. (SKIENA, 1999) Desta maneira, o estudo de melhores e diferentes soluções para este problema, pode representar a otimização de diversas problemáticas humanas.

2.2.3 Formulação Matemática e Principais Soluções do Problema da Mochila

Na abordagem clássica do problema da mochila, supõe-se que exista uma mochila com capacidade total de W e n itens distintos, cada um com seu respectivo peso $w_1, \ldots, w_n$ e valor $v_1, \ldots, v_n$. Em termos matemáticos, a função objetivo procura maximizar a soma dos valores dos itens selecionados. A variável de decisão x_i se comporta da seguinte maneira: se o item i é alocado na mochila, a variável assume o valor 1, caso contrário, 0. Por fim, a única restrição ao problema se caracteriza por um somatório de peso não superior à capacidade da mochila (W).

Assim, o modelo matemático para o problema da mochila 0-1 é dado por,

$$\max \sum_{i=1}^{n} V_i X_i \tag{2.1}$$

sujeito à:

$$\sum_{i=1}^{n} W_i X_i \leq W \tag{2.2}$$

$$X_i \in \{0, 1\} \tag{2.3}$$

$$X_{ij} = \begin{cases} 1, \text{se o item } i \text{ for selecionado;} \\ 0, \text{caso contrário.} \end{cases} \tag{2.4}$$

O PM pode ser resolvido por uma abordagem ingênua, na qual bastaria programar um computador para examinar todas as possíveis combinações, selecionando a melhor daquelas que satisfazem a restrição. No entanto, à medida que as possíveis

combinações são ampliadas, sendo estas vetores binários x de ordem de 2^n, o tempo de resposta torna-se inviável. Por exemplo, mesmo um computador capaz de examinar um bilhão de vetores por segundo, exigiria mais de 30 anos para selecionar entre um número de itens n = 60 e 60 anos para n = 61 (SILVANO; PAOLO, 1990).

Isto se deve ao fato de que o problema da mochila é um membro da classe de complexidade tempo polinomial não determinístico (NP-difícil). Para esses problemas, não há algoritmos de solução eficientes conhecidos, ou seja, não existem algoritmos cujo tempo computacional cresce apenas como um polinômio do tamanho das instâncias do problema (MURAWSKI; BOSSAERTS, 2016).

Com isso, percebe-se que avaliar todas as opções não é uma solução plausível para o problema. Neste contexto, outras soluções que se utilizam de aproximações, programação dinâmica ou do algoritmo *branch-and-bound* foram desenvolvidas para reduzir o tempo da solução.

2.2.3.1 Aproximações e o Algoritmo Guloso

Para alguns problemas NP-completos, pode ser o suficiente encontrar soluções viáveis, mesmo que não sejam ideais. As aproximações normalmente apresentam uma diferença sobre o valor da solução encontrada e o valor da solução ótima, mas são processadas muito mais rapidamente.

No método Guloso constrói-se um algoritmo que trabalha em estágios, considerando uma entrada por vez. Em cada estágio é tomada uma decisão avaliando se uma entrada particular é uma solução ótima. Isto é feito tomando-se as entradas em uma ordem determinada por um processo de seleção, que é baseado em alguma medida de otimização que pode ou não ser a função objetivo. Na maioria das vezes, porém, essas medidas de otimização resultarão em algoritmos que gerarão soluções sub-ótimas.

Em 1957, George Dantzig (DANTZIG, 1957) propôs um algoritmo de aproximação guloso para resolver o problema da mochila ilimitada. Sua versão classifica os itens em ordem decrescente de valor por unidade de peso ($v_i \setminus w_i$). Em seguida, ele os insere na mochila, começando com o maior número possível de cópias do primeiro tipo de item, se houver, até que não haja mais espaço para mais. Daí passa-se ao segundo item (LAGOUDAKIS, 1996).

Além do método de escolher a melhor razão lucro/peso, outros algoritmos gulosos selecionam apenas os objetos que tem o maior lucro e que se encaixam na mochila, até que não se possa mais adicionar objeto algum. Apesar de bastante rápido e muitas vezes obter resultados próximos da solução ótima, o algoritmo pode, por vezes, obter resultados distantes do ótimo (LAGOUDAKIS, 1996).

Segundo (ZIVIANI et al., 2004) o algoritmo guloso pode ser implementado em

um processo iterativo que faz a melhor escolha de acordo com uma das estratégias apresentadas acima, desde que essa escolha atenda a capacidade da mochila. A complexidade desse algoritmo é O(n * log n), pois se faz necessária a ordenação dos itens de acordo com a estratégia de seleção gulosa e a inserção de cada item até que a mochila não suporte nenhum outro item. A complexidade de espaço é O(n), um vetor usado para guardar a solução. (SOUZA; RAFAEL, 2014)

2.2.3.2 Branch and Bound

O *branch-and-bound* é um método útil e confiável para encontrar soluções ótimas para problemas de programação inteira e discreta aplicados a vários problemas de otimização combinatória. Este método foi proposto por Land e Doig (1960) e consiste em enumerar soluções viáveis, consideradas promissoras, em uma árvore de pesquisa explícita ou implícita. (SOUZA; RAFAEL, 2014)

Cada nó na árvore pode ser visto como uma solução possível ou uma solução parcial do problema. Ao utilizar funções matemáticas e/ou algoritmos que determinam o limite superior e inferior do problema em relação ao valor ótimo da função objetivo, um grande número de nós da árvore pode ser descartado, reduzindo assim não apenas o espaço para resolver o problema mas também o seu tempo de execução. Na melhor das hipóteses, é necessário calcular completamente um caminho através da árvore e podar o resto. No pior dos casos, é necessário calcular a árvore inteira completamente e o algoritmo obtido pela aplicação do método é exponencial. (SOUZA; RAFAEL, 2014)

2.2.3.3 Programação Dinâmica

Programação Dinâmica, ou Função Recursiva, foi utilizada para resolver esse problema na década de 50. Este é um método aprimorado de usar recursividade que ao invés de chamar uma função várias vezes, na primeira vez que é chamado, armazena o resultado para que cada vez que a função for chamada novamente volte o resultado e não uma requisição para ser resolvida. Neste cenário deve-se notar que troca-se tempo por espaço em memória.

A abordagem de programação dinâmica pode ser descrita na idéia a seguir. Considere que o valor de x_n pode ser 0 ou 1. Se $x_n = 0$, então o melhor lucro possivelmente obtido é aquele obtido a partir dos objetos restantes $n - 1$ com capacidade de mochila W. Se $x_n = 1$, então o melhor lucro possivelmente obtido é o lucro p_n (já selecionado) mais o melhor lucro obtido pelo restante $n - 1$ objetos com capacidade de mochila $W - w_n$ (neste caso deve ser $w_n \leq W$). Portanto, o lucro ideal será o máximo desses dois “melhores” lucros (LAGOUDAKIS, 1996).

A ideia acima leva à seguinte relação de recorrência, onde $P(i, m)$, $i = 1, ..., n$, $m = 0, ..., W$ é o melhor lucro obtido dos objetos $1, \ldots, i$ com capacidade de mochila

m:

$$P(i,m) = \begin{cases} P(i-1,m), & W_i > m \\ \max, \begin{cases} P(i-1,m), & \\ P(i-1,m-w_i)+p_i, & W_i \leq m \end{cases} \end{cases} \tag{2.5}$$

com condições iniciais:

$$P(1,m) = \begin{cases} 0, & w1 > m \\ p1, & w1 \leq m \end{cases} \tag{2.6}$$

Então, o lucro ótimo é $P(n,W)$. Um algoritmo de programação dinâmica simplesmente tem que construir uma tabela $n \times W$ e calcular as entradas $P(i,m), i = 1,...,n$, $m = 0,...,W$, de maneira *"bottom up"*, de baixo para cima. Assim que o lucro ótimo $P(n,W)$ foi calculado, a solução ótima X pode ser encontrada por retrocesso através da tabela e atribuindo 0 e 1 para x_i de acordo com as seleções da função max (LAGOUDAKIS, 1996).

O algoritmo tem uma complexidade temporal de O(nW). Note que isto não é polinomial, uma vez que W pode ser exponencialmente grande para n, e $W = 2n$. Tal complexidade é chamada pseudo-polinomial. (LAGOUDAKIS, 1996)

2.2.3.4 O Uso de *Solvers* no Contexto do Problema da Mochila

Um *solver* é um software, possivelmente na forma de um programa de computador autônomo ou como uma biblioteca de software, que fornece soluções para problemas matemáticos. Um solucionador obtém descrições de problemas de forma genérica e calcula sua solução. Em um solucionador, a ênfase está na criação de um programa ou biblioteca que possa ser facilmente aplicado a problemas de características semelhantes.

O uso de software de otimização requer que uma função f seja definida em uma linguagem de programação adequada. Com os valores de entrada, o módulo de software que realiza f fornecerá o valor computado $f(x)$ e, em alguns casos, informações adicionais.

Deve-se notar que diferentes módulos de software de otimização podem ser facilmente testados na mesma função f, ou, um determinado software de otimização pode ser usado para diferentes funções f.

Existe uma vasta gama de solvers, exemplos incluem MATLAB, R, AMPL, MINTO, OptimJ e muitos outros. Neste trabalho, o Gurobi Optimizer foi usado porque é gratuito

para uso acadêmico, resolve Problemas de Programação Linear Inteira (nos quais o PM é categorizado) e é facilmente incorporado a programas em Python.

Gurobi é reconhecido como o solver que mais representa o estado-da-arte atualmente para programação matemática. Ele foi projetado para explorar arquiteturas modernas e processadores multi-core, usando as implementações dos mais recentes algoritmos. Isso permite flexibilidade na modelagem de problemas e aumenta a capacidade de adicionar mais complexidade aos modelos, a fim de melhor representar os problemas do mundo real a serem resolvidos (GUROBI…, s.d).

De acordo com o site do Gurobi (MIXED-INTEGER…, s.d), os problemas de programação linear inteira mista são geralmente resolvidos usando um algoritmo *branch-and-bound* baseado em programação linear, já expresso anteriormente neste trabalho. O método básico *branch-and-bound* aplicado pelo Gurobi Solver pode ser descrito da seguinte maneira.

Inicia-se pelo MIP original. Não sabendo como resolver este problema diretamente, todas as restrições de integralidade são removidas. O LP resultante é chamado de relaxamento de programação linear do MIP original. Este LP é então resolvido. Se o resultado satisfizer todas as restrições de integralidade, mesmo que estas não tenham sido explicitamente impostas, então esta solução teve sorte e o processo é interrompido (MIXED-INTEGER…, s.d).

Normalmente o caso ainda não está resolvido, então o procedimento normal é escolher alguma variável que seja restrita a ser inteira, mas cujo valor no relaxamento do LP é fracionário. Suponha que esta variável seja x, e seu valor na relaxação LP seja 5.7, este valor pode ser excluído impondo as restrições $x \leq 5.0$ e $x \geq 6.0$. A variável x é então chamada de variável de ramificação, e diz-se que as soluções ramificaram-se em x, produzindo os dois sub-MIPs P1 e P2. Deve ficar claro que, se for possível calcular as soluções ótimas para cada um de P1 e P2, então, tomando o melhor dessas duas soluções, isso será ótimo para o problema original, P0. Dessa forma, P0 foi substituído por dois MIPs mais simples (ou pelo menos mais restritos). (MIXED-INTEGER…, s.d)

Caso o problema não tenha sido resolvido ainda, então, recursivamente, aplica-se a mesma ideia a esses dois MIPs, resolvendo os relaxamentos LP correspondentes e, se necessário, selecionando variáveis de ramificação. Ao fazer isso, uma árvore de pesquisa, como a mostrada na Figura 1, é gerada. Os MIPs gerados pelo procedimento de pesquisa são chamados de nós da árvore, com P0 designado como o nó raiz. As folhas da árvore (nós verdes) são todos os nós que ainda não foram ”ramificados”. Em geral, quando uma solução é alcançada (nó roxo), todos os outros nós de folha podem ser descartados e o MIP original foi resolvido. (MIXED-INTEGER…, s.d)

Ao usar o solver toda a estrutura da árvore, o processamento e a tomada de

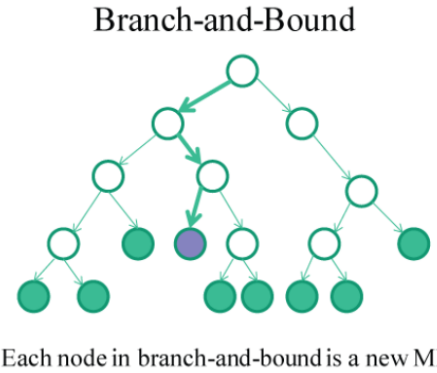

Figura 1 – Cada nó em uma arvore do branch-and-bound é um novo MIP
Fonte: (MIXED-INTEGER…, s.d)

decisão ficam ocultos do usuário, o que, por sua vez, pode usar diretamente os comandos de linguagem de programação escolhida para descrever o problema e obter os resultados desejados. Os passos para fazê-lo geralmente incluem: criar um modelo; adicionar as variáveis; definir a função objetivo; modelar as restrições; otimizar; e finalmente, obter os resultados.

3 O Modelo Matemático

O presente trabalho, conforme visto anteriormente, tem como um de seus objetivos propor um modelo matemático capaz de otimizar o problema de seleção de sensores em estações de coleta.

Um modelo é uma abstração ou representação matemática de um problema de interesse. Neste sentido, o processo de modelagem consolida-se como a parte essencial da resolução de problemas de forma otimizada. Por sua vez, otimização é o processo que busca encontrar a melhor solução de um problema. Esta solução, também conhecida como solução ótima, é qualquer uma das melhores soluções viáveis para o problema (SARKER; NEWTON, 2007).

De acordo com (SARKER; NEWTON, 2007), modelos de programação matemática, de uma maneira geral, apresentam 3 principais componentes:

- Função Objetivo: é a função de uma ou mais variáveis de decisão que se quer otimizar, minimizando-a ou maximizando-a;
- Variáveis de decisão: são aquelas que se alteram em busca da melhor solução durante o processo de otimização, podendo ser contínuas (reais), inteiras ou discretas;
- Restrições: são funções de igualdade ou desigualdade que representam limitações ao problema.

Segundo (GANDOMI; YANG; ALAVI, 2013 apud CARVALHO et al., 2014), os problemas de otimização, de uma maneira geral, podem ser classificados da seguinte maneira apresentada na Figura 2.

No presente trabalho, o problema de seleção de sensores em estações de coleta foi modelado para as seguintes classificações: multivariável, único objetivo, com restrições, discreta, linear e determinística. Além disto, cabe ressaltar que o problema a ser resolvido, além de se tratar de um problema de programação linear, é também inteiro e binário.

De maneira similar ao Problema da Mochila, a seleção de sensores em estações de coleta tem como objetivo escolher os sensores que melhor atendam os requisitos do usuário dentro de um orçamento dado. As variáveis de decisão deste problema são

Figura 2 – Classificação dos problemas de otimização
(CARVALHO et al., 2014)

definidas como:

$$X_{ij} = \begin{cases} 1, \text{se o sensor } i \text{ do tipo } j \text{ for selecinado;} \\ 0, \text{caso contrário.} \end{cases} \tag{3.1}$$

onde $i = 1, ..., n$ indica cada um exemplar de sensor do tipo $j = 1, ..., m$. Este tipo indica que tipo de medida o sensor requisitado pelo usuário deve fornecer (por exemplo: temperatura, umidade, pressão etc).

Introduz-se ainda $M_{ij} \in \mathbb{R}^+$ como um índice que indica em que grau o sensor i do tipo j atende os requisitos determinados pelo usuário. Dentre estes requisitos pode-se ter precisão, robustez, disponibilidade entre outros. Portanto, o usuário pode informar requisitos diferentes para sensores de cada tipo desejado. A quantidade mínima de sensores do tipo j desejada pelo usuário é informada pelo parâmetro $N_j \in \mathbb{N}^+$.

O limite do orçamento do usuário é indicado por $B \in \mathbb{R}^+$ e o custo de cada exemplar de sensor i do tipo j é dado por $V_{ij} \in \mathbb{R}^+$.

O programa inteiro procurará obter aqueles sensores que maximizam o atendimento dos requisitos sem ultrapassar o orçamento e oferecendo o mínimo de sensores de cada tipo pedido pelo usuário, conforme as equações abaixo.

Função objetivo e restrições:

$$\max \sum_{i=1}^{n} \sum_{j+1}^{m} M_{ij} X_{ij} \tag{3.2}$$

sujeito à:

$$\sum_{i=1}^{n} \sum_{j+1}^{m} V_{ij} X_{ij} \leq B \tag{3.3}$$

$$\sum_{i=1}^{n} X_{ij} = N_j, \quad \forall\, j = 1...m \tag{3.4}$$

Observa-se que a Função Objetivo busca maximizar o atendimento dos requisitos do usuário com os sensores selecionados. A seleção final dos sensores é realizada pela definição das variáveis de decisão X_{ij}, mais especificamente quando esta obtém o valor 1. A primeira restrição (Equação 3.3) estabelece um limite orçamentário ao problema. Por sua vez, a Equação 3.4 garante que a solução final contenha o número exato de tipos de sensores estabelecidos pelo usuário. Por exemplo, se o usuário define que ele deseja um sensor de temperatura e dois de umidade, o algoritmo deve apresentar em sua solução final exatamente um sensor de temperatura e dois de umidade.

Além disto pode-se elencar um conjunto de considerações a serem levadas em conta no modelo, sendo elas:

$$M_{ij},\ V_{ij},\ N_j, \text{ and } B \text{ são números não negativos;} \tag{3.5}$$

$$\sum_{j=1}^{m} V_{ij} > B; \text{ and} \tag{3.6}$$

$$V_{ij} \leq B, \quad \forall\, i = 1...n \text{ and } j = 1...m. \tag{3.7}$$

A respeito das considerações apresentadas acima cabem os seguintes comentários: se a soma dos valores de todos os sensores não for maior que o orçamento determinado pelo usuário (Equação 3.6) a solução do problema impreterivelmente será a seleção de todos os sensores; um sensor não será considerado na solução caso este apresente valor superior ao orçamento. Existe ainda uma quarta restrição que deve ser observada, o problema só apresentará uma solução se a soma dos N_j sensores de menor custo de cada tipo de sensor for menor ou igual ao orçamento.

3.1 Índice de atendimento dos requisitos

A metodologia de cálculo do Índice de atendimento dos requisitos (M_{ij}) segue a proposta por (WANG et al., 2015). Neste método, assume-se que cada sensor é definido como um ponto em um espaço multidimensional de propriedades, tais como, localização, robustez, precisão e frequência de coleta. Desta forma tem-se que

$$M_{ij} = \sqrt{\sum_{t=1}^{T} W_{tj}(U_{tj} - S_{tij})^2} \tag{3.8}$$

Onde:

- $M_{ij} \in$ [0,1];
- T é o número de propriedades dos sensores;
- W_{tj} é a prioridade da propriedade t no tipo j;
- U_{tj} é o valor do requisito do usuário da propriedade t no tipo j;
- S_{tij} é o valor da propriedade t do sensor i do tipo j.

Através da normalização dos valores das prioridades da propriedades (W_{ij}) , dos valores dos requisitos (U_{ij}) e dos valores das propriedades (S_{tij}) pode-se calcular para cada sensor sua distância euclidiana ponderada. A partir deste processo, pode-se calcular para cada sensor um número que representa o quão perto está um sensor real (no espaço multidimensional de propriedades) dos requisitos estabelecidos pelo usuário. Por isso, quanto maior o M, maior será o atendimento aos requisitos do usuário.

3.2 Complexidade

Pode-se demonstrar que o problema de seleção de sensores com orçamento é NP-Difícil, conforme mostrado no Teorema 1.

Teorema 1. Problema da seleção de sensores é NP-Difícil

Prova: Para demonstrar que o problema da seleção de sensores é NP-Difícil deve-se encontrar um problema-base que seja reconhecidamente NP-Difícil e, subsequentemente, encontrar uma redução de tempo polinomial deste problema-base ao problema da seleção de sensores. Pode-se utilizar o problema da mochila como o problema-base para esta demonstração.

Seja o problema da mochila dado pelos seguintes parâmetros reais positivos p_i, w_i e c (MARTELLO; TOTH, 1990), respectivamente, o valor de cada item i, o peso de cada item e a capacidade da mochila. Neste problema deve-se determinar se o item i será ou não adicionado à mochila, de acordo com a variável de decisão $x_i \in 0, 1$.

Pode-se reduzir este problema ao problema da seleção de sensores assumindo um único tipo de sensor $j = 1$ e fazendo $M_{i1} = p_i$, $V_{i1} = w_i$ e $B = c$.

O problema da seleção de sensores escolherá livremente os itens que melhor atendam os requisitos e que caibam no orçamento. O que é exatamente o desejado no problema da mochila, que procura escolher os itens mais preciosos (maior valor de p_i, ou M_{i1}, no caso do problema da seleção de sensores) sem ultrapassar a capacidade da mochila (c, ou B, no caso do problema da seleção de sensores).

Entretanto, o problema de seleção de sensores possui um parâmetro N_1 que deve ser informado. Já que o valor de N_j é fortemente relacionado à função objetivo do problema da mochila (i.e., mais itens fazem a função objetivo crescer), o valor de N_j pode ser definido por uma busca binária entre 1 e o número total de itens a ser considerado no problema da mochila. Para um dado valor de N_1 testado, se uma solução não é encontrada então sabe-se que o número de itens está além da capacidade da mochila e, por isso, deve-se buscar um valor menor de N_1; por outro lado, se uma solução é encontrada deve-se testar um valor maior de N_1, para tentar melhorar a função objetivo.

Por fim, as variáveis de solução X_{i1} do problema da seleção de sensores para o melhor valor de N_1 corresponderão à solução do problema da mochila ($x_i = X_{i1}$).

Como a redução do problema envolve apenas a atribuição de parâmetros e variáveis de decisão e uma busca binária, que pode ser feita em tempo polinomial, demonstra-se que o problema da mochila pode ser reduzido em tempo polinomial ao problema da seleção de sensores e que este último é NP-difícil.

4 Experimentos e Resultados

Este capítulo apresenta avaliações de algoritmos para a solução do problema de seleção de sensores.

4.1 Descrição dos experimentos

Foram elaborados duas soluções distintas para o problema de solução de sensores. Para a primeira solução, adotou-se o algoritmo *branch-and-bound* (GUROBI..., s.d) implementado no Gurobi Optimizer 7.0.1[1] com licença estudantil devido sua facilidade de uso, contudo outro *solver* poderia ter sido utilizado com mesmo propósito. A biblioteca para Python Gurobipy foi utilizada para a codificação do modelo. A segunda solução, uma heurística para obtenção de solução aproximada, utilizou uma abordagem gulosa desenvolvida em Python.

A simulação de cenários, foi desenvolvida para o estado atual do modelo matemático e com os dois algoritmos concebidos. É de se destacar que o trabalho contou com a execução de vários experimentos, realizados a partir de uma infraestrutura composta de uma máquina de sistema operacional OS X, versão 10.9.5, processador 2.6 GHZ Intel Core i7 e 8 GB de memoria RAM.

Além disso, para as simulações, foi gerado um banco de dados com um 500 mil sensores, igualmente divididos em vinte tipos de sensores. O índice de atendimento dos requisitos (M_{ij}) de cada sensor, visto que não existia uma massa real de dados ou usuários com necessidades específicas mapeadas, foi definido aleatoriamente através de uma distribuição Beta, com $alpha = 1$ e $beta = 3$. O custo (V) de cada sensor foi extraído de uma distribuição Normal, com média igual a $10 e desvio padrão $5.

A distribuição Beta foi escolhida para M_{ij} por gerar valores entre 0 e 1 e por ter alta probabilidade de geração de valores baixos (aproximadamente 80% dos valores ficam abaixo de 0,4). Desta forma, há um menor número de sensores cujo índice de atendimento ao usuário seja alto o que diminui a probabilidade de casos de solução mais simples (nas quais apenas o custo importaria). Note-se que as variáveis aleatórias são independentes, i.e., não há uma relação entre o valor de M_{ij} e de V para capturar situação de pior caso.

Os dois algoritmos foram submetidos a diferentes cenários variando os tipos de sensores e o número de sensores em cada tipo. Os primeiros testes contaram com apenas um tipo de sensor e o número de sensores variou em 1000, 5000, 10000 e

[1] http://www.gurobi.com/

50000. Em seguida, foram realizados testes com 10 tipos de sensores, onde o número de sensores por tipo variou em 100, 500, 1000, 5000, 10000 e 50000. Finalmente, os testes com 20 tipos de sensores contaram com a variação de sensores 50, 250, 500, 2500, 5000 e 25000 por tipo.

Todos os casos compartilharam o mesmo orçamento máximo ($500) e o mesmo número de sensores a serem selecionados (50). O número de sensores de cada tipo exigido pelo usuário (N_j) foi desenhado a partir de uma distribuição Multinomial com $n = 50$ e $p_1 = p_2 = \cdots = p_{20} = 1/20$. Vale ressaltar que, os sensores foram selecionados aleatoriamente do banco de dados de acordo com uma distribuição uniforme.

Os experimentos consistiram em repetidas amostragens aleatórias para obter resultados numéricos, isto é, sucessivas simulações seguindo-se os preceitos do Método Monte Carlo. Ou seja, com este número de repetições pode-se calcular probabilidades heuristicamente, tal como se, de fato, se registrassem os resultados reais de uma seleção de sensores por um usuário. Com isto pode-se obter aproximações numéricas de funções complexas e resultados estatisticamente confiáveis (PAULA et al., 2017). Cada cenário foi executado 30 vezes. E para os resultados foram tomadas as médias e intervalos de confiança (95% de confiança) para três métricas: tempo de otimização, valor ótimo do índice global de atendimento (função objetivo) e orçamento utilizado na solução ótima.

4.2 Implementação Otimizada do Modelo

O código a seguir demonstra a implementação do modelo por meio da biblioteca Gurobupy na linguagem Python. Além disto, o código mostra a geração das variáveis aleatórias para simulação dos cenários.

```
from gurobipy import *
import traceback
import sys
import random
import linecache
import datetime
import time
import numpy

if len(sys.argv) == 7:
   n = int(sys.argv[1])
   m = int(sys.argv[2])
   B = int(sys.argv[3])
   s = int(sys.argv[4])
```

```
    random.seed(s)
    #inicializa o gerador de número aleatórios
    num_rep = int(sys.argv[5])
    numSensPedidos = int(sys.argv[6])

else:
    print("Informar: n m B s num_rep numSensPedidos")
    print("n = Sensores(i) por tipo")
    print("m = Numero de tipos(j)")
    print("B = Orçamento do usuário")
    print("s = Seed (Int)")
    print("num_rep = número de repetições")
    print("numSensPedidos = número total de sensores pedidos")
    exit(1)

dt = datetime.datetime.now()
dt = dt.strftime('%Y-%m-%d_%H-%M')
nomeDoDiretorio = str(n) + "_" + str(m)
+ "_" + str(B) + "_" + str(s) + "_"+ str(num_rep) + "_" + str(numSensPedidos)
    + "_" + str(dt)
os.makedirs(nomeDoDiretorio)


def criarMatriz(linha, coluna):
    M = [[0 for x in range(coluna)] for y in range(linha)]
    return M


for num_da_rep in range(num_rep):

    # N = Número de sensores escolhidos por tipo.
    # exemplo: N = [1, 1, 3, 1, 1, 0, 0]
    #A probabilidade de ocorrer cada um é a mesma. Total de sensores é
        numSensPedidos.
    #N = numpy.random.multinomial(numSensPedidos, [1/numSensPedidos]*m)
    N = numpy.random.multinomial(numSensPedidos, [1.0/m]*m)

    try:
        #CRIAÇÃO DO MODELO
        print("CRIANDO O MODELO . . . ")
        model = Model("SensorSelect")
```

```
#M = atendimento dos requisitos, variando entre 0-1
#Matriz sensor i tipo j.
#exemplo: 5 tipos, cada tipo tem 4 sensores.
#    M = [[0.2, 0.3, 0.4, 0, 0],
#       [0, 0, 0.5, 0, 0],
#       [0, 0, 0.6, 0, 0],
#       [0.8, 1, 1, 1, 0]]


#V = custos dos sensores i do tipo j.
#exemplo: V = [[20.00, 10.22, 1.12, 2.32, 3.34],
#             [ 2.99, 3.87, 5.22, 4.21,30.34],
#            [ 6.11, 8.33, 8.44, 4.39,10.00],
#            [ 1.11, 2.21, 4.34, 4.53, 5.43]]

print("CRIANDO AS MATIZES M e V . . .")
M = criarMatriz(n,m)
V = criarMatriz(n,m)

for numDeTipos in range(m): #passar por todos os arquivos CSVs dos tipos
    de sensores

   #Randomicamente escolhe n linhas de um arquivo.
   #exemplo: pegar 4 sensores de uma lista de 2500.
   numLinhas = random.sample(range(1,50001), n)

   numDoSensor = 0
   for numLinha in numLinhas:
      l = linecache.getline("file" + str(numDeTipos)+".csv", numLinha)
      tipo,indiceM,numValor,barraN = l.split(";")
      M[numDoSensor][numDeTipos] = indiceM
      V[numDoSensor][numDeTipos] = numValor
      numDoSensor += 1

nrange = range(n)
mrange = range(m)

#CRIAÇÃO DAS VARIÁVEIS DE DECISÃO
print("CRIANDO AS VARIÁVEIS DE DECISÃO . . .")
#X[i,j] == 1 se sensor for selecionado; 0 caso contrário
X = model.addVars(nrange, mrange, vtype=GRB.BINARY, name="X")
#print(X[(0,0)])
```

```
#ESTRUTURAÇÃO DA FUNÇÃO OBJETIVO
print("ESTRUTURAÇÃO DA FUNÇÃO OBJETIVO . . .")
listaProdutosMX = []
for i in nrange:
   for j in mrange:
      listaProdutosMX.append(M[i][j] * X[(i,j)])
#nao é possivel usar numpy.multiply pq é preciso acessar cada objeto em X.
#X é um array de tupleDict
#listaProdutosMX = numpy.multiply(M,X)

model.setObjective(sum(listaProdutosMX),GRB.MAXIMIZE)

listaProdutosVX = []
for i in nrange:
   for j in mrange:
      listaProdutosVX.append(V[i][j] * X[(i,j)])

#DEFINIÇÃO DAS RESTRIÇÕES
print("ADICIONANDO AS RESTRIÇÕES . . . ")

#listaProdutosVX = numpy.multiply(V,X)
model.addConstr(sum(listaProdutosVX), GRB.LESS_EQUAL, B, "c0")

for j in mrange:
   model.addConstr(X.sum('*',j), GRB.GREATER_EQUAL, N[j], "c1")

#OTIMIZAÇÃO
print("OTIMIZANDO . . .")
#time.time = tempo total
#time.clock = tempo em que procesador trabalhou nas funções
tClock = time.clock()
tTime = time.time()
model.optimize()
tempoDeOtimizacaoTime = time.time() - tTime
tempoDeOtimizacaoClock = time.clock() - tClock

tempoDeOtimizacaoTime = str(tempoDeOtimizacaoTime)
tempoDeOtimizacaoClock = str(tempoDeOtimizacaoClock)

#ESCRITA DAS RESPOSTAS
print("SALVANDO AS RESPOSTAS ENCONTRADAS . . . ")
```

```
      arquivo = "/Users/nichenevercosa/SensorSelect/" + str(nomeDoDiretorio) +
          "/" + str(num_da_rep) + ".txt"
      arquivoRespostas = open(arquivo, "w")

      if (model.solCount > 0 ):
         arquivoRespostas.write("N = " + str(N) + "\n")
         arquivoRespostas.write("M = " + str(M) + "\n")
         arquivoRespostas.write("V = " + str(V) + "\n")

         arquivoRespostas.write("Tempo de otimizacao (time.time) = " +
             tempoDeOtimizacaoTime + "\n")
         arquivoRespostas.write("Tempo de otimizacao (time.clock) = "+
             tempoDeOtimizacaoClock + "\n")

         arquivoRespostas.write('Objective value for the current solution = ')
         arquivoRespostas.write(str(model.objVal)+ "\n")
         arquivoRespostas.write('The best known bound on the optimal objective
             = ')
         arquivoRespostas.write(str(model.ObjBoundC)+ "\n")

         for v in model.getVars():
            arquivoRespostas.write(str(v.varName)+" = ")
            arquivoRespostas.write(str(v.x)+ "\n")

      else:
         arquivoRespostas.write("Nao exite uma solucao para este problema")
      print("FIM DA REPETIÇÃO " + str(num_da_rep) + " . . . ")

   except GurobiError:
      print('Error reported: ')
      traceback.print_exc()

print(" . . END . . \o/ ")
```

As figuras 3. 4 e 5 mostram os resultados para o cenário básico com apenas 1 tipo de sensor e com um número crescente de sensores. Como pode ser visto na Figura 3, o tempo médio de otimização deste primeiro cenário mostra um comportamento exponencial, que concorda com a complexidade apresentada e comprovada no modelo matemático. O tempo médio de solução para o caso com 50.000 sensores foi de apenas 1,08 segundos.

A Figura 4 mostra que, nos testes com 1.000 sensores, o valor médio da função

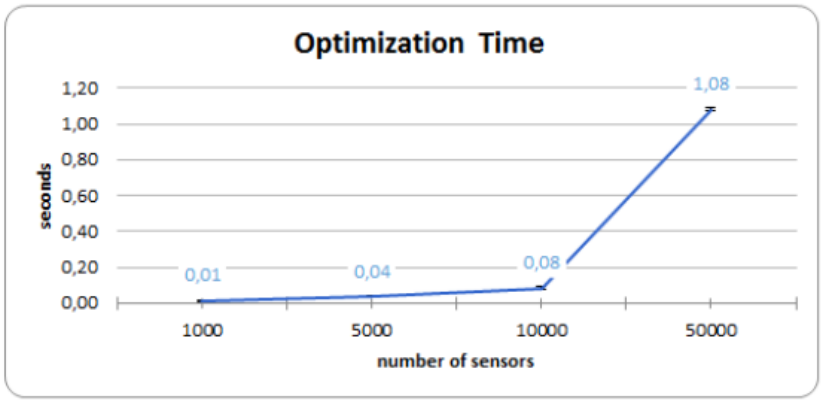

Figura 3 – Tempo de Otimização para a solução ótima no cenário com apenas 1 tipo de sensor.

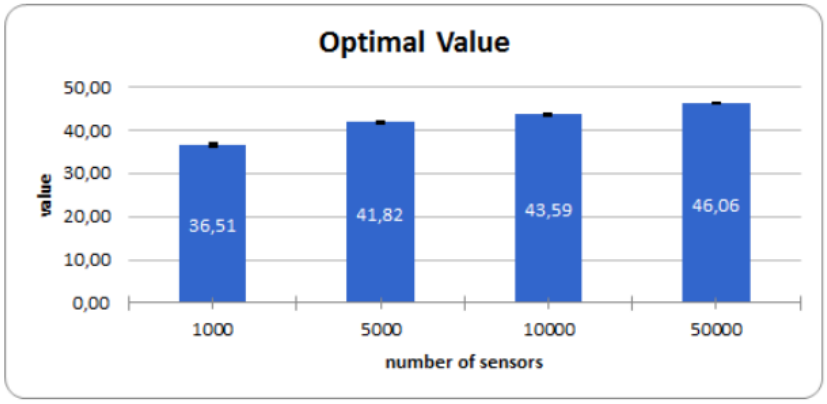

Figura 4 – Valor Ótimo para a solução ótima no cenário com apenas 1 tipo de sensor.

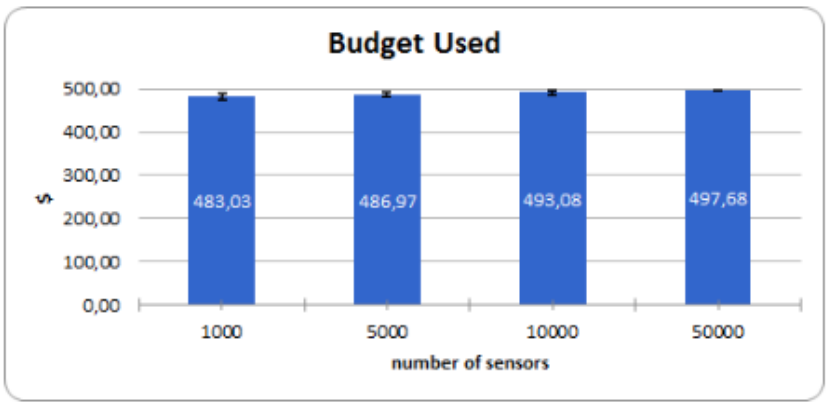

Figura 5 – Orçamento Utilizado para a solução ótima no cenário com apenas 1 tipo de sensor.

objetivo é igual a 36.51 e, no caso de 50.000 sensores, esse valor é igual a 46.06. Os números crescentes retratam o fato de que quando o algoritmo possui mais sensores para escolher, pode-se selecionar os melhores índices correspondentes e, portanto, encontrar os melhores sensores para cada solicitação específica. Contudo, observa-se que este fenômeno tem um limite já que o crescimento torna-se pequeno com o

aumento do número de sensores.

No que diz respeito ao orçamento (com 1 tipo de sensor), pode-se destacar que, uma vez que o orçamento foi definido em $500 para todos os casos, o algoritmo encontrou soluções que usavam quase a quantidade total em todos os casos, embora os intervalos de confiança indicam que os valores estatisticamente não são os mesmos.

Também pode ser observado a partir dos resultados para os outros cenários nas Figuras 6, 7 e 8 que os resultados são similares ao do primeiro cenário com apenas 1 tipo de sensor. Assim, pode-se inferir que o número de tipos de sensores não afeta o comportamento geral do modelo. A Figura 12, apresentando os resultados relativos ao tempo de otimização para os três cenários simulados, também reforça essa semelhança.

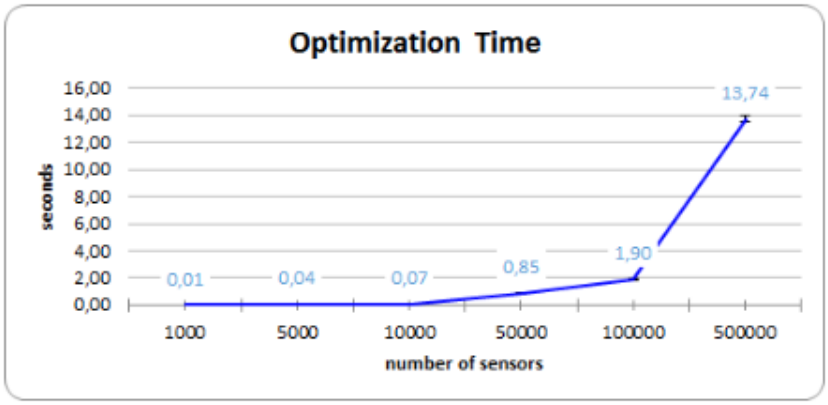

Figura 6 – Tempo de Otimização para a solução ótima com 10 tipos de sensores.

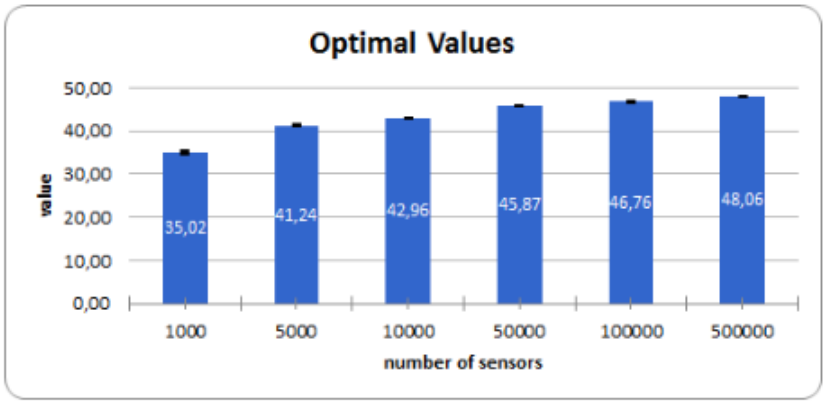

Figura 7 – Valor Ótimo para a solução ótima com 10 tipos de sensores.

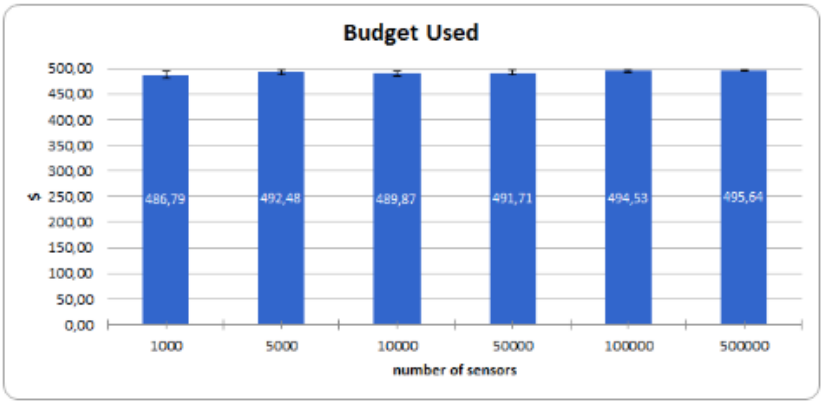

Figura 8 – Orçamento Utilizado para a solução ótima com 10 tipos de sensores.

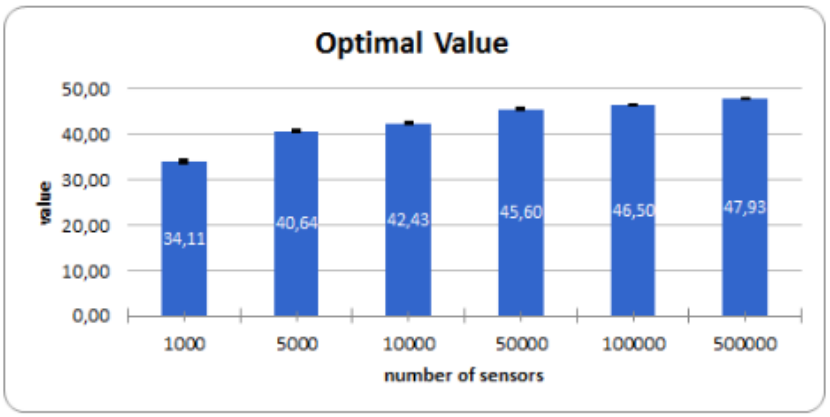

Figura 10 – Valor Ótimo para a solução ótima com 20 tipos de sensores.

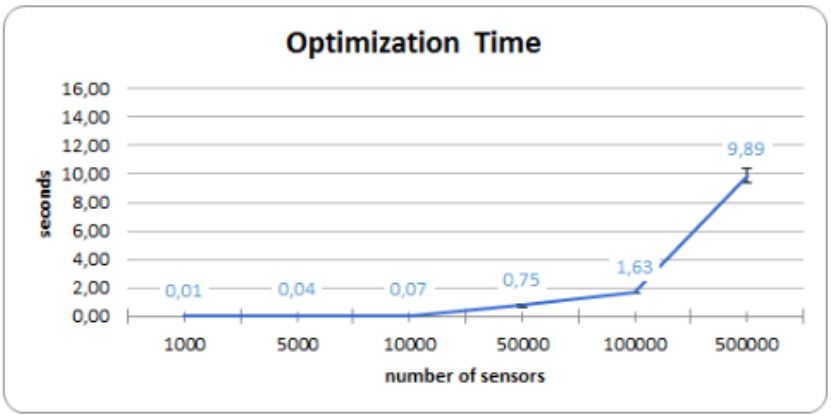

Figura 9 – Tempo de Otimização para a solução ótima com 20 tipos de sensores.

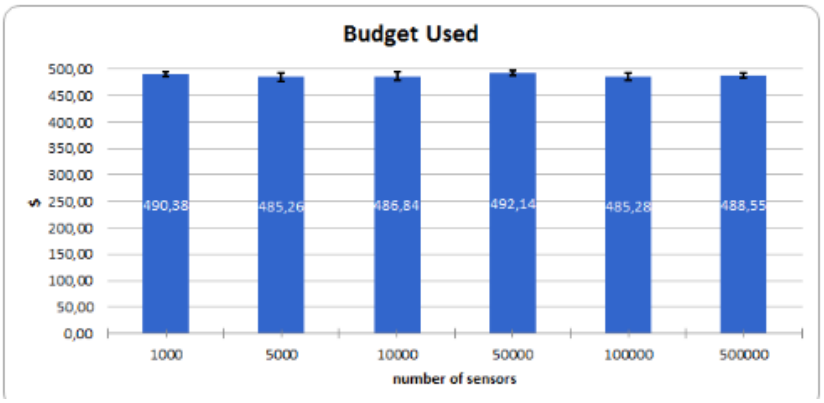

Figura 11 – Orçamento Utilizado para a solução ótima com 20 tipos de sensores.

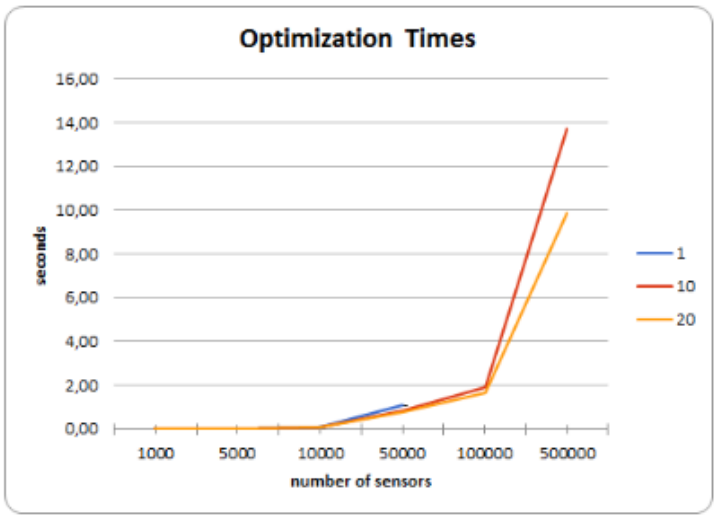

Figura 12 – Tempo de Otimização para a solução ótima com 1, 10 e 20 Tipos de Sensores.

4.3 Implementação Gulosa

Nesta etapa, é apresentada a implementação do segundo algoritmo proposto neste trabalho, uma solução do problema de seleção de sensores utilizando uma abordagem gulosa, como uma heurística que busca encontrar soluções aproximadas para o problema de forma rápida. A motivação para a busca deste tipo solução se dá pela natureza interativa e multiusuária do sistema S²aaS. Neste tipo de sistema, o algoritmo de seleção pode vir a se tornar um gargalo caso demande um elevado tempo de execução o que prejudicaria a experiência do usuário.

O algoritmo guloso proposto ordena os sensores, com base em um parâmetro desejado, do mais relevante para o menos relevante, e seleciona, nesta ordem, o número de sensores de cada tipo requisitados de modo que o orçamento não seja ultrapassado. Caso o algoritmo percorra toda a lista e não encontre a quantidade de

sensores requisitados dentro do limite orçamentário, este não apresenta resposta.

Foram realizadas 3 parametrizações. A primeira parametrização procurou encontrar a solução com a melhor relação custo-benefício (M/V), i.e., entre o índice de atendimento dos requisitos (M_{ij}) e o valor (V_{ij}). A segunda, por sua vez, priorizou os valores dos sensores (V). Por fim, a terceira buscou selecionar os sensores priorizando o índice de atendimento dos requisitos (M).

4.3.1 Algoritmo Guloso com maior M/V

O algoritmo desenvolvido que leva em consideração o parâmetro maior M/V é apresentado a seguir:

```
import traceback
from heapq import nlargest
import sys
import os
import random
import linecache
import datetime
import time
import numpy

global numDotipo
global listaItens

global usedBudget
global somaM

global done
global escolhidos

if len(sys.argv) == 7:
   n = int(sys.argv[1])
   m = int(sys.argv[2])
   B = int(sys.argv[3])
   s = int(sys.argv[4])
   random.seed(s) #inicializa o gerador de número aleatórios
   num_rep = int(sys.argv[5])
   numSensPedidos = int(sys.argv[6])

else:
   print("Informar: n m B s num_rep numSensPedidos")
```

```
    print("n = Sensores(i) por tipo")
    print("m = Numero de tipos(j)")
    print("B = Orçamento do usuário")
    print("s = Seed (Int)")
    print("num_rep = número de repetições")
    print("numSensPedidos = número total de sensores pedidos")
    exit(1)

dt = datetime.datetime.now()
dt = dt.strftime('%Y-%m-%d_%H-%M')
nomeDoDiretorio = str(n) + "_" + str(m) + "_" + str(B) + "_" + str(s) + "_"+
     str(num_rep) + "_" + str(numSensPedidos) + "_" + str(dt)
os.makedirs(nomeDoDiretorio)

def criarMatriz(linha, coluna):
    M = [[0 for x in range(coluna)] for y in range(linha)]
    return M

for num_da_rep in range(num_rep):

    usedBudget = 0
    somaM = 0

    arquivo = "/Users/nichenevercosa/SensorSelect/" + str(nomeDoDiretorio) +
        "/" + str(num_da_rep) + ".txt"
    arquivoRespostas = open(arquivo, "w")

    N = numpy.random.multinomial(numSensPedidos, [1.0/m]*m)

    print ("CRIANDO MATRIZES V, M e DivisaoMV\n")
    V = criarMatriz(n,m)
    M = criarMatriz(n,m)
    DivisaoMV = criarMatriz(n,m)

    for numDeTipos in range(m):
        numLinhas = random.sample(range(1,50001), n)
        numDoSensor = 0
        for numLinha in numLinhas:
            l = linecache.getline("file" + str(numDeTipos)+".csv", numLinha)
            tipo,indiceM,numValor,barraN = l.split(";")
            M[numDoSensor][numDeTipos] = round(float(indiceM),5)
            V[numDoSensor][numDeTipos] = round(float(numValor),2)
```

```
      numDoSensor += 1

nrange = range(n)
mrange = range(m)

for i in nrange:
   for j in mrange:
      if (V[i][j] == 0):
         valorDivisao = 0
      else:
         valorDivisao = (M[i][j] / V[i][j])
      DivisaoMV[i][j] = round(float(valorDivisao),5)

numDotipo = 0
print ("OTIMIZANDO\n")

tTime = time.time()
done = False
escolhidos = []
for numSenEscolhidos in N:
   done = False
   listaItens = []

   if (numSenEscolhidos > 0):
      for i in nrange:
         valorDiv = (DivisaoMV[i][numDotipo])
         item = ((i,numDotipo), valorDiv)
         listaItens.append(item)

      resposta = nlargest(numSenEscolhidos, listaItens, key=lambda e:e[1])

      for maioresIndices in resposta:
         iSensor, jSensor = maioresIndices[0][0], maioresIndices[0][1]
         valorm = M[iSensor][jSensor]
         valorv = V[iSensor][jSensor]

         sensorescolhido = (iSensor,jSensor)
         escolhidos.append(sensorescolhido)

         somaM += valorm
         usedBudget += valorv
```

```
            if (usedBudget > B):
               done = True
               break

         if (done):
            arquivoRespostas.write("Nao exite uma solucao para este problema")
            break

      numDotipo += 1

   tempoDeOtimizacaoTime = time.time() - tTime
   tempoDeOtimizacaoTime = str(tempoDeOtimizacaoTime)

   print("GUARDANDO RESPOSTAS \n")
   if (usedBudget <= B):

      arquivoRespostas.write("N = " + str(N) + "\n")
      arquivoRespostas.write("M = " + str(M) + "\n")
      arquivoRespostas.write("V = " + str(V) + "\n")
      arquivoRespostas.write("MV = " + str(DivisaoMV)+ "\n")
      arquivoRespostas.write("Tempo de otimizacao (time.time) = " +
          tempoDeOtimizacaoTime + "\n")
      arquivoRespostas.write('The best known bound on the optimal objective = ')
      arquivoRespostas.write(str(somaM)+ "\n")
      arquivoRespostas.write("O orcamento utilizado foi = "+ str(usedBudget)+
          "\n")
      arquivoRespostas.write("Lista dos sensores escolhidos = "+
          str(escolhidos))

   print ("\n---FIM da repeticao " + str(num_da_rep) + "---\n")

   arquivoRespostas.close()

print (" - - FIM - - \o/ ")
```

As Figuras 13, 14 e 15 mostram os resultados deste algoritmo no cenário com apenas 1 tipo de sensor.

A parametrização que busca a melhor relação M/V seria a escolha mais comum em uma abordagem gulosa. No entanto, uma vez que o parâmetro M_{ij} apresenta valores de 0 a 1, e uma parcela significativa de V_{ij} também, a solução não encontra soluções com bons resultados, mas sim aquelas que possuem orçamento muito baixo

e que ainda tem índices de atendimento razoáveis. O valor ótimo encontrado é aproximadamente metade do obtido pelo algoritmo ótimo, como pode ser visto nas Figuras 14, 17 e 20.

Conforme pode ser visto nas Figuras 15, 18 e 21, à medida que se aumentou o número de sensores, o algoritmo trouxe como resposta soluções cada vez mais baratas. Sendo assim, este parâmetro não foi adequado para o problema da seleção de sensores da maneira que este foi concebido.

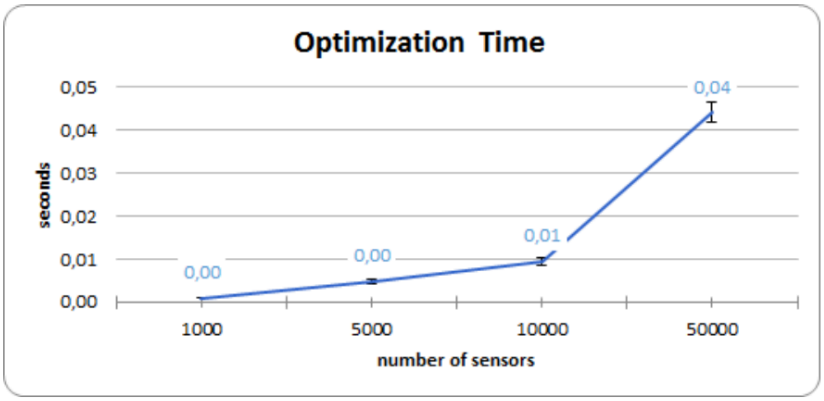

Figura 13 – Tempo de Otimização para algoritmo guloso parâmetro maior M/V com 1 tipo de Sensor.

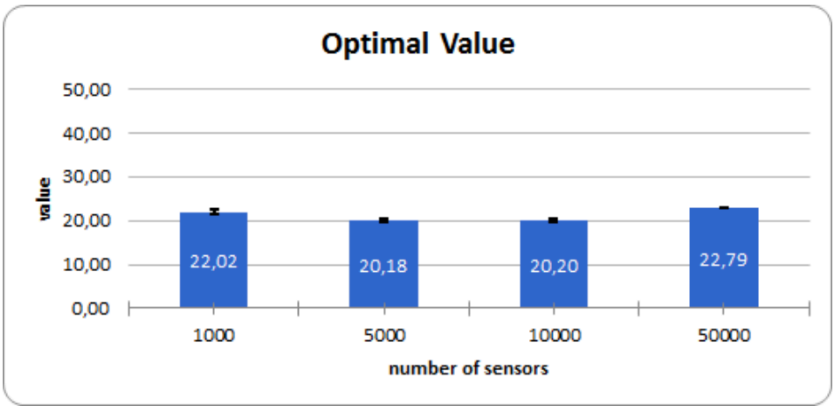

Figura 14 – Valor da Função Objetivo para algoritmo guloso parâmetro maior M/V com 1 tipo de Sensor.

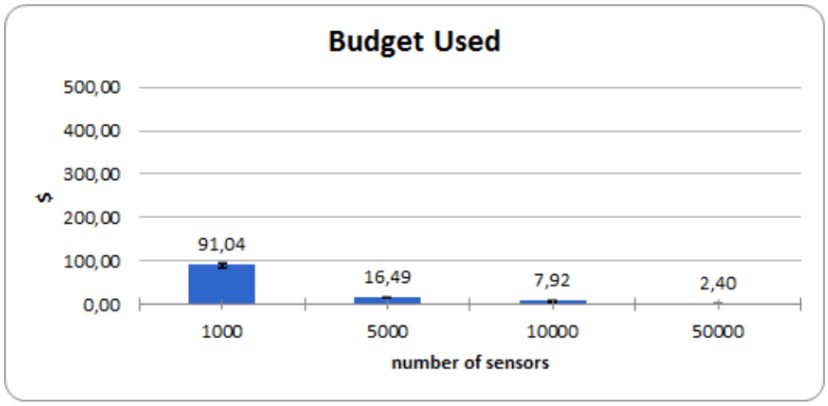

Figura 15 – Orçamento Utilizado para algoritmo guloso parâmetro maior M/V com 1 tipo de Sensor.

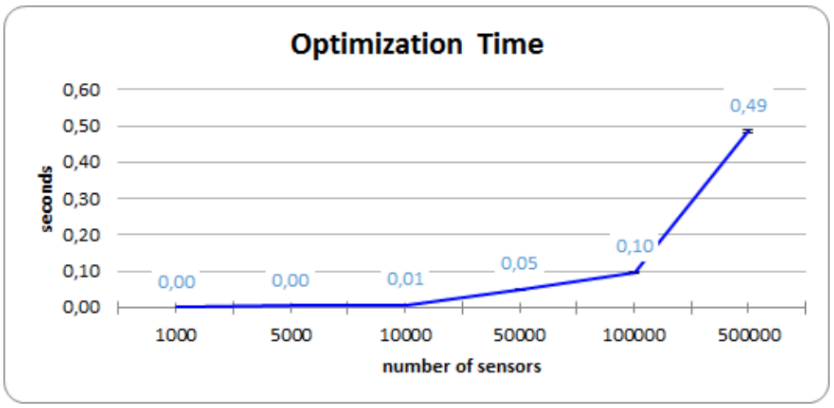

Figura 16 – Tempo de Otimização para algoritmo guloso parâmetro maior M/V com 10 tipos de Sensores.

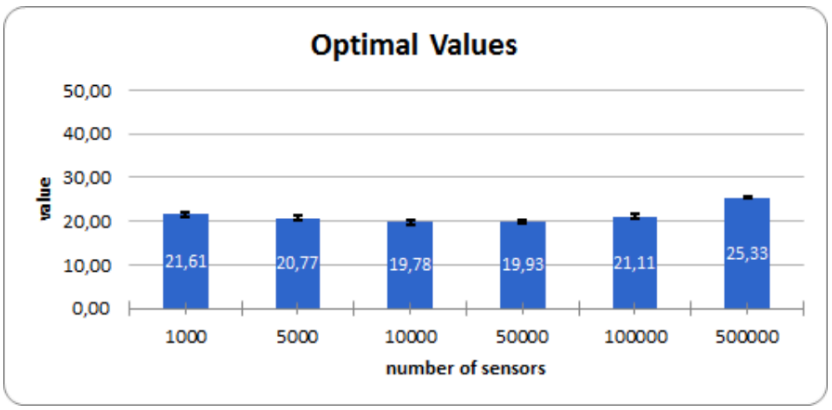

Figura 17 – Valor Ótimo para algoritmo guloso parâmetro maior M/V com 10 tipos de Sensores.

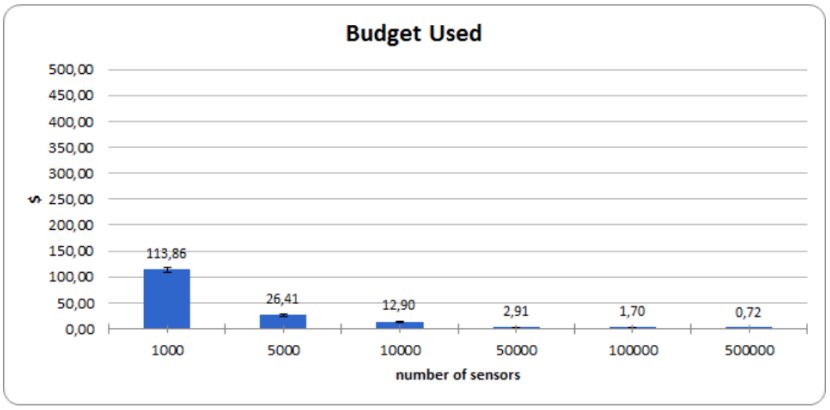

Figura 18 – Orçamento Utilizado para algoritmo guloso parâmetro maior M/V com 10 tipos de Sensores.

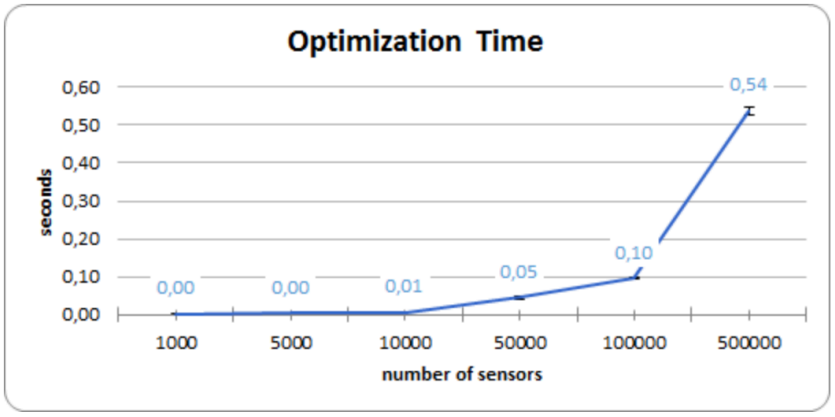

Figura 19 – Tempo de Otimização para algoritmo guloso parâmetro maior M/V com 20 tipos de Sensores.

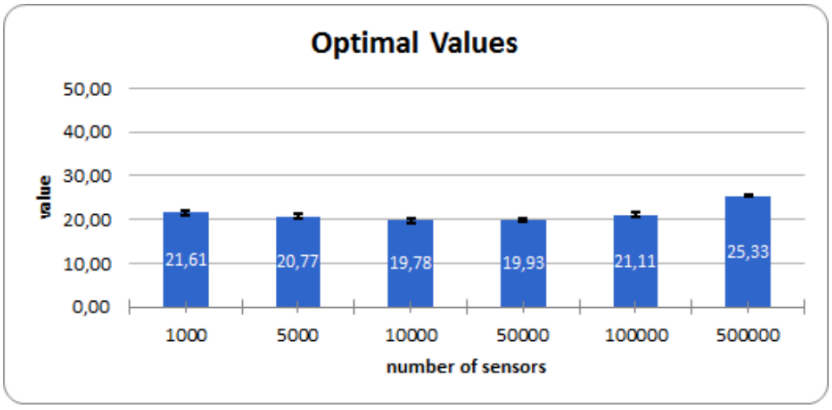

Figura 20 – Valor Ótimo para algoritmo guloso parâmetro maior M/V com 20 tipos de Sensores.

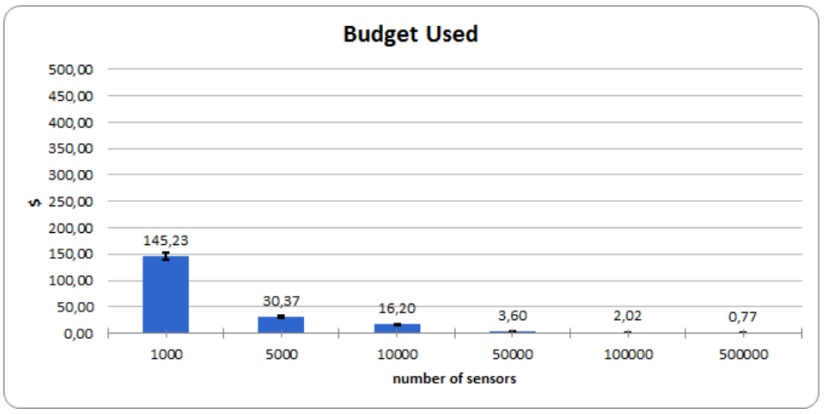

Figura 21 – Orçamento Utilizado para algoritmo guloso parâmetro maior M/V com 20 tipos de Sensores.

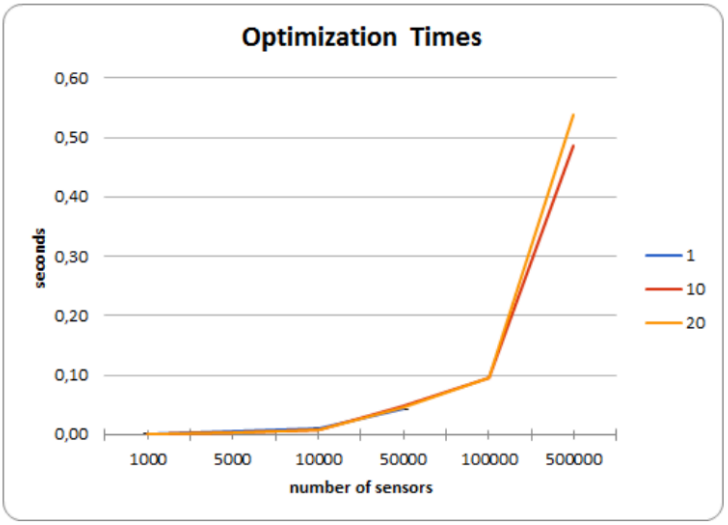

Figura 22 – Tempos de Otimização no Algoritimo Guloso de parâmetro maior M/V com 1, 10 e 20 Tipos de Sensores.

4.3.2 Algoritmo Guloso (Parâmetro: maior V)

A parametrização que busca o maior V foi a que apresentou piores resultados dentre todos os algoritmos, uma vez que, de fato, encontrou resultados utilizando-se o máximo do orçamento, mas não obteve resultados significativos em termos do índice de atendimento aos requisitos, com resultados sempre inferiores a 9 (como pode ser visto nas Figuras 24, 27 e 30, numa escala de 0 a 50. Sendo assim, este parâmetro não atende ao problema da seleção de sensores. O algoritmo desenvolvido é bastante similar ao do parâmetro M/V e por isto pode ser encontrado no Apêndice A.

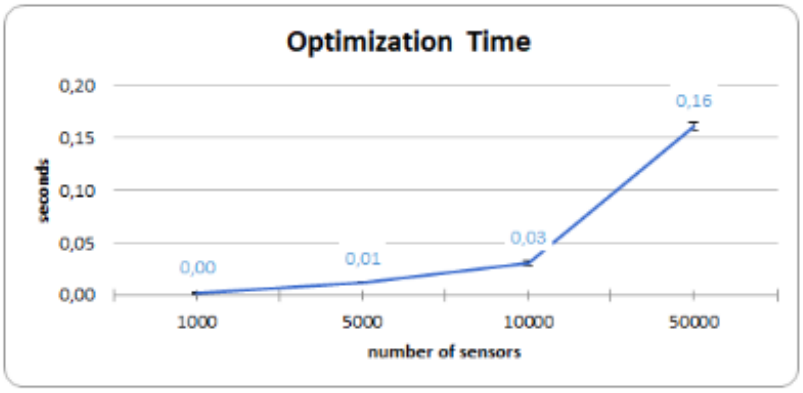

Figura 23 – Tempo de Otimização para algoritmo guloso parâmetro maior V com 1 tipo de Sensor.

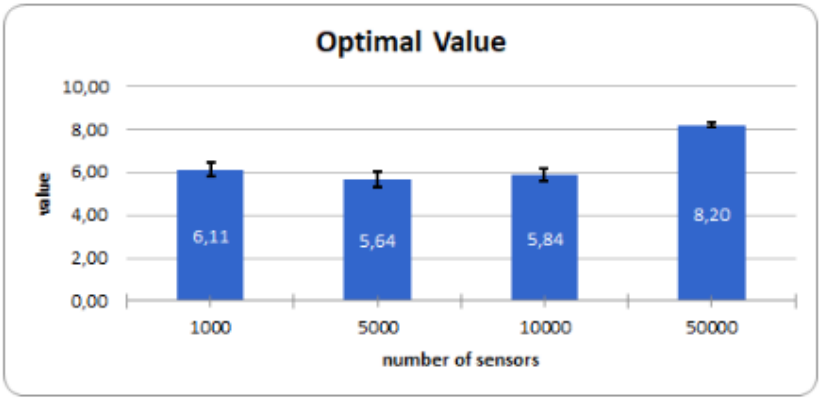

Figura 24 – Valor Ótimo para algoritmo guloso parâmetro maior V com 1 tipo de Sensor.

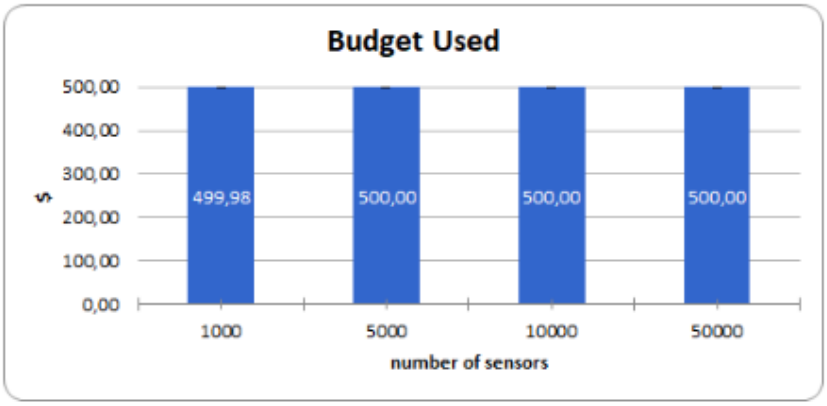

Figura 25 – Orçamento Utilizado para algoritmo guloso parâmetro maior V com 1 tipo de Sensor.

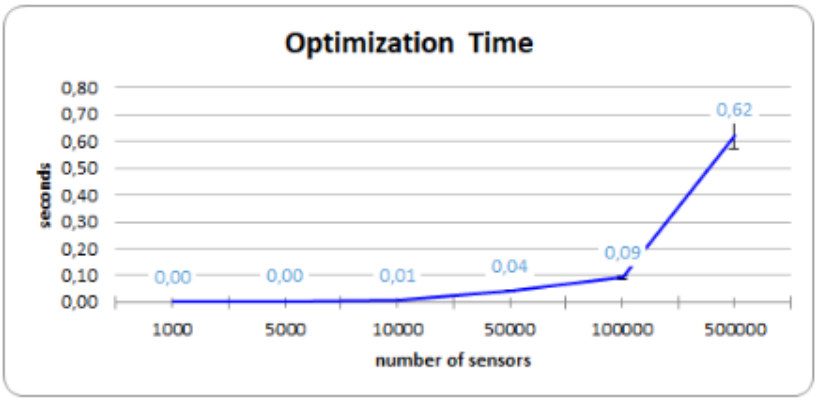

Figura 26 – Tempo de Otimização para algoritmo guloso parâmetro maior V com 10 tipos de Sensores.

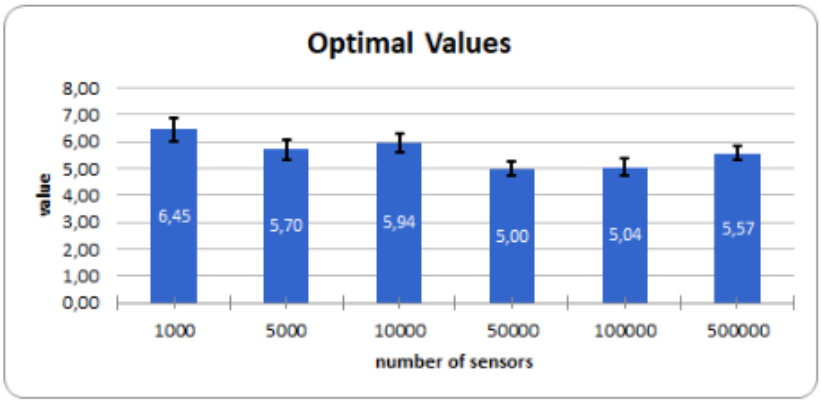

Figura 27 – Valor Ótimo para algoritmo guloso parâmetro maior V com 10 tipos de Sensores.

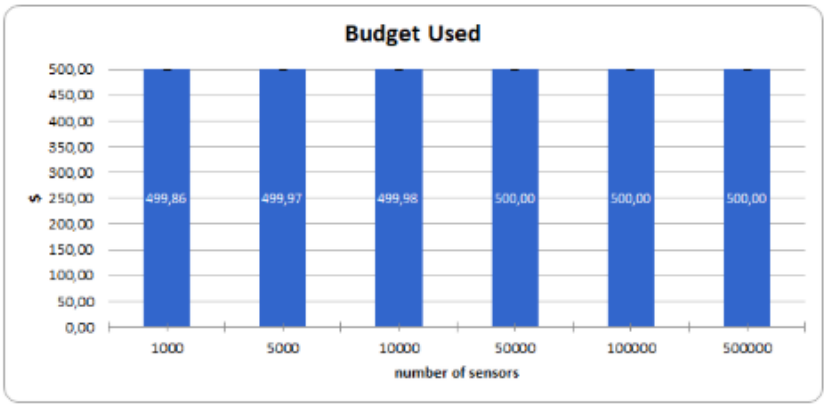

Figura 28 – Orçamento Utilizado para algoritmo guloso parâmetro maior V com 10 tipos de Sensores.

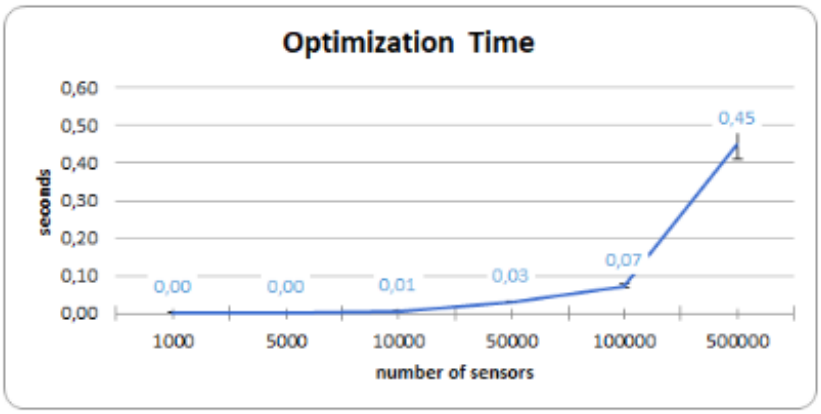

Figura 29 – Tempo de Otimização para algoritmo guloso parâmetro maior V com 20 tipos de Sensores.

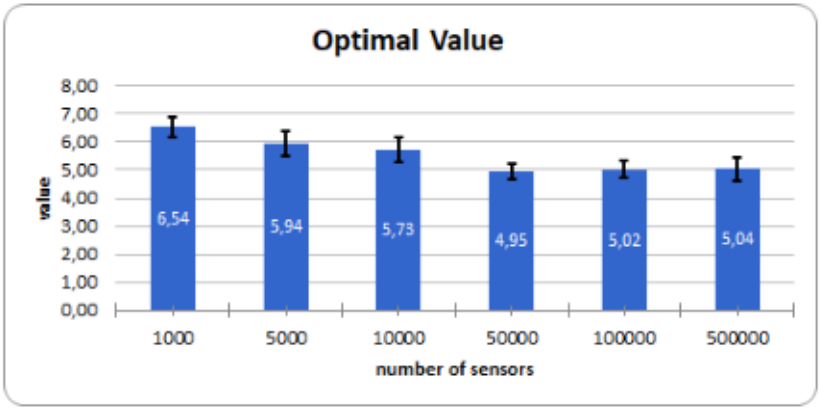

Figura 30 – Valor Ótimo para algoritmo guloso parâmetro maior V com 20 tipos de Sensores.

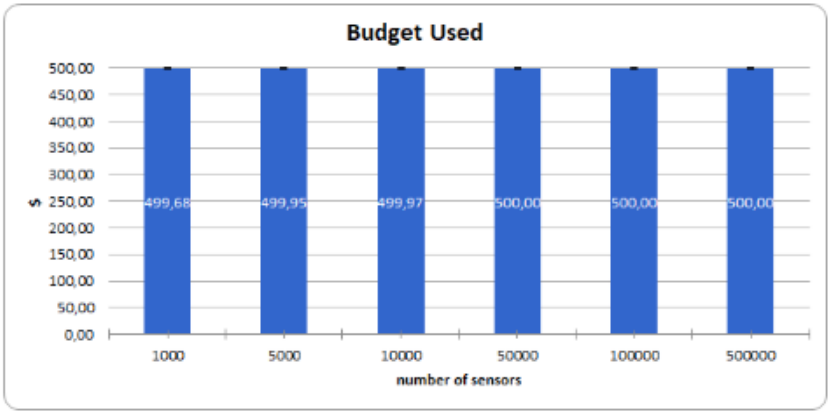

Figura 31 – Orçamento Utilizado para algoritmo guloso parâmetro maior V com 20 tipos de Sensores.

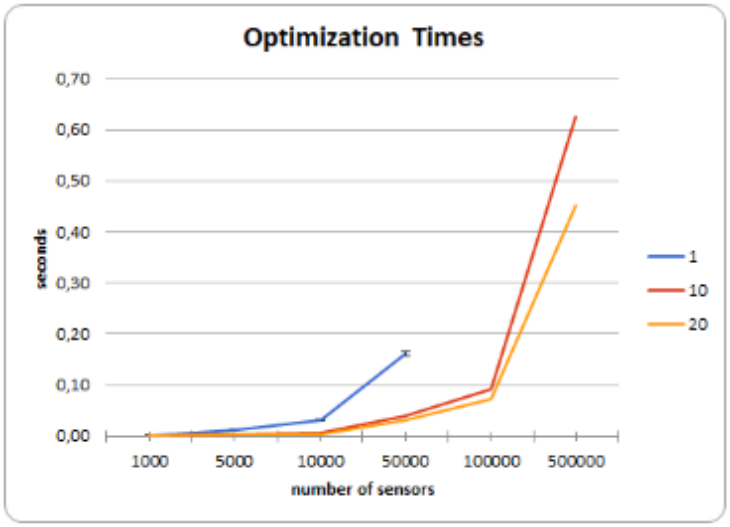

Figura 32 – Tempos de Otimização no Algoritmo Guloso de parâmetro maior V com 1, 10 e 20 Tipos de Sensores.

4.3.3 Algoritmo Guloso (Parâmetro: maior M)

A parametrização que busca o maior M foi a que apresentou melhores resultados dentre as abordagens gulosas, uma vez que, de fato, encontrou resultados bastante elevados em termos do índice de atendimento aos requisitos (Figuras 34, 37 e 40), e ainda obteve respostas que utilizavam mais de 95% do orçamento. Destaca-se como ponto muito forte para esta opção seu tempo de otimização bastante pequeno, menor que 1 segundo para todos os casos (Figuras 33, 36 e 39).

Desta maneira, mesmo para um cenário com um elevado número de sensores, a solução é bastante próxima (em termos da soma dos índices de atendimento dos requisitos) da solução ótima encontrada pelo *solver*, mas seu tempo de otimização é consideravelmente menor, representando uma diferença de mais de 10 segundos.

O algoritmo desenvolvido é bastante similar ao do parâmetro M/V e por isto pode ser encontrado no Apêndice B.

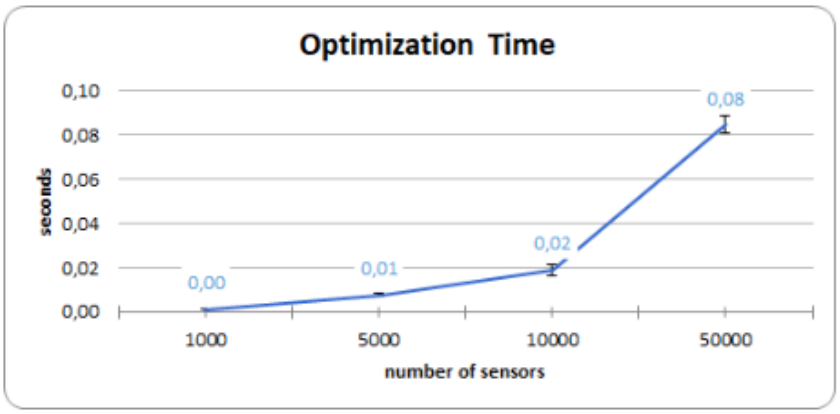

Figura 33 – Tempo de Otimização para algoritmo guloso parâmetro maior M com 1 tipo de Sensor.

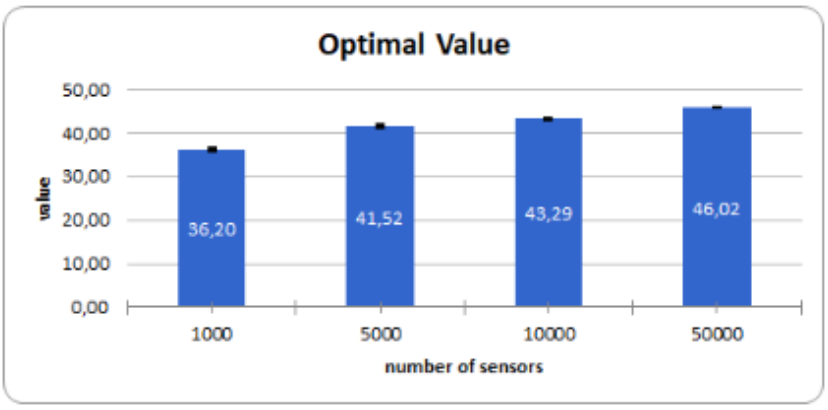

Figura 34 – Valor Ótimo para algoritmo guloso parâmetro maior M com 1 tipo de Sensor.

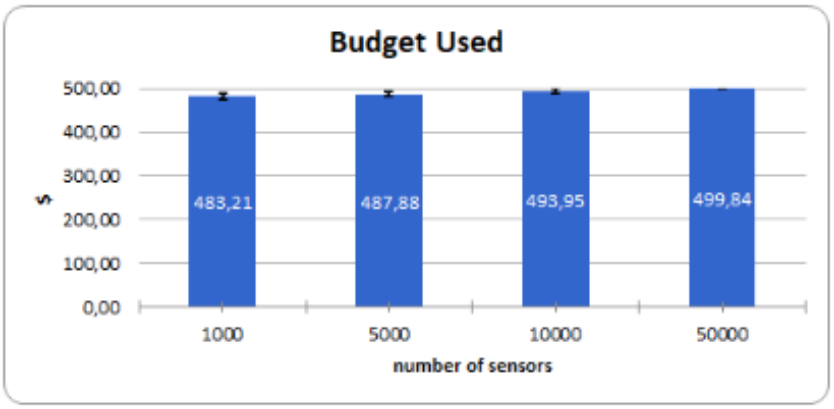

Figura 35 – Orçamento Utilizado para algoritmo guloso parâmetro maior M com 1 tipo de Sensor.

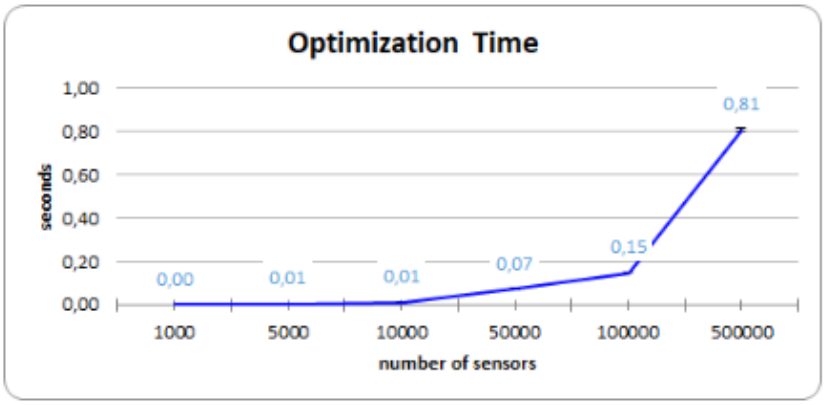

Figura 36 – Tempo de Otimização para algoritmo guloso parâmetro maior M com 10 tipos de Sensores.

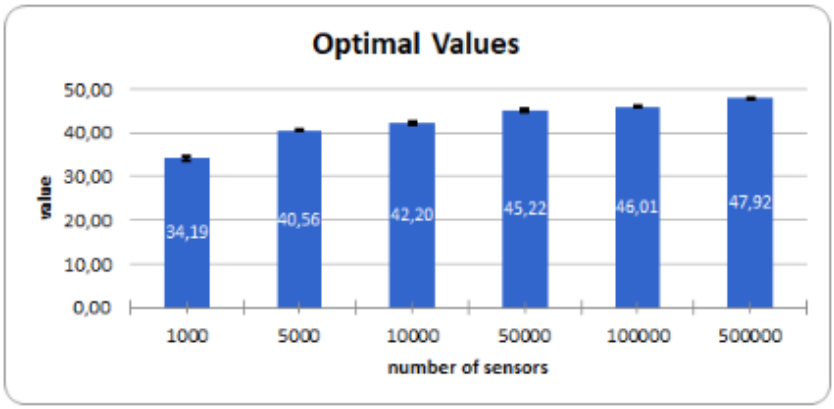

Figura 37 – Valor Ótimo para algoritmo guloso parâmetro maior M com 10 tipos de Sensores.

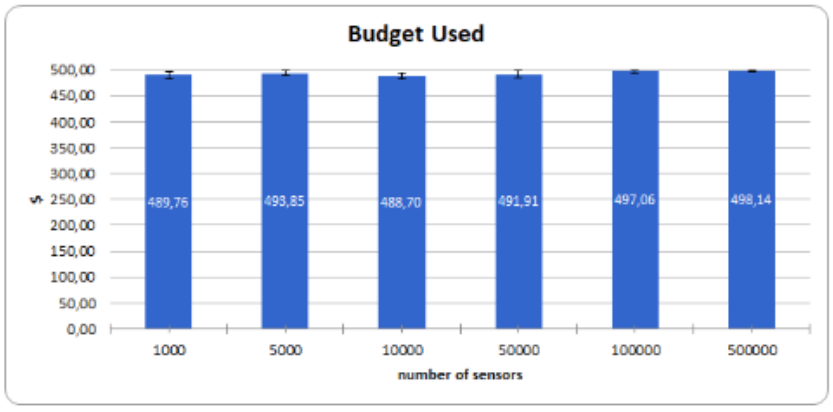

Figura 38 – Orçamento Utilizado para algoritmo guloso parâmetro maior M com 10 tipos de Sensores.

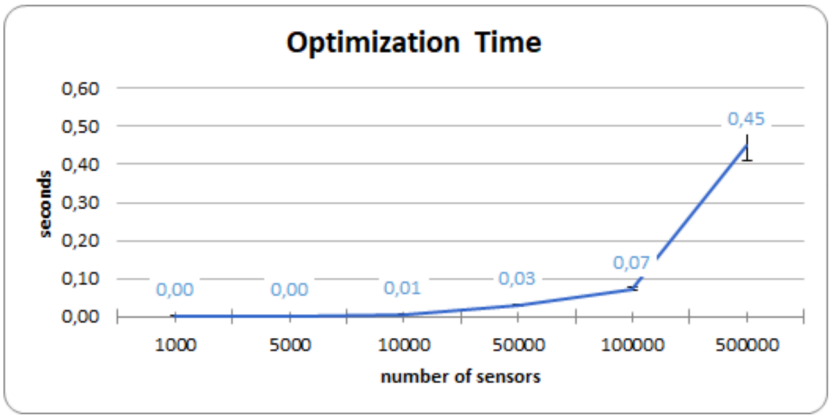

Figura 39 – Tempo de Otimização para algoritmo guloso parâmetro maior M com 20 tipos de Sensores.

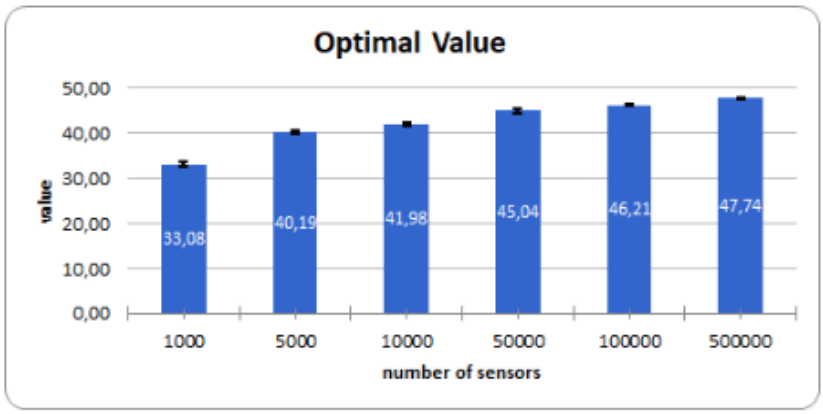

Figura 40 – Valor Ótimo para algoritmo guloso parâmetro maior M com 20 tipos de Sensores.

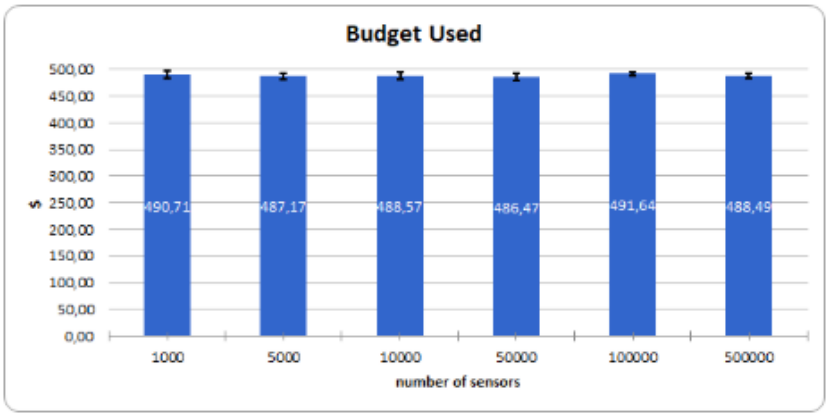

Figura 41 – Orçamento Utilizado para algoritmo guloso parâmetro maior M com 20 tipos de Sensores.

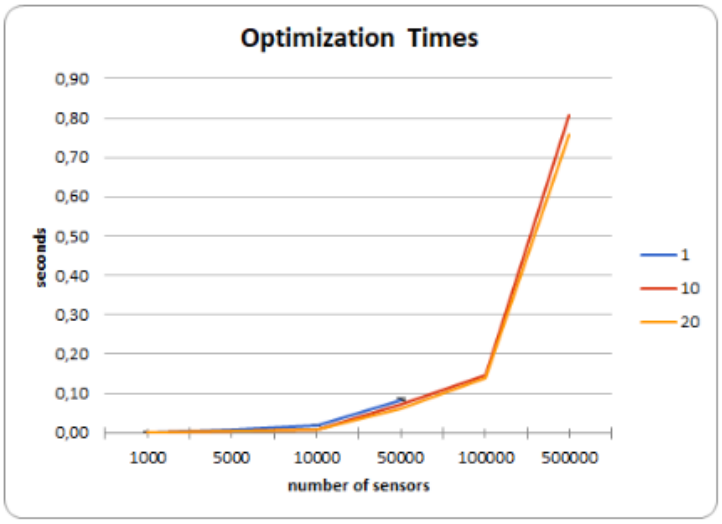

Figura 42 – Tempos de Otimização no Algoritimo Guloso de parâmetro maior M com 1, 10 e 20 Tipos de Sensores.

5 Conclusão

O presente trabalho apresentou um modelo matemático extensível, capaz de maximizar o atendimento a requisitos de entrada, tais como precisão, robustez e disponibilidade, para diferentes tipos de sensores indicados pelo usuário. Provou-se ainda que o problema apresentado é NP-Difícil, uma vez que o problema da mochila pode ser reduzido em tempo polinomial ao problema de seleção de sensores.

O trabalho convergiu para uma implementação de soluções, utilizando dois algoritmos. O primeiro utilizou um *solver* que emprega o algoritmo *branch-and-bound* para solução do problema, que em seu estado atual, permite a tomada de decisão quanto a seleção de sensores de maneira eficaz. O segundo utilizou um algoritmo guloso avaliando diferentes parâmetros, nomeadamente, índice de atendimento dos requisitos do usuário (M_{ij}), custo do sensor V_{ij} e a relação custo benefício (M_{ij}/V_{ij}). Ambos os algoritmos foram avaliados através de simulações Monte Carlo, com uma análise baseada em 3 métricas: tempo de otimização, valor da função obtido na solução ótima e orçamento utilizado na solução ótima. Visando a verificação e reprodução dos experimento, todos os códigos desenvolvidos no projeto se encontram disponíveis.

De uma maneira geral, a abordagem guloso apresentou tempos de otimização inferiores à solução por meio de *solver*. Sendo a abordagem gulosa baseada no índice de atendimento dos requisitos do usuário a que apresentou os melhores resultados para as instâncias testadas neste problema. O estratégia gulosa encontrou resultados próximos (aproximadamente 4% do valor obtido pelo algoritmo ótimo) com tempo de solução menor do que 1 segundo, quando comparado ao algoritmo otimizado. Esta diferença de tempo pode ser substancial na seleção do algoritmo a ser utilizado, uma vez que em casos com a massa de dados ainda maior que a avaliada, esta diferença tende a ser ainda mais expressiva e pode vir a ser um diferencial de uma futura aplicação.

Os resultados obtidos neste trabalho de conclusão de curso foram financiados por bolsa de iniciação científica com plano intitulado Desenvolvimento de algoritmos para seleção de estações de coleta baseado em parâmetros definidos por usuários em um sistema S²aaS no Programa Institucional de Bolsas de Iniciação Científica (PIBIC) da Fundação de Amparo a Ciência e Tecnologia do Estado de Pernambuco. Estes resultados foram também apresentados na Jornada de Iniciação Científica da FACEPE em 2017 e publicados como o artigo científico *Sensor Selection Model and Algorithms Based on User Requirements for Sensing as a Service* desta autora e de seus orientadores na Revista de Exatas e Tecnológicas em 2018.

5.1 Dificuldades Encontradas

A solução do problema para seleção de estações de coleta baseado em parâmetros definidos por usuários em um sistema S²aaS perpassa pela incorporação de uma elevada gama de diferentes requisitos de entrada a serem selecionados pelo usuário, configurando-se assim o maior desafio relacionado a este trabalho.

É de se destacar que estes requisitos podem apresentar diferentes desafios em relação à sua modelagem matemática, uma vez que podem ser caracterizados de maneiras distintas. Por exemplo, quando se trata do tipo de sensor que trata de temperatura, o usuário poderia estabelecer um intervalo mínimo e máximo para um requisito como Intervalo de Medição, ao passo que para um requisito como qualidade do enlace de comunicação, este poderia ser caracterizado de maneira categórica (boa, ruim, etc).

Esta grande variabilidade de parâmetros torna a modelagem matemática, mais especificamente o cálculo do fator de atendimento dos requisitos (M_{ij}), bastante complexa. Logo, verificou-se a necessidade de evoluir o modelo matemático em partes, tornando-o mais robusto ao passo que novas tipologias de requisitos são implementadas.

Além disto, a relação matemática M/V se tornou um desafio, uma vez que o algoritmo guloso com este parâmetro, que seria uma escolha bem comum para este tipo de problema, não apresentou bons resultados.

5.2 Trabalhos Futuros

O problema de seleção de sensores, do ponto de vista da abordagem do Problema da Mochila, pode e deve ainda ser abordado por soluções que levem em consideração a utilização de programação dinâmica, a fim de se identificar uma maneira que busque ao mesmo tempo, os resultados ótimos encontrados pelo *solver*, bem como o ínfimo tempo da abordagem gulosa implementada. Desta forma, os benefícios poderiam ser combinados.

Outro importante aspecto, é a parametrização de diferentes tipos de requisitos a fim de tornar a solução mais robusta e completa, de modo a abranger mais possibilidades de escolha por parte do usuário.

Referências

BARTHOLDI, J. J. The knapsack problem. In: *Building intuition*. [S.l.]: Springer, 2008. p. 19–31. Citado 2 vezes nas páginas 19 e 20.

BENGHANEM, M. Measurement of meteorological data based on wireless data acquisition system monitoring. *Applied energy*, Elsevier, v. 86, n. 12, p. 2651–2660, 2009. Citado na página 12.

CARVALHO, É. d. C. R. et al. Solução de problemas de otimização com restrições usando estratégias de penalização adaptativa e um algoritmo do tipo pso. Universidade Federal de Juiz de Fora (UFJF), 2014. Citado 2 vezes nas páginas 26 e 27.

DANTZIG, G. B. Discrete-variable extremum problems. *Operations research*, INFORMS, v. 5, n. 2, p. 266–288, 1957. Citado 2 vezes nas páginas 19 e 21.

DIFFIE, W.; HELLMAN, M. E. Multiuser cryptographic techniques. In: ACM. *Proceedings of the June 7-10, 1976, national computer conference and exposition*. [S.l.], 1976. p. 109–112. Citado na página 19.

EBRAHIMI, M. et al. An adaptive meta-heuristic search for the internet of things. *Future Generation Computer Systems*, Elsevier, v. 76, p. 486–494, 2017. Citado na página 17.

GANDOMI, A. H.; YANG, X.-S.; ALAVI, A. H. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with computers*, Springer, v. 29, n. 1, p. 17–35, 2013. Citado na página 26.

GUROBI Optimizer. s.d. <http://www.gurobi.com/products/gurobi-optimizer>. Accessed: 2018-12-12. Citado 2 vezes nas páginas 24 e 31.

KELLERER, H.; PFERSCHY, U.; PISINGER, D. Multidimensional knapsack problems. In: *Knapsack problems*. [S.l.]: Springer, 2004. p. 235–283. Citado na página 20.

LAGOUDAKIS, M. G. The 0-1 knapsack problem–an introductory survey. Citeseer, 1996. Citado 3 vezes nas páginas 21, 22 e 23.

MARTELLO, S.; TOTH, P. *Knapsack Problems, J*. [S.l.]: Wiley & Sons, Chichester, 1990. Citado na página 30.

MERKLE, R.; HELLMAN, M. Hiding information and signatures in trapdoor knapsacks. *IEEE transactions on Information Theory*, IEEE, v. 24, n. 5, p. 525–530, 1978. Citado na página 19.

MIXED-INTEGER Programming (MIP) - A Primer on the Basics. s.d. <http://www.gurobi.com/resources/getting-started/mip-basics>. Accessed: 2017-02-15. Citado 2 vezes nas páginas 24 e 25.

MORAIS, R. et al. A zigbee multi-powered wireless acquisition device for remote sensing applications in precision viticulture. *Computers and electronics in agriculture*, Elsevier, v. 62, n. 2, p. 94–106, 2008. Citado na página 12.

MURAWSKI, C.; BOSSAERTS, P. How humans solve complex problems: The case of the knapsack problem. *Scientific reports*, Nature Publishing Group, v. 6, p. 34851, 2016. Citado na página 21.

NETO, J. B. B. et al. Sensing in the collaborative internet of things. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 15, n. 3, p. 6607–6632, 2015. Citado na página 16.

PAULA, R. R. d. et al. Método de monte carlo e aplicações. Volta Redonda, 2017. Citado na página 32.

PERERA, C. et al. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 25, n. 1, p. 81–93, 2014. Citado 2 vezes nas páginas 12 e 15.

PERERA, C. et al. Sensing as a service model for smart cities supported by internet of things. *Transactions on emerging telecommunications technologies*, Wiley Online Library, v. 25, n. 1, p. 81–93, 2014. Citado 2 vezes nas páginas 15 e 16.

PERERA, C. et al. Sensor search techniques for sensing as a service architecture for the internet of things. *IEEE Sensors Journal*, IEEE, v. 14, n. 2, p. 406–420, 2014. Citado na página 17.

SARKER, R. A.; NEWTON, C. S. *Optimization modelling: a practical approach*. [S.l.]: CRC Press, 2007. Citado 2 vezes nas páginas 18 e 26.

SHENG, X. et al. Sensing as a service: Challenges, solutions and future directions. *IEEE Sensors journal*, IEEE, v. 13, n. 10, p. 3733–3741, 2013. Citado na página 16.

SILVANO, M.; PAOLO, T. *Knapsack problems: algorithms and computer implementations*. [S.l.]: John Wiley & Sons Hoboken, NJ, 1990. Citado 3 vezes nas páginas 18, 19 e 21.

SKIENA, S. Who is interested in algorithms and why?: lessons from the stony brook algorithms repository. *ACM SIGACT News*, Citeseer, v. 30, n. 3, p. 65–74, 1999. Citado na página 20.

SOUZA, É. L. d.; RAFAEL, E. A. L. Abordagens para resolver o problema da mochila 0/1. *REVISTA IGAPÓ-Revista de Educação Ciência e Tecnologia do IFAM*, v. 3, 2014. Citado na página 22.

TRUONG, C.; ROMER, K.; CHEN, K. Fuzzy-based sensor search in the web of things. In: IEEE. *Internet of Things (IOT), 2012 3rd International Conference on the*. [S.l.], 2012. p. 127–134. Citado na página 17.

WANG, W. et al. An experimental study on geospatial indexing for sensor service discovery. *Expert Systems with Applications*, Elsevier, v. 42, n. 7, p. 3528–3538, 2015. Citado 3 vezes nas páginas 15, 17 e 29.

ZASLAVSKY, A.; PERERA, C.; GEORGAKOPOULOS, D. Sensing as a service and big data. *arXiv preprint arXiv:1301.0159*, 2013. Citado na página 15.

ZIVIANI, N. et al. *Projeto de algoritmos: com implementações em Pascal e C*. [S.l.]: Thomson Luton, 2004. v. 2. Citado na página 21.

A Algoritimo Greedy com Parâmetro V

```
import traceback
from heapq import nlargest
import sys
import os
import random
import linecache
import datetime
import time
import numpy

global numDotipo
global listaItens

global usedBudget
global somaM

global done
global escolhidos

if len(sys.argv) == 7:
   n = int(sys.argv[1])
   m = int(sys.argv[2])
   B = int(sys.argv[3])
   s = int(sys.argv[4])
   random.seed(s) #inicializa o gerador de número aleatórios
   num_rep = int(sys.argv[5])
   numSensPedidos = int(sys.argv[6])

else:
   print("Informar: n m B s num_rep numSensPedidos")
   print("n = Sensores(i) por tipo")
   print("m = Numero de tipos(j)")
   print("B = Orçamento do usuário")
   print("s = Seed (Int)")
   print("num_rep = número de repetições")
   print("numSensPedidos = número total de sensores pedidos")
   exit(1)
```

```
dt = datetime.datetime.now()
dt = dt.strftime('%Y-%m-%d_%H-%M')
nomeDoDiretorio = str(n) + "_" + str(m) + "_" + str(B) + "_" + str(s) + "_"+
    str(num_rep) + "_" + str(numSensPedidos) + "_" + str(dt)
os.makedirs(nomeDoDiretorio)

def criarMatriz(linha, coluna):
   M = [[0 for x in range(coluna)] for y in range(linha)]
   return M

for num_da_rep in range(num_rep):

   usedBudget = 0
   somaM = 0

   arquivo = "/Users/nichenevercosa/SensorSelect/" + str(nomeDoDiretorio) +
       "/" + str(num_da_rep) + ".txt"
   arquivoRespostas = open(arquivo, "w")


   N = numpy.random.multinomial(numSensPedidos, [1.0/m]*m)

   print ("CRIANDO MATRIZES V, M e DivisaoMV\n")
   V = criarMatriz(n,m)
   M = criarMatriz(n,m)
   DivisaoMV = criarMatriz(n,m)

   for numDeTipos in range(m):
      numLinhas = random.sample(range(1,50001), n)
      numDoSensor = 0
      for numLinha in numLinhas:
         l = linecache.getline("file" + str(numDeTipos)+".csv", numLinha)
         tipo,indiceM,numValor,barraN = l.split(";")
         M[numDoSensor][numDeTipos] = round(float(indiceM),5)
         V[numDoSensor][numDeTipos] = round(float(numValor),2)
         numDoSensor += 1

   nrange = range(n)
   mrange = range(m)

   numDotipo = 0
```

```
print ("OTIMIZANDO\n")

tTime = time.time()
done = False
escolhidos = []
for numSenEscolhidos in N:
   done = False
   listaItens = []

   if (numSenEscolhidos > 0):
      for i in nrange:
         valorDiv = (V[i][numDotipo])
         item = ((i,numDotipo), valorDiv)
         listaItens.append(item)

      itensSorted = sorted(listaItens, key = lambda item: item[1], reverse =
          True)
      countEscolhidos = 0

      for maioresIndices in itensSorted:
         iSensor, jSensor = maioresIndices[0][0], maioresIndices[0][1]
         valorm = M[iSensor][jSensor]
         valorv = V[iSensor][jSensor]

         if ((usedBudget + valorv) <= B):
            countEscolhidos += 1
            sensorescolhido = (iSensor,jSensor)
            escolhidos.append(sensorescolhido)

            somaM += valorm
            usedBudget += valorv

         if (countEscolhidos == numSenEscolhidos):
            #done = True
            break

      if (countEscolhidos < numSenEscolhidos):
         done = True
      if (done):
         arquivoRespostas.write("Nao exite uma solucao para este problema")
         break
```

```
      numDotipo += 1

   tempoDeOtimizacaoTime = time.time() - tTime
   tempoDeOtimizacaoTime = str(tempoDeOtimizacaoTime)

   print("GUARDANDO RESPOSTAS \n")
   if (usedBudget <= B):

      arquivoRespostas.write("N = " + str(N) + "\n")
      arquivoRespostas.write("M = " + str(M) + "\n")
      arquivoRespostas.write("V = " + str(V) + "\n")
      arquivoRespostas.write("MV = " + str(DivisaoMV)+ "\n")
      arquivoRespostas.write("Tempo de otimizacao (time.time) = " +
          tempoDeOtimizacaoTime + "\n")
      arquivoRespostas.write('The best known bound on the optimal objective = ')
      arquivoRespostas.write(str(somaM)+ "\n")
      arquivoRespostas.write("O orcamento utilizado foi = "+ str(usedBudget)+
          "\n")
      arquivoRespostas.write("Lista dos sensores escolhidos = "+
          str(escolhidos))

   print ("\n---FIM da repeticao " + str(num_da_rep) + "---\n")

   arquivoRespostas.close()

print (" - - FIM - - \o/ ")
```

B Algoritimo Greedy com Parâmetro M

```
import traceback
from heapq import nlargest
import sys
import os
import random
import linecache
import datetime
import time
import numpy

global numDotipo
global listaItens

global usedBudget
global somaM

global done
global escolhidos

if len(sys.argv) == 7:
   n = int(sys.argv[1])
   m = int(sys.argv[2])
   B = int(sys.argv[3])
   s = int(sys.argv[4])
   random.seed(s) #inicializa o gerador de número aleatórios
   num_rep = int(sys.argv[5])
   numSensPedidos = int(sys.argv[6])

else:
   print("Informar: n m B s num_rep numSensPedidos")
   print("n = Sensores(i) por tipo")
   print("m = Numero de tipos(j)")
   print("B = Orçamento do usuário")
   print("s = Seed (Int)")
   print("num_rep = número de repetições")
   print("numSensPedidos = número total de sensores pedidos")
   exit(1)
```

```
dt = datetime.datetime.now()
dt = dt.strftime('%Y-%m-%d_%H-%M')
nomeDoDiretorio = str(n) + "_" + str(m) + "_" + str(B) + "_" + str(s) + "_"+
    str(num_rep) + "_" + str(numSensPedidos) + "_" + str(dt)
os.makedirs(nomeDoDiretorio)

def criarMatriz(linha, coluna):
   M = [[0 for x in range(coluna)] for y in range(linha)]
   return M

for num_da_rep in range(num_rep):

   usedBudget = 0
   somaM = 0

   arquivo = "/Users/nichenevercosa/SensorSelect/" + str(nomeDoDiretorio) +
       "/" + str(num_da_rep) + ".txt"
   arquivoRespostas = open(arquivo, "w")


   N = numpy.random.multinomial(numSensPedidos, [1.0/m]*m)

   print ("CRIANDO MATRIZES V, M e DivisaoMV\n")
   V = criarMatriz(n,m)
   M = criarMatriz(n,m)
   DivisaoMV = criarMatriz(n,m)

   for numDeTipos in range(m):
      numLinhas = random.sample(range(1,50001), n)
      numDoSensor = 0
      for numLinha in numLinhas:
         l = linecache.getline("file" + str(numDeTipos)+".csv", numLinha)
         tipo,indiceM,numValor,barraN = l.split(";")
         M[numDoSensor][numDeTipos] = round(float(indiceM),5)
         V[numDoSensor][numDeTipos] = round(float(numValor),2)
         numDoSensor += 1

   nrange = range(n)
   mrange = range(m)

   numDotipo = 0
```

```
print ("OTIMIZANDO\n")

tTime = time.time()
done = False
escolhidos = []
for numSenEscolhidos in N:
   done = False
   listaItens = []

   if (numSenEscolhidos > 0):
      for i in nrange:
         valorDiv = (M[i][numDotipo])
         item = ((i,numDotipo), valorDiv)
         listaItens.append(item)

      itensSorted = sorted(listaItens, key = lambda item: item[1], reverse =
          True)
      countEscolhidos = 0

      for maioresIndices in itensSorted:
         iSensor, jSensor = maioresIndices[0][0], maioresIndices[0][1]
         valorm = M[iSensor][jSensor]
         valorv = V[iSensor][jSensor]

         if ((usedBudget + valorv) <= B):
            countEscolhidos += 1
            sensorescolhido = (iSensor,jSensor)
            escolhidos.append(sensorescolhido)

            somaM += valorm
            usedBudget += valorv

         if (countEscolhidos == numSenEscolhidos):
            #done = True
            break

      if (countEscolhidos < numSenEscolhidos):
         done = True
      if (done):
         arquivoRespostas.write("Nao exite uma solucao para este problema")
         break
```

```
      numDotipo += 1

   tempoDeOtimizacaoTime = time.time() - tTime
   tempoDeOtimizacaoTime = str(tempoDeOtimizacaoTime)

   print("GUARDANDO RESPOSTAS \n")
   if (usedBudget <= B):

      arquivoRespostas.write("N = " + str(N) + "\n")
      arquivoRespostas.write("M = " + str(M) + "\n")
      arquivoRespostas.write("V = " + str(V) + "\n")
      arquivoRespostas.write("MV = " + str(DivisaoMV)+ "\n")
      arquivoRespostas.write("Tempo de otimizacao (time.time) = " +
          tempoDeOtimizacaoTime + "\n")
      arquivoRespostas.write('The best known bound on the optimal objective = ')
      arquivoRespostas.write(str(somaM)+ "\n")
      arquivoRespostas.write("O orcamento utilizado foi = "+ str(usedBudget)+
          "\n")
      arquivoRespostas.write("Lista dos sensores escolhidos = "+
          str(escolhidos))

   print ("\n---FIM da repeticao " + str(num_da_rep) + "---\n")

   arquivoRespostas.close()

print (" - - FIM - - \o/ ")
```