UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

José Ferreira Reis Fonseca

Formulação Linear Inteira e Heurísticas para o problema da K-Cobertura e M-Conectividade em Redes de Sensores Sem Fio

Belo Horizonte
2023

José Ferreira Reis Fonseca

Formulação Linear Inteira e Heurísticas para o problema da K-Cobertura e M-Conectividade em Redes de Sensores Sem Fio

Versão Final

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Thiago Ferreira de Noronha
Coorientador: Iago Augusto de Carvalho

Belo Horizonte
2023

Fonseca, José Ferreira Reis Fonseca.

F676f Formulação linear inteira e heurísticas para o problema da K-cobertura e M-conectividade em redes de sensores sem fio. [recurso eletrônico] / José Ferreira Reis Fonseca– 2023.
1 recurso online (43 f. il., color.) : pdf.

Orientador: Thiago Ferreira de Noronha
Coorientador: Iago Augusto de Carvalho.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f.41-43.

1. Computação – Teses. 2. Redes de sensores sem fio – Teses. 3. Programação linear – Teses. 4. Programação heurística – Teses. I. Noronha, Thiago Ferreira de. II. Carvalho, Iago Augusto de. III Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*22(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais – ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Formulação Linear Inteira e Heurísticas para o problema da K-Cobertura e M-Conectividade em redes de sensores sem fio

JOSÉ FERREIRA REIS FONSECA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Thiago Ferreira de Noronha - Orientador
Departamento de Ciência da Computação - UFMG

Documento assinado digitalmente
gov.br
IAGO AUGUSTO DE CARVALHO
Data: 16/07/2024 17:39:30-0300
Verifique em https://validar.iti.gov.br

Prof. Iago Augusto de Carvalho - Coorientador
Ciência da Computação - UNIFAL-MG

Documento assinado digitalmente
gov.br
FERNANDA SUMIKA HOJO DE SOUZA
Data: 17/07/2024 19:41:18-0300
Verifique em https://validar.iti.gov.br

Profa. Fernanda Sumika Hojo de Souza
Departamento de Computação - UFOP

Documento assinado digitalmente
gov.br
MARCIO COSTA SANTOS
Data: 19/07/2024 09:09:33-0300
Verifique em https://validar.iti.gov.br

Prof. Márcio Costa Santos
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 1 de setembro de 2023.

Dedico este trabalho a todos que trilham caminhos incertos, e buscam sempre as melhores alternativas.

Agradecimentos

Agradeço primeira e principalmente a minha esposa, Lucianna, que teve de me ouvir falar palavras sem sentido e números por todos esses anos e frequentemente realinhar meus pensamentos embolados. Agradeço aos meus pais, Conceição e Sérgio, por sempre terem me garantido os recursos para dedicar tanto do meu tempo a uma ciência cara e tão distante do nosso cotidiano - em especial me treinado para ter uma curiosidade obstinada, vital para quem vira a noite procurando bugs em software. Agradeço aos meus avós Mercês e Acrimar, Eunice e José, que desde cedo estiveram me ensinando a fazer o que penso, mexer no que me desperta curiosidade e andar até onde minhas pernas podem me levar, indo mais longe faça chuva ou faça sol. Agradeço aos meus orientadores, Thiago e Iago, que tiveram a paciência de me ensinar o que agora sei serem conceitos básicos de postura e escrita acadêmica, estatística e programação linear inteira, através de uma pandemia e do complexo mundo que emergiu depois. Agradeço aos meus muitos amigos e grande família, que mantém minha cabeça no lugar e coração batendo forte.
E agradeço aos que vieram antes e abriram o caminho para que alguém como eu chegasse onde cheguei.

"*Ubuntu*"
(Nações Zulu e Xhosa)

Resumo

Um sensor sem fio é um dispositivo capaz de coletar dados de um ou mais pontos de interesse (*points of interest* - POIs) e transmitir os dados coletados para outros sensores sem o uso de fios. Uma rede de sensores sem fio (*wireless sensor network* - WSN) pode ser estruturada de forma que seus sensores continuamente coletem dados de um conjunto de POIs e os transmitam de sensor em sensor até um subconjunto de sensores base denominados *sinks*. Por vezes, a cobertura em algum POI só pode ser garantida se vários sensores estiverem posicionados de forma a coletar seus dados. A propriedade de K-Cobertura é atribuída a WSNs em que cada POI tem seus dados coletados por K ou mais sensores e transmitidos para algum sensor base. Sensores estão sujeitos a falhas que podem impedi-los de transmitir ou coletar dados de POIs. WSNs robustas são capazes de coletar dados de todos os POIs mesmo após falhas em vários de seus sensores. Uma WSN tem a propriedade de M-Conectividade quando seus sensores são capazes de coletar dados de todos os POIs e transmiti-los para algum sensor base mesmo após $M - 1$ sensores sofrerem falhas. O problema da K-Cobertura e M-Conectividade (KCMC) consiste em determinar o menor subconjunto de sensores de uma WSN que sejam suficientes para garantir as propriedades de K-Cobertura e M-Conectividade. Neste trabalho são apresentados dois programas lineares inteiros (*integer linear programs* - ILPs) que formulam o KCMC. Também são apresentadas quatro heurísticas para esse mesmo problema. As quatro heurísticas apresentadas neste trabalho são baseadas na meta-heurística *fix-and-optimize*, em que um método heurístico é utilizado para reduzir o problema original. O sub-problema resultante é então resolvido por um algoritmo exato. As heurísticas apresentadas são diferenciadas pelo método heurístico empregado, mas todas utilizam o algoritmo *branch-and-bound* com um dos ILPs como método exato. O desempenho de cada heurística foi avaliado por meio de experimentos computacionais, permitindo comparação estatística para determinar a heurística que produz os melhores resultados.

Palavras-chave: redes de sensores sem fio; programação linear inteira; cobertura; conectividade; *fix-and-optimize*.

Abstract

A wireless sensor is a device capable of collecting data from one or more Points of Interest (POIs) and wirelessly transmitting the collected data. A wireless sensor network (WSN) may be deployed in such a way that each sensor continuously collects data from a set of POIs, and all data is transmitted to a subset of sensors called *sinks*. In some situations, the coverage of a POI can only be assured if several sensors collect its data. The property of K-Coverage is attributed to WSNs in which each POI may have its data collected by at least K sensors, with all data being transmitted to one or more sinks. A sensor may suffer failures that compromise its capabilities for data collection and transmission. The property of M-Connectivity is attributed to *robust* WSNs, in which data can be collected in all POIs and transmitted to sinks even if $M-1$ sensors suffer failures simultaneously. The K-Coverage M-Connectivity (KCMC) problem consists of computing the minimal subset of sensors in a WSN that suffice for the K-Coverage and M-Connectivity problems. In this master's thesis, two Integer Linear Programs (ILPs) and four heuristics are presented for the KCMC. The four heuristics presented are based on fix-and-optimize meta-heuristics, when an heuristic method is used to reduce the original problem. The resulting subproblem is then solved by an exact algorithm. Each heuristic differs by the heuristic method employed, but all use the branch-and-bound algorithm with one of the presented ILPs as the exact method. Computational experiments were used to evaluate the heuristics and determine the one that more often yields the best results.

Keywords: wireless sensor networks; integer linear programming; coverage; connectivity; fix-and-optimize.

Lista de Figuras

Lista de Tabelas

Lista de Pseudocódigos

Sumário

Capítulo 1

Introdução

Um *sensor* é um dispositivo capaz de coletar dados sobre o espaço ao seu redor. Dizemos que o espaço do qual um sensor pode coletar dados é *coberto* pelo sensor. Sensores *sem fio* utilizam tecnologias como ondas eletromagnéticas para transmitir dados para outros dispositivos. As capacidades de cobertura e comunicação de um sensor sem fio podem ser efetivas apenas até dadas distâncias, de forma que um único dispositivo pode não ser suficiente para cobrir uma determinada área por inteiro.

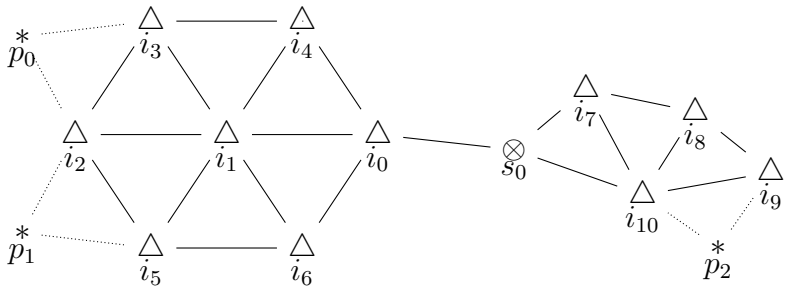
Chamamos uma área que pode ser coberta por um sensor de *ponto de interesse* (*Point of Interest* - POI). Um POI é uma abstração em que toda uma área ou um conjunto de objetos são imaginados como um ponto. Neste trabalho consideramos POIs imóveis que continuamente produzem informações que devem ser coletadas por sensores. Um único sensor é capaz de cobrir todos os POIs que estejam suficientemente próximos, sendo necessário outros sensores para cobrir POIs mais distantes.

Wireless Sensor Networks (*redes de sensores sem fio* - WSNs) são grupos de sensores que transmitem dados entre si sem o uso de fios. Um sensor pode transmitir dados que coletou de algum POI para outro sensor, que pode transmitir esses mesmos dados para outro sensor. Uma WSN pode conter sensores que não cobrem nenhum POI, sendo utilizados apenas para retransmissão de dados. Um ou mais sensores numa WSN podem ter capacidades ou recursos não disponíveis para outros sensores, como conexões diretas com a Internet ou redes de satélites [15]. Esses sensores são chamados de *sinks* (*bases*). Uma WSN pode ser utilizada para cobrir um dado conjunto de POIs e transmitir os dados coletados para uma ou mais *sinks* [12, 31, 34].

O uso de sensores sem fio permite flexibilidade e facilidade na implantação de uma rede. Sensores sem fio podem ser jogados de aviões sobre acidentes geográficos e outros locais de difícil acesso [15] ou podem ser posicionados de forma a contornar obstáculos na transmissão de dados [28]. Essas características de WSNs são desejáveis em várias indústrias [1, 21], especialmente em setores como agricultura de precisão [30], monitoramento ambiental [27], e detecção de ameaças [3, 4].

Um único sensor pode não ser suficiente para coletar toda a informação de um mesmo POI de forma confiável [22]. Os componentes em um sensor usados para coleta de dados podem não ser suficientemente precisos para certas aplicações [33] como identificação de alvos e invasões [4]. Em casos como esses, é necessário o uso de vários sensores para cobrir um mesmo

Figura 1: Uma WSN em que sensores estão representados por triângulos, POIs por asteriscos e a única sink por ⊗.



Fonte: Autores.

POI.

Falhas em sensores podem comprometer a capacidade de uma WSN de transmitir dados de POIs para uma *sink*. Sensores estão sujeitos a defeitos de fabricação, desgaste de componentes, exaustão de bateria ou interrupção no fornecimento de energia, sabotagem e outros eventos adversos que comprometem sua capacidade de coletar ou transmitir dados [3]. Dizemos que um sensor que se torna incapaz de coletar ou transmitir dados sofre uma falha. Dizemos que uma WSN que não é capaz de transmitir dados de todos os POIs para alguma *sink* devido a falhas em um ou mais de seus sensores foi *comprometida*. Dado um POI coberto por apenas um sensor de uma WSN, se esse sensor sofrer uma falha, não será possível coletar e transmitir os dados do POI para uma *sink*. Mesmo a falha de um sensor que não cobre nenhum POI pode comprometer uma WSN, como no caso representado na figura 1. Nesta figura, três POIs são representados por asteriscos, a única *sink* está representada pelo símbolo ⊗ e onze sensores estão representados por triângulos. Dois sensores que são capazes de transmitir dados entre si estão conectados por uma linha. Se o sensor i_0 sofrer uma falha, a WSN representada na figura 1 será comprometida, pois não será capaz de transmitir os dados coletados nos POIs p_0 e p_1 para a *sink* s_0.

Uma WSN é *robusta* quando é capaz de transmitir os dados coletados em todos os POIs para alguma *sink* mesmo após falhas em um ou mais de seus sensores. A WSN ilustrada na figura 1 é capaz de transmitir os dados coletados em todos os seus POIs mesmo em caso de falhas simultâneas nos sensores i_3, i_4, i_5, i_6, i_7, i_8, e i_9. Contudo, em caso de falha no sensor i_0, a WSN será comprometida. Ao avaliar se uma WSN é robusta, consideramos apenas a menor quantidade de sensores que precisariam sofrer falhas simultâneas para comprometer a WSN.

Pode ser desejável construir WSNs com a menor quantidade possível de sensores. Em certas aplicações, o custo de obtenção e implantação de sensores pode ser relevante e proporcional à quantidade de sensores utilizados. A computação do menor subconjunto dos sensores de uma WSN que são suficientes para que esta seja robusta e com vários sensores cobrindo cada POI tem complexidade exponencial [22].

O problema da K-Cobertura e M-Conectividade (*K-Coverage M-Connectivity* - KCMC) consiste na determinação do menor subconjunto de sensores de uma WSN que são suficientes para que a rede seja robusta e com K ou mais sensores cobrindo cada POI [18]. Cada POI em

uma WSN com a *propriedade de K-Cobertura* é coberto por ao menos K sensores e seus dados podem ser transmitidos para alguma *sink*. Quando todos os POIs de uma WSN tem a propriedade de K-Cobertura, dizemos que a WSN como um todo tem esta propriedade. Uma WSN tem a *propriedade de M-Conectividade* quando é possível remover $M - 1$ de seus sensores e, ainda assim, os dados coletados em todos os POIs podem ser transmitidos para *sinks*. Neste trabalho, apresentamos uma nova variante do problema KCMC descrita por meio de dois programas lineares inteiros (*Integer Linear Programs* - ILP). Diferentemente de outras variantes do problema KCMC na literatura, a definição de conectividade descrita neste trabalho é a mesma do teorema de Menger [9, 26]. Este trabalho também apresenta quatro heurísticas para o problema KCMC, todas baseadas no algoritmo de Dinitz [10, 11] e na meta-heurística *fix-and-optimize* [8, 16].

1.1 Objetivos

O objetivo deste trabalho é descrever uma variante do problema KCMC por meio de ILPs e propor heurísticas para a variante. A variante apresentada neste trabalho define a propriedade de M-Conectividade de forma similar ao teorema de Menger [9, 26]. Este teorema atribui a propriedade de M-Conectividade a todas as WSNs que não são comprometidas mesmo após a remoção de qualquer subconjunto de $M - 1$ de seus sensores. No trabalho de Gupta e coautores [18], WSNs tem a propriedade de M-Conectividade apenas se cada sensor estiver conectado a M ou mais sensores. Nosso trabalho propõe quatro heurísticas, testadas por meio de experimentos computacionais utilizando instâncias criadas seguindo a metodologia de Gupta e coautores [18].

Capítulo 2

Trabalhos Relacionados

O problema da K-Cobertura [33] é estudado independentemente do problema KCMC. O problema consiste em determinar subconjuntos minimais dos sensores de uma WSN que são suficientes para garantir a propriedade de K-Cobertura. No trabalho [22] os autores demonstraram que o problema da K-Cobertura pertence a classe de complexidade NP-Completo.

Os autores de [29] propõem uma variante do problema da K-Cobertura na qual POIs e sensores são posicionados em uma grade de formato regular. O trabalho apresenta um ILP que formula o problema, uma heurística de busca local e um algoritmo genético. Experimentos computacionais obtiveram soluções ótimas para grades de até $14 \cdot 14 = 196$ POIs e sensores. O algoritmo genético proposto apresenta desempenho superior ao da heurística de busca local em grades de até $50 \cdot 50 = 2500$ POIs e sensores.

Uma variante do problema da K-Cobertura proposta no trabalho [33] considera que cada sensor tem uma bateria de capacidade limitada e pode temporariamente interromper suas atividades para poupar a carga da bateria. Uma solução dessa variante é uma programação de atividades de sensores que maximiza o tempo em que cada POI permaneça coberto por ao menos K sensores diferentes enquanto seus dados são comunicados para alguma *sink*. Em uma outra variante do problema da K-Cobertura apresentada em [33] um sensor pode escolher cobrir apenas um subconjunto dos POIs em seus arredores, conservando a carga de sua bateria. Os autores demonstraram que, nessa variação, o problema pode ser resolvido em tempo polinomial com um algoritmo de fluxo em redes. Outra variante em que um sensor sempre deve cobrir todos os POIs possíveis a todo tempo pertence à classe de complexidade NP-Completo. Os autores do trabalho [33] propuseram um algoritmo aproximativo e uma heurística *ad-hoc* para essa última variante.

O problema da M-Conectividade consiste em determinar subconjuntos minimais dos sensores de uma WSN que são suficientes para garantir a propriedade de M-Conectividade. Este problema foi abordado recentemente em trabalhos como [2, 5, 6, 12, 17, 20, 23] e [29]. No trabalho [18], os autores tratam uma variante do problema em que uma WSN tem a propriedade de M-Conectividade quando cada um de seus sensores está conectado a exatamente M outros sensores. Dessa forma, todo sensor poderá transmitir dados para algum outro mesmo no caso de falhas em $M - 1$ de seus M vizinhos. A propriedade de M-Conectividade é definida da mesma forma em outros trabalhos como [2, 12, 17, 20] e [29]. Na WSN ilustrada na figura 1, cada

sensor está conectado a dois ou mais outros sensores. Contudo, no caso de falha no sensor i_0, não seria possível comunicar dados dos POIs p_0 e p_1 para a *sink* s_0. Neste trabalho, a propriedade é atribuída a todas as WSNs em que $M-1$ sensores podem ser removidos e, ainda assim, os dados coletados de todos os POIs podem ser transmitidos para alguma *sink*. Esta definição é similar a utilizada no teorema de Menger [9, 26].

Trabalhos relacionados propõem algoritmos para o problema KCMC inspirados em fenômenos naturais. O trabalho [29] propõe um algoritmo genético mencionado em outros trabalhos. Os autores de [18] propuseram um algoritmo genético que aprimora o modelo proposto por [29] com novos operadores evolutivos. No trabalho [20] os autores propõem heurísticas baseadas no algoritmo de busca de gravitação oposta (do inglês, *Oppositional Gravitational Search algorithm*), enquanto os autores de [17] propuseram um algoritmo de otimização baseado em biogeografia. As duas soluções propostas obtiveram melhores resultados que [29].

Trabalhos relacionados propõem variações do problema KCMC e formas de obter soluções para as mesmas. Uma variante do problema KCMC proposta no trabalho [2] considera que cada sensor tem uma bateria de capacidade limitada e pode temporariamente interromper suas atividades para poupar a carga da bateria. Nessa variante, cada sensor deve permanecer conectado com M outros sensores a todo tempo. O trabalho apresenta algoritmo de convolução e três heurísticas gulosas. Variantes do problema KCMC que consideram espaços bidimensionais e tridimensionais foram estudadas nos trabalhos [6, 17, 18]. O algoritmo IIC (do inglês, *Immigrant Imperialist Competitive algorithm*) proposto em [6] obteve resultados superiores aos de [17] e [18] em experimentos com a variante bidimensional. Uma lista mais compreensiva de variantes e algoritmos para o problema KCMC pode ser encontrada na revisão da literatura realizada por [12].

A variante da propriedade de M-Conectividade considerada neste trabalho é similar a do teorema de Menger [9, 26]. O teorema de Menger atribui a propriedade de M-Conectividade a WSNs nas quais é possível transmitir dados de POIs para *sinks* mesmo após a remoção de $M-1$ sensores. O teorema de Menger não exige que todos os sensores da WSN possam se transmitir com exatamente M sensores.

As heurísticas propostas no presente trabalho são baseadas na meta-heurística *fix-and-optimize* (termo em inglês para *fixar-e-otimizar*) [8, 13, 14, 16, 25]. A meta-heurística *fix-and-optimize* busca viabilizar o uso de algoritmos exatos cujo tempo de processamento para instâncias maiores que certo tamanho seria muito longo. Para isso, a meta-heurística divide o processo de solução de instâncias de problemas como o KCMC em duas etapas. Na primeira etapa, são empregadas heurísticas para remover (ou *fixar*) elementos do problema, computando um *sub-problema* da instância original. Na segunda etapa, são empregados algoritmos exatos de otimização, que têm como entrada o sub-problema computado. Neste trabalho, as heurísticas empregadas na primeira etapa buscam computar subconjuntos dos sensores, e apenas os sensores desses subconjuntos são considerados na segunda etapa. O algoritmo exato empregado é o *branch-and-bound* [32], configurado com um dos ILPs propostos nesse trabalho.

Capítulo 3

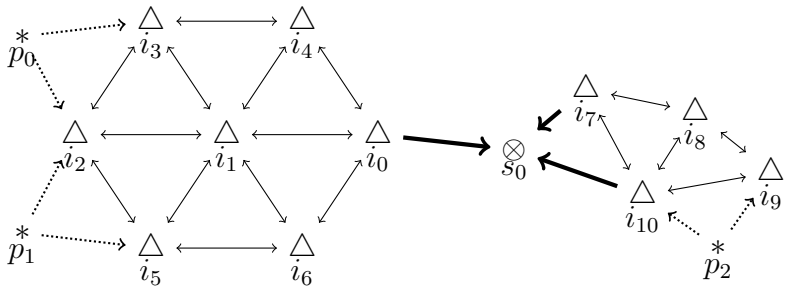
Definição do Problema e ILP

Uma WSN com uma ou mais *sinks* pode ser modelada como uma WSN com uma única *sink*. Seja Θ o conjunto de *sinks*. Seja s_0 uma *sink* imaginária ou *virtual* que não cobre nenhum POI. Pode-se dizer que s_0 é capaz de se comunicar com todas as *sinks* em Θ e nenhum outro sensor. Como os únicos dispositivos capazes de transmitir dados para s_0 são as *sinks* do conjunto Θ, se quaisquer dados foram transmitidos para s_0, esses dados têm de ter sido antes transmitidos para alguma *sink* do conjunto Θ. Assim, uma WSN em que sensores coletam dados em todos os POIs e os transmitem para s_0 é equivalente a uma WSN em que sensores transmitem os dados coletados para uma ou mais *sinks* em Θ. A definição do problema KCMC, a modelagem de uma WSN como uma instância do problema e os ILPs apresentados neste trabalho utilizam o recurso de *sink* virtual s_0.

Uma WSN pode ser modelada como uma instância do problema KCMC. Seja P o conjunto de POIs e I o conjunto de sensores de uma WSN. Seja Θ o conjunto de *sinks*. Se $|\Theta| = 1$, então s_0 é a única *sink*. Se $|\Theta| > 1$, então s_0 é a *sink* virtual e $I = I \bigcup \Theta$ é o conjunto de sensores acrescido das *sinks* em Θ. Seja $W(P \bigcup I \bigcup s_0, A)$ um grafo em que os POIs, sensores e s_0 da WSN são os vértices e A é o conjunto de arestas. As arestas em A são direcionadas. As arestas em A_K são sempre iniciadas em POIs e terminadas em sensores. O subconjunto $A_M \subset A$ contém arestas (i, j) que ligam todos os pares $i, j \in I$ de sensores capazes de se comunicar diretamente. Neste trabalho, a comunicação entre sensores é simétrica, de forma que para cada aresta $(i, j) \in A_M$ existe também a aresta inversa $(j, i) \in A_M$. O subconjunto $A_S \subset A$ contém arestas (i, s_0) que ligam sensores capazes de se comunicar com s_0 diretamente. Não existem arestas iniciadas em s_0. A figura 2 ilustra um grafo como W, sendo as arestas do conjunto A_K representadas por setas com linhas pontilhadas, as de A_M por setas bidirecionais e as de A_S por setas em negrito. A tupla $\langle W, K, M \rangle$, que contém o grafo W e os valores dos parâmetros K e M é uma instância do problema KCMC.

O problema KCMC determina subconjuntos minimais dos sensores da instância na entrada, cada conjunto com sensores suficientes para as propriedades de K-Cobertura e M-Conectividade. A instância na entrada consiste numa tupla $\langle W, K, M \rangle$ composta pelo grafo $W(P \bigcup I \bigcup s_0, A)$ e os parâmetros K e M. Soluções na saída do problema são conjuntos minimais de sensores, subconjuntos dos sensores da instância na entrada. Se algum sensor for removido de um conjunto minimal computado, uma ou ambas as propriedades de K-Cobertura e M-Conectividade

Figura 2: Uma WSN em que o conjunto A_K é representado por setas com linhas pontilhadas, A_M por setas bidirecionais e A_S por setas em negrito.



Fonte: Autores

serão comprometidas. Dado um conjunto minimal que soluciona o problema de forma exata, não é possível que exista outro conjunto com menos sensores e que ainda sejam suficientes para as duas propriedades. A maior quantidade possível de sensores no conjunto minimal na saída é a quantidade total de sensores da instância na entrada, por se tratar de um subconjunto. A menor quantidade possível de sensores no conjunto minimal na saída é o valor do parâmetro K do problema KCMC. Uma solução com menos de K sensores não permite que cada POI seja coberto por ao menos K sensores diferentes. É possível que uma instância do KCMC tenha uma solução com exatamente K sensores se cada um cobrir todos os POIs e se comunicar diretamente com s_0, tendo, portanto, K cobertura e K conectividade. Neste trabalho, $K \geq M$, de forma que K-conectividade é sempre suficiente para atender ao parâmetro M do problema KCMC.

Os dois ILPs apresentados neste trabalho diferenciam-se pela forma como tratam *fluxos*. Seja um caminho em um grafo uma sequência de vértices, em que cada vértice da sequência está na origem de uma aresta que o conecta diretamente com o próximo vértice da sequência. O primeiro vértice do caminho não é destino de nenhuma aresta na sequência, e o último não é origem de nenhuma aresta na sequência. Dois caminhos são ditos distintos se não compartilham vértices. Este trabalho se refere apenas a caminhos iniciados em algum POI $p \in P$ e finalizados em s_0, de forma que o primeiro sensor no caminho $\phi(p)$ é capaz de cobrir o POI $p \in P$ e o último é capaz de se comunicar com s_0. Cada POI envia uma unidade de *fluxo* por cada caminho que tem origem no POI. Assim, s_0 recebe uma unidade de fluxo para cada POI que esteja na origem de um caminho finalizado em s_0. O ILP fluxo-único computa sensores o bastante para que s_0 receba ao menos $M * |P|$ unidades de fluxo, sendo que cada POI está na origem de ao menos M caminhos diferentes. Uma árvore formada pelos sensores de caminhos iniciados em cada POI e finalizados em s_0 é um subconjunto dos sensores com M-Conectividade para $M = 1$. Assim, um conjunto de M árvores que não compartilhem sensores entre si contém sensores o bastante para a propriedade de M-Conectividade. O ILP multi-fluxo computa M árvores que não compartilham sensores entre si, mas cada árvore contém os sensores de caminhos de cada POI até s_0. Os dois ILPs utilizam o algoritmo de Dinitz para máximo fluxo em grafos e podem ser computados pelo algoritmo *branch-and-bound* [24, 32] implementado no pacote de software comercial Gurobi [19].

3.1 ILP Multi-fluxo

O ILP multi-fluxo tem na entrada uma tupla $\langle W, K, M \rangle$. Seja $L \subset W$ um subconjunto dos vértices do grafo, subconjunto que pode ser subdividido em ao menos M subconjuntos menores $\ell \subset L$. Em cada subconjunto ℓ os vértices formam uma árvore que alcança s_0 e cada POI $p \in P$. Uma dada árvore $\ell \in L$ pode ter várias arestas conectadas a s_0, mas apenas uma aresta entre a árvore e um dado POI $p \in P$. Cada POI $p \in P$ está conectado a todas as árvores de L. Seja x_i^ℓ um vetor binário de $|I| * M$ posições. Seja $x_i^\ell = 1$ se o sensor $i \in I$ pertence a árvore $\ell \in L$, e seja $x_i^\ell = 0$ em caso contrário. Seja $y_{ij}^{p\ell}$ um vetor binário de $|A| * |P| * M$ posições. Seja $y_{ij}^{p\ell} = 1$ se a aresta $(i, j) \in A | i, j \in V$ pertence à árvore ℓ e a árvore ℓ está conectada ao POI $p \in P$, e seja $y_{ij}^{p\ell} = 0$ em caso contrário. A restrição (3.2) garante que cada sensor só pode estar conectado a uma única árvore $\ell \in L$. A restrição (3.5) garante a cobertura mínima em cada POI $p \in P$. A restrição (3.4) relaciona os vetores **x** e **y**. A restrição (3.3) restringe as arestas e sensores que podem fazer parte de um mesmo caminho. A função objetivo (3.1) minimiza o número de sensores utilizados ao minimizar o somatório do vetor x. Denotamos por $I(p) \subset I$ o subconjunto dos sensores capazes de cobrir o POI p.

$$\min \sum_{i \in I} \sum_{\ell \in L} x_i^\ell \tag{3.1}$$

$s.a.$

$$\sum_{\ell \in L} x_i^\ell \leq 1, \quad \forall i \in I \tag{3.2}$$

$$\sum_{(i,j) \in A} y_{ij}^{p\ell} - \sum_{(j,i) \in A} y_{ji}^{p\ell} = \begin{cases} 1, \text{ se } i = p \\ \text{-1, se } i = s \\ 0, \text{ em caso contrário} \end{cases} \quad \forall p \in P, \ell \in L, i \in I \tag{3.3}$$

$$\sum_{(i,j) \in A} y_{ij}^{p\ell} \leq x_i^\ell, \quad \forall p \in P, \ell \in L, i \in I \tag{3.4}$$

$$\sum_{\ell \in L} \sum_{i \in I(p)} x_i^\ell \geq k, \quad \forall p \in P \tag{3.5}$$

3.2 ILP Fluxo-único

O ILP fluxo-único tem na entrada uma tupla $\langle W, K, M \rangle$. O ILP computa M árvores no grafo $W(P \bigcup I \bigcup s_0, A)$, cada árvore formada por todos os sensores de caminhos iniciados em cada POI. Diferentemente do ILP multi-fluxo, o ILP fluxo-único não identifica explicitamente a

qual das M árvores cada sensor fazem parte, apenas determinando se um sensor faz parte de ao menos uma das árvores. Seja x_i um vetor binário de $|I|$ posições que representa quais sensores fazem parte da solução computada. Seja $x_i = 1$ se o sensor $i \in I$ participa da solução, e seja $x_i = 0$ em caso contrário. Seja y_{ij}^p um vetor binário de $|A| * |P|$ posições. A restrição (3.8) relaciona os vetores **x** e **y** garantindo que uma posição em y só pode ter valor 1 se os sensores da aresta (i, j) também participarem da solução. A restrição (3.9) garante a cobertura mínima em cada POI $p \in P$. A restrição (3.7) restringe as arestas e sensores que podem fazer parte de um mesmo caminho. A função objetivo (3.6) minimiza o somatório dos valores do vetor x. Denotamos por $I(p) \subset I$ o subconjunto dos sensores capazes de cobrir o POI p.

$$\min \sum_{i \in I} x_i \tag{3.6}$$

$s.a.$

$$\sum_{(i,j) \in A} y_{ij}^p - \sum_{(j,i) \in A} y_{ji}^p = \begin{cases} \text{m, se } i = p \\ \text{-m, se } i = s_0 \\ \text{0, em caso contrário} \end{cases} , \forall p \in P, i \in I \tag{3.7}$$

$$\sum_{(i,j) \in A} y_{ij}^p \leq x_i, \quad \forall p \in P, i \in I \tag{3.8}$$

$$\sum_{i \in I(p)} x_i \geq k, \quad \forall p \in P \tag{3.9}$$

Capítulo 4

Heurísticas

Neste capítulo, são descritas quatro heurísticas determinísticas para o problema KCMC elaboradas neste trabalho. Heurísticas determinísticas tem comportamento previsível, não dependendo de valores aleatórios e, portanto, tendo os mesmos resultados em execuções subsequentes. Todas as heurísticas apresentadas neste trabalho empregam a meta-heurística *fix-and-optimize*. O objetivo das heurísticas empregadas na primeira etapa é obter um subconjunto dos sensores da instância do problema KCMC que são suficientes para as propriedades de K-Cobertura e M-Conectividade. Quanto menos sensores nos subconjuntos obtidos menor o esforço computacional necessário para computar o algoritmo exato na segunda etapa, que é o *branch-and-bound* [32]. Contudo é possível que um subconjunto computado por uma heurística na primeira etapa não contenha sensores necessários para que o algoritmo *branch-and-bound* compute um resultado ótimo. As heurísticas apresentadas neste trabalho balanceiam de formas diferentes a economia de esforço computacional e a probabilidade de computação de resultados ótimos.

Todas as heurísticas apresentadas neste trabalho são determinísticas e utilizam três algoritmos internamente: o algoritmo de Dinitz, o procedimento AddCov e um algoritmo *branch-and-bound*. Variantes do algoritmo de Dinitz são utilizadas para computar subconjuntos de sensores suficientes para a propriedade de M-Conectividade, e o procedimento AddCov acrescenta sensores nos subconjuntos até que sejam também suficientes para a propriedade de K-Cobertura. O algoritmo *branch-and-bound* é utilizado para minimizar a quantidade de sensores nos subconjuntos computados pelas variantes do algoritmo de Dinitz e o procedimento AddCov.

4.1 Algoritmo de Dinitz

O algoritmo de Dinitz [10, 11] computa o máximo fluxo entre dois vértices de um grafo. Outros trabalhos também se referem ao algoritmo de Dinitz como *Dinic*. O máximo fluxo entre dois vértices consiste na maior quantidade de caminhos disjuntos entre eles. Dois caminhos são ditos disjuntos quando não compartilham vértices entre si. O algoritmo de Dinitz computa em

cada iteração um caminho entre os vértices, desconsiderando, na iteração seguinte, os vértices que pertencem aos caminhos computados anteriormente. No pior caso, o algoritmo de Dinitz pode ser computado em tempo $O(|I|^2 * |A|)$, sendo I o conjunto de vértices e A o de arestas num grafo $G(I, A)$.

O algoritmo de Dinitz pode ser aplicado em instâncias do problema KCMC para determinar a conectividade de POIs. Inicialmente, computa-se um grafo $B(P \bigcup I \bigcup s_0, A')$ que tem os mesmos vértices do grafo $W(P \bigcup I \bigcup s_0, A)$ e arestas que ligam os mesmos vértices, contudo na direção contrária. O algoritmo então realiza uma busca-em-largura a partir de s_0, anotando quantas iterações foram necessárias para chegar a cada sensor. A quantidade de iterações necessárias para chegar a um dado sensor é dita sua *distância* de s_0. Na segunda etapa, o algoritmo realiza uma busca-em-profundidade no grafo original W a partir de cada POI $p \in P$, priorizando sensores com menor distância para s_0. Em cada iteração do algoritmo de busca-em-profundidade, um caminho é computado entre p e s_0. O algoritmo continua iterando até não haverem mais caminhos possíveis, e contabiliza como máximo fluxo originado no POI p a maior quantidade de caminhos disjuntos que existem entre p e s_0.

4.2 Procedimento AddCov

O procedimento AddCov, descrito no pseudocódigo 1, acrescenta sensores em um conjunto Δ até que seus membros sejam suficientes para a propriedade de K-Cobertura. O procedimento AddCov tem na entrada o grafo W, valor de K e um conjunto Δ de sensores. Inicialmente verifica-se a cobertura em cada POI utilizando-se apenas os sensores do conjunto Δ. Se a cobertura for suficiente, o conjunto Δ é retornado. Entre as linhas 4 e 7 procedimento AddKCov itera cada POI $p \in P$ que não seja coberto por ao menos K sensores do conjunto Δ. Denotamos por $I(p) \subset I$ o subconjunto dos sensores capazes de cobrir o POI p. Para cada POI p, o procedimento concede um *voto* para cada um dos sensores do conjunto $I(p) \setminus \Delta$ que poderiam cobrir p e não estão no conjunto Δ. Entre as linhas 9 e 12, o procedimento itera cada sensor em ordem decrescente do número de votos que recebeu. Em cada iteração, um sensor é acrescentado ao conjunto Δ, e se verifica se seus membros são suficientes para a propriedade de K-Cobertura. Se sim, o procedimento é interrompido e o conjunto Δ é retornado.

Procedimento AddCov(W, K, Δ)

```
1: if VerificaKCobertura(W, K, Δ) then
2:     return Δ
3: votos ← Vetor[I]                                  ▷ Vetor de |I| posições de valor 0
4: for p ∈ P do
5:     if |I(p) ∩ Δ| < K then                        ▷ I(p) são sensores de C que podem cobrir p
6:         for i ∈ I(p) \ Δ do
7:             votos[i] ← votos[i] + 1
8: sensores_ordenados ← OrdenacaoDecrescente(I \ Δ, votos)
9: for i ∈ sensores_ordenados do
10:    Δ ← Δ ⋃ {i}
11:    if VerificaKCobertura(W, K, Δ) then
12:        return Δ
```

Pseudocódigo 1: Procedimento AddCov

Procedimento DeltaBranchAndBound(W, K, M)

```
1: A'_K ← {(i, j) ∀(i, j) ∈ A_K : i, j ∈ Δ}
2: A'_M ← {(i, j) ∀(i, j) ∈ A_M : i, j ∈ Δ}
3: A'_S ← {(i, j) ∀(i, j) ∈ A_S : i, j ∈ Δ}
4: W' ← (Δ ⋃ {s_0}, A'_K ⋃ A'_M ⋃ A_S)
5: return BranchAndBound(W', K, M)
```

Pseudocódigo 2: Procedimento DeltaBranchAndBound

4.3 Procedimento DeltaBranchAndBound

O procedimento $DeltaBranchAndBound(W, K, M, \Delta)$ descrito no pseudocódigo 2 recebe na entrada uma instância $\langle W, K, M\rangle$ do problema KCMC e um conjunto Δ. O conjunto Δ é utilizado para computar a instância $\langle W', K, M\rangle$ que contém apenas os sensores de $\Delta \in I$. Na linha 1, o conjunto de arestas $A'_K \subset A_K$ definido como um subconjunto das arestas de A_K cujos vértices nas extremidades estejam em Δ. O conjunto $A_K \subset A$ é o conjunto de arestas do grafo W que têm origem em POIs e destino nos sensores que cobrem os POIs. Na linha 2, o conjunto de arestas $A'_M \in A_M$ definido como um subconjunto das arestas de A_M cujos vértices nas extremidades estejam em Δ. O conjunto $A_M \subset A$ é o conjunto de arestas do grafo W cujos vértices nas extremidades são sensores que podem se comunicar diretamente. Na linha 3, o conjunto de arestas $A'_S \in A_S$ definido como um subconjunto das arestas de A_S cujos vértices nas origens estejam em Δ. O conjunto $A_S \in A$ é é o conjunto de arestas do grafo W cujos vértices são sensores que podem se comunicar diretamente com s_0. Na linha 4, o grafo W' é definido pelos vértices em $\Delta \bigcup \{s_0\}$ e pelas arestas em $A'_K \bigcup A'_M \bigcup A'_S$. Na linha 5, o procedimento $DeltaBranchAndBound$ executa o algoritmo *branch-and-bound* recebendo a tupla $\langle W', K, M\rangle$ como parâmetros. O ILP utilizado é o fluxo-único.

Heurística DKOV(W, K, M)

1: $\Delta \leftarrow \emptyset$
2: **for** $p \in P$ **do**
3: **for** $\ell \in \{1, 2, ..., M\}$ **do**
4: $um\_caminho \leftarrow OnePathDinitz(W, p, s_0, \Delta)$
5: $\Delta \leftarrow \Delta \bigcup \{um\_caminho\}$
6: $\Delta \leftarrow AddCov(W, K, \Delta)$
7: **return** $DeltaBranchAndBound(W, K, M, \Delta)$

Pseudocódigo 3: Heurística DKOV para o problema KCMC

4.4 Heurística DKOV

O algoritmo de Dinitz pode ser usado para computar caminhos entre um dado POI $p \in P$ e s_0 iterativamente. Cada caminho computado não contém vértices pertencentes a caminhos computados em iterações anteriores. O algoritmo de Dinitz pode ser interrompido após computar M caminhos disjuntos entre o POI p e s_0. Os vértices dos M caminhos computados são suficientes para garantir a propriedade de M-Conectividade do POI p. Computando o algoritmo para cada POI $p \in P$, os vértices que compõem os M caminhos disjuntos computados serão suficientes para garantir a propriedade de M-conectividade em todos os POIs. Se cada POI deve ser coberto por ao menos K sensores e $K \leq M$, então todos os POIs têm cobertura suficiente. Contudo, se $K > M$, é possível que existam POIs que não sejam suficientemente cobertos pelos sensores dos caminhos computados pelo algoritmo de Dinitz.

A heurística DKOV para o problema KCMC computa o algoritmo de Dinitz iterativamente para cada POI $p \in P$, de forma a obter um conjunto de sensores suficientes para M-Conectividade. Sensores são, então, acrescentados no conjunto até que seus membros sejam suficientes para K-Cobertura. Inicialmente, a heurística define o conjunto $\Delta \leftarrow \emptyset$ como um conjunto vazio. Em cada iteração, o algoritmo de Dinitz é utilizado para identificar M caminhos entre o POI p e s_0. Os sensores que compõem cada caminho computado são acrescentados no conjunto Δ. Na linha 6 a heurística DKOV computa o procedimento AddCov, descrito no pseudocódigo 1, para acrescentar sensores no conjunto Δ até que sejam suficientes para a propriedade de K-Cobertura. O pseudocódigo 3 descreve o funcionamento da heurística DKOV, sendo que o procedimento $OnePathDinitz(W, p, s_0, \Delta)$ aplica o algoritmo de Dinitz para computar os sensores de um único caminho entre p e s_0 que não contenha sensores do conjunto Δ. O procedimento $DeltaBranchAndBound(W, K, M, \Delta)$ computa o algoritmo *branch-and-bound* na instância $\langle W, K, M \rangle$ do problema KCMC na entrada, considerando apenas os sensores do conjunto $\Delta \in I$.

Variações da heurística DKCOV podem ter menos sensores no conjunto Δ. Podem, por exemplo, ter caminhos iniciados em diferentes POIs, mas que contenham os mesmos sensores.

Heurística REUSE(W, K, M)

```
1:  votos ← Vetor[I]                                   ▷ Vetor de |I| posições de valor 0
2:  for p ∈ P do
3:      d ← ∅
4:      for ℓ ∈ {1, 2, ..., M} do
5:          um_caminho ← OnePathDinitz(W, p, s_0, d)
6:          d ← d ⋃ um_caminho
7:          for i ∈ um_caminho do
8:              votos[i] ← votos[i] + 1
9:  Δ ← ∅
10: for p ∈ P do
11:     δ ← ∅
12:     for ℓ ∈ {1, 2, ..., M} do
13:         um_caminho ← BuscaProfundidade(W, p, s_0, votos, δ)
14:         δ ← δ ⋃ um_caminho
15:     Δ ← Δ ⋃ δ
16: Δ ← AddCov(W, K, Δ)
17: return DeltaBranchAndBound(W, K, M, Δ)
```

Pseudocódigo 4: Heurística REUSE para o problema KCMC

A propriedade de M-Conectividade requer M caminhos disjuntos entre um mesmo POI e a *sink*, mas dois caminhos iniciados em POIs diferentes podem conter os mesmos sensores.

4.5 Heurística REUSE

A heurística REUSE é uma variante da heurística DKOV. Ela tem como objetivo computar um conjunto Δ de menor cardinalidade que o computado por DKOV. Para isso, REUSE utiliza caminhos iniciados em POIs diferentes, mas que contêm os mesmos sensores. Assim, a quantidade de sensores no conjunto Δ pode ser menor, o que implica um menor esforço computacional no algoritmo *branch-and-bound*.

O pseudocódigo 4 descreve o funcionamento da heurística REUSE. Para todo POI $p \in P$, a heurística utiliza o algoritmo de Dinitz para computar caminhos entre p e s_0. Sempre que um caminho é computado na linha 5, cada um de seus sensores recebe um voto. O algoritmo de Dinitz é interrompido após identificar M caminhos entre p e s_0. Na linha 9, a heurística REUSE define o conjunto vazio $\Delta \leftarrow \emptyset$. Para todo POI $p \in P$, a heurística inicializa o conjunto vazio δ e realiza uma busca em profundidade entre p e s_0. Em cada iteração, a busca em profundidade visita primeiro o sensor com maior quantidade de votos, evitando os sensores do conjunto δ. Quando a busca em profundidade chega em s_0, os sensores que compõem o caminho compu-

tado pela busca são acrescentados no conjunto δ. Na linha 15, após computar M caminhos entre um POI $p \in P$ e s_0 executando a busca em profundidade M vezes, os membros do conjunto δ são acrescentados no conjunto Δ. Após iterar todos os POIs, a heurística REUSE utiliza o procedimento AddCov para acrescentar sensores no conjunto Δ até que os membros do mesmo, que já são suficientes para a propriedade de M-Conectividade, sejam suficientes para K-Cobertura. O conjunto Δ é utilizado para computar o algoritmo *branch-and-bound* com o procedimento $DeltaBranchAndBound(W, K, M, \Delta)$.

A heurística REUSE computa um conjunto Δ que pode ser menor que o computado pela heurística DKOV ao priorizar o uso de sensores que participam de mais caminhos. Sensores que participam de mais caminhos proveem conectividade para mais que um POI, contudo, cada sensor é apenas um único elemento no conjunto Δ. A heurística REUSE contabiliza votos em sensores ao computar exatamente M caminhos entre cada POI $p \in P$ e s_0. É possível que alguns sensores participem de mais caminhos do que os computados. É possível que, ao se computar mais caminhos, o conjunto Δ resultante tenha menos membros.

4.6 Heuristica BREADTH

A heurística BREADTH difere da heurística REUSE ao computar mais de M caminhos iniciados em cada POI para computar votos. Após computar o M-ésimo caminho iniciado no POI $p \in P$, a heurística BREADTH continua computando caminhos até identificar algum que contenha mais sensores que o maior caminho computado até então. A heurística evita caminhos demasiadamente grandes ao interromper a execução assim que um caminho maior do que o suficiente é computado.

O pseudocódigo 5 descreve o funcionamento da heurística BREADTH. Para todo POI $p \in P$, a heurística utiliza o algoritmo de Dinitz para computar caminhos entre p e s_0. Sempre que um caminho é computado, cada um de seus sensores recebe um voto. A heurística registra o valor da maior quantidade de sensores em um caminho dentre os primeiros M computados. Após computar M caminhos, a heurística continua computando caminhos até obter algum que contenha mais sensores que o maior valor registrado. Na linha 15, a heurística BREADTH define o conjunto vazio $\Delta \leftarrow \emptyset$. Para cada POI $p \in P$, a heurística inicializa o conjunto vazio δ e realiza uma busca em profundidade entre p e s_0. Em cada iteração, a busca em profundidade visita primeiro o sensor com maior quantidade de votos, evitando os sensores do conjunto δ. Quando a busca em profundidade chega em s_0, os sensores que compõem o caminho computado pela busca são acrescentados no conjunto δ. Na linha 21, após computar M caminhos entre um POI $p \in P$ e s_0, os membros do conjunto δ são acrescentados no conjunto Δ. O conjunto Δ é utilizado para computar o algoritmo *branch-and-bound* com o procedimento

Heurística BREADTH(W, K, M)

```
votos ← Vetor[I]                                   ▷ Vetor de |I| posições de valor 0
for p ∈ P do
    d ← ∅
    um_caminho ← ∅
    quantidade ← 0
    maior_caminho ← 0
    while quantidade < M OR maior_caminho ≤ |um_caminho| do
        um_caminho ← OnePathDinitz(W, p, s_0, d)
        d ← d ⋃ um_caminho
        for i ∈ um_caminho do
            votos[i] ← votos[i] + 1
        quantidade ← quantidade + 1
        if quantidade < M AND maior_caminho < |um_caminho| then
            maior_caminho ← |um_caminho|
Δ ← ∅
for p ∈ P do
    δ ← ∅
    for ℓ ∈ {1, 2, ..., M} do
        um_caminho ← BuscaProfundidade(W, p, s_0, votos, δ)
        δ ← δ ⋃ um_caminho
    Δ ← Δ ⋃ δ
Δ ← AddCov(W, K, Δ)
return DeltaBranchAndBound(W, K, M, Δ)
```

Pseudocódigo 5: Heurística BREADTH para o problema KCMC

$DeltaBranchAndBound(W, K, M, \Delta)$.

A heurística BREADTH pode acrescentar mais sensores no conjunto Δ do que a heurística REUSE. Isso porque a heurística BREADTH computa mais caminhos do que REUSE, de forma que sensores podem receber votos por fazerem parte de caminhos que não são necessários para a propriedade de M-Conectividade. Por ter mais sensores no conjunto Δ, a heurística BREADTH pode obter resultados melhores que REUSE após computar procedimento $DeltaBranchAndBound(W, K, M, \Delta)$. Contudo, o esforço computacional necessário para o algoritmo *branch-and-bound* cresce exponencialmente com a quantidade de sensores no conjunto $\Delta \in I$, de forma que a inclusão de mais sensores pode inviabilizar seu uso na prática.

4.7 Heurística FEWER

A heurística FEWER computa um conjunto $\Delta \in I$ de igual ou menor tamanho que os conjuntos computados pelas heurísticas DKOV, REUSE e BREADTH. As três heurísticas são

Heurística FEWER(W, s, K, M**)**

1: $\Delta_{DKOV} \leftarrow$ **DKOV**(W, s, K, M)
2: $\Delta_{REUSE} \leftarrow$ **REUSE**(W, s, K, M)
3: $\Delta_{BREADTH} \leftarrow$ **BREADTH**(W, s, K, M)
4: $\Delta_{FEWER} \leftarrow MIN(\Delta_{DKOV}, \Delta_{REUSE}, \Delta_{BREADTH})$
5: $\Delta \leftarrow AddCov(W, K, \Delta_{FEWER})$
6: **return** $DeltaBranchAndBound(W, K, M, \Delta)$

Pseudocódigo 6: Heurística FEWER para o problema KCMC

computadas até concluírem a computação do conjunto Δ, antes de chamarem o procedimento $BranchAndBound$. Os conjuntos Δ_{DKOV}, Δ_{REUSE} e $\Delta_{BREADTH}$ são comparados, e o conjunto com menos membros é chamado de Δ_{FEWER}. A heurística então computa o procedimento AddCov 1 para acrescentar sensores no conjunto Δ_{FEWER} até que seus membros sejam suficientes para a propriedade de K-Cobertura. O conjunto Δ_{FEWER} é utilizado para computar o algoritmo *branch-and-bound* com o procedimento $DeltaBranchAndBound(W, K, M, \Delta_{FEWER})$. O pseudocódigo 6 descreve o funcionamento da heurística FEWER.

A heurística FEWER prioriza a economia de esforço computacional no algoritmo *branch-and-bound*. Para isso, a heurística minimiza a quantidade de sensores no conjunto Δ_{FEWER} utilizando técnicas similares as das heurísticas DKOV, REUSE e BREADTH. Por essa razão a heurística FEWER é tanto eficaz quanto, ou mais eficaz na redução do esforço computacional necessário para o algoritmo *branch-and-bound* que as demais heurísticas apresentadas nesse trabalho. Contudo, seus resultados podem não ser tão bons quanto o de outras heurísticas, porque sensores necessários para melhores resultados podem não estar no conjunto Δ_{FEWER}.

Capítulo 5

Experimentos e Resultados

As instâncias do problema KCMC utilizadas nos experimentos foram geradas de forma *ad-hoc* seguindo a metodologia do trabalho de Gupta [18]. Um conjunto P de POIs, um conjunto I de sensores e um conjunto Θ contendo uma única *sink* são posicionados numa área de 300×300. Os POIs e sensores tem coordenadas que seguem uma distribuição uniforme inteira $U(0, 300)$. A única *sink* de Θ, que portanto é s_0, é posicionada no centro da área nas coordenadas $(150; 150)$. O grafo $W(P \bigcup I \bigcup s_0, A)$ é gerado calculando-se as distâncias entre os POIs, sensores e s_0. Cada aresta (p, i) no conjunto $A_K \subset A$ tem origem em um POI $p \in P$ e algum sensor $i \in I$ que está a 50 ou menos unidades de distância euclidiana de p. Cada aresta (i, j) no conjunto $A_M \subset A$ tem origem e destino em sensores $i, j \in I$ que estejam a 100 ou menos unidades de distância euclidiana um do outro. Neste trabalho, a comunicação entre sensores é simétrica, de forma que, para cada aresta $(i, j) \in A_M$ temos também a aresta inversa (j, i). Cada aresta (i, s_0) no conjunto $A_S \subset A$ tem origem em um sensor $i \in I$ que estejam a 100 ou menos unidades de distância euclidiana de s_0. Os valores 50 e 100 correspondem, respectivamente, aos valores do *raio de cobertura* e do *raio de comunicação* utilizados no trabalho de Gupta [18]. Dada a natureza aleatória do posicionamento dos POIs e sensores, em algumas situações, pode não ser possível resolver o problema KCMC para dados valores de K e M. Nesses casos, novos posicionamentos aleatórios de POIs e sensores são gerados até que seja possível resolver o problema KCMC para dados valores de K e M. Assim, são construídas instâncias do problema KCMC como tuplas $\langle W, K, M \rangle$, que podem ser resolvidas pelo algoritmo *branch-and-bound*.

As instâncias geradas neste trabalho podem ser divididas em 36 classes com 10 instâncias em cada classe. Dentre as 360 instâncias, não existem pares que tenham grafos de comunicação isomórficos. Todas as instâncias de uma mesma classe têm uma mesma quantidade de POIs e sensores e mesmos valores de K e M. A quantidade de POIs em cada classe varia entre 100 e 200, a quantidade de sensores entre 100 e 500 e os valores de K e M entre 1 e 3 de forma que $K \geq M$.

Os experimentos realizados utilizam artefatos de software desenvolvidos neste trabalho. Os ILPs descritos no capítulo 3 foram implementados com uso do pacote de software comercial Gurobi [19] versão 9.5.1 e a linguagem Python 3.10. Os procedimentos e heurísticas descritos no capítulo 4 foram implementados na linguagem de programação C++ e compilados utilizando-se as ferramentas GNU GCC/G++ 12.2. O processamento de uma dada instância é dado como con-

cluído em três situações: no término do processamento do algoritmo; quando o tempo decorrido chega a uma hora; ou quando não há memória disponível suficiente para concluir o processamento do algoritmo. O processamento de cada instância foi realizado independentemente em ambientes isolados, cada um com 8Gb de memória e processador Intel Xeon de 3.1GHz. O processamento de cada instância utiliza um único núcleo de processamento. Os experimentos não foram repetidos porque as heurísticas analisadas são determinísticas e têm resultados similares em execuções subsequentes.

Os ILPs e heurísticas são comparados por meio do método descrito por Carvalho no trabalho [7] para comparação estatística de algoritmos. Dados um conjunto de algoritmos e instâncias, todos os algoritmos são executados para cada instância e seus resultados e tempo de execução são anotados. O método de Carvalho pode ser aplicado quando alguns dos algoritmos comparados não retornam resultados para todas as instâncias. Para cada instância, comparamos cada par de algoritmos. Em cada comparação, o algoritmo considerado superior é o que tem o melhor resultado. Se um dos algoritmos comparados não obteve resultados, consideramos que tem qualidade igual a todos os outros algoritmos que não obtiveram resultados para a mesma instância, e qualidade inferior de todos os algoritmos que obtiveram algum resultado. Se os dois algoritmos comparados tiveram o mesmo resultado para a mesma instância, o algoritmo considerado superior é o que foi processado em menos tempo. Se os dois algoritmos comparados foram processados em quantidades iguais de tempo, ambos tem a mesma qualidade. Comparamos os algoritmos para obter um *ranking* dos melhores para cada instância. Algoritmos de qualidade similar tem *ranking* equivalente média das posições que ocupam. A superioridade de algum algoritmo sobre os demais pode ser detectada por testes estatísticos baseados em *ranking*, como o teste de Friedman, e um teste *post-hoc* subsequente, como o de Nemenyi.

5.1 Comparação dos ILPs

O experimento preliminar compara os dois ILPs apresentados no capítulo 3. O algoritmo *branch-and-bound* implementado no pacote de software comercial Gurobi [19] foi configurado em dois artefatos de software, cada um implementando um dos ILPs. Cada artefato de software foi utilizado para processar cada uma das 360 instâncias geradas neste trabalho, em ambiente de execução exatamente igual aos dos demais experimentos. A tabela 1 exibe os resultados do algoritmo *branch-and-bound* configurado com o ILP multi-fluxo, e a tabela 2 os do algoritmo configurado com o ILP fluxo-único. No total, foram computados resultados ótimos para 176 das 360 instâncias.

As primeiras quatro colunas das tabelas 1 e 2 indicam as quantidades de POIs, sensores e os valores de K e M de todas as instâncias de uma mesma classe. A quinta coluna de cada tabela

$\|P\|$	$\|I\|$	K	M	#otm	#LM	#LT	#int	#linhas	#colunas	#naozeros	#nodos	LI	LS	%*gap*	tempo(s)
100	100	1	1	10	0	0	2.400	2.0E+04	3.3E+05	8.5E+05	1.000	14.900	14.900	0.000	15.535
100	100	2	1	10	0	0	2.000	2.0E+04	3.3E+05	8.4E+05	1.000	29.200	29.200	0.000	8.662
100	100	2	2	10	0	0	3.700	4.1E+04	6.6E+05	1.7E+06	1.400	29.200	29.200	0.000	157.599
100	100	3	1	10	0	0	1.600	2.0E+04	3.2E+05	8.3E+05	1.000	43.800	43.800	0.000	4.802
100	100	3	2	10	0	0	2.700	4.1E+04	6.6E+05	1.7E+06	1.000	43.300	43.300	0.000	59.605
100	100	3	3	9	0	1	4.700	6.1E+04	9.8E+05	2.5E+06	17.700	46.300	46.300	0.002	1561.536
100	300	1	1	9	0	1	2.600	6.1E+04	2.5E+06	7.2E+06	1.000	12.400	12.400	0.000	1063.526
100	300	2	1	10	0	0	2.000	6.1E+04	2.6E+06	7.2E+06	1.600	24.900	24.900	0.000	664.369
100	300	2	2	0	3	7	0.286	1.2E+05	5.1E+06	1.4E+07	1.000	24.521	24.714	0.810	-
100	300	3	1	10	0	0	2.300	6.1E+04	2.5E+06	7.2E+06	1.000	37.500	37.500	0.000	406.428
100	300	3	2	2	3	5	0.857	1.2E+05	5.0E+06	1.4E+07	1.000	38.000	38.000	0.357	599.062
100	300	3	3	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	1	1	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	2	1	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	2	2	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	3	1	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	3	2	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	3	3	0	10	0	0.000	-	-	-	-	-	-	-	-
200	100	1	1	10	0	0	2.800	4.1E+04	8.2E+05	1.9E+06	1.000	16.600	16.600	0.000	53.922
200	100	2	1	10	0	0	2.000	4.1E+04	7.8E+05	1.8E+06	1.000	33.600	33.600	0.000	20.390
200	100	2	2	8	0	2	4.400	8.1E+04	1.6E+06	3.7E+06	15.600	33.804	33.900	0.006	1106.941
200	100	3	1	10	0	0	1.300	4.1E+04	7.9E+05	1.8E+06	1.000	49.700	49.700	0.000	14.018
200	100	3	2	10	0	0	3.600	8.1E+04	1.6E+06	3.6E+06	1.000	50.300	50.300	0.000	152.008
200	100	3	3	3	0	7	5.600	1.2E+05	2.3E+06	5.3E+06	21.300	51.420	51.600	0.025	2612.202
200	300	1	1	0	9	1	0.000	1.2E+05	5.5E+06	1.5E+07	1.000	0.000	0.000	-	-
200	300	2	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	300	2	2	0	10	0	0.000	-	-	-	-	-	-	-	-
200	300	3	1	0	9	1	0.000	1.2E+05	5.5E+06	1.5E+07	1.000	42.000	42.000	-	-
200	300	3	2	0	10	0	0.000	-	-	-	-	-	-	-	-
200	300	3	3	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	1	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	2	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	2	2	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	3	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	3	2	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	3	3	0	10	0	0.000	-	-	-	-	-	-	-	-

Tabela 1: Resultados do ILP multi-fluxo

mostra em quantas das dez instâncias de cada classe foi possível obter soluções ótimas. A sexta coluna de cada tabela mostra em quantas das dez instâncias não foi possível obter soluções porque o programa requisitou mais memória que o total disponível no ambiente de execução dos experimentos. A sétima coluna de cada tabela mostra em quantas das dez instâncias não foi possível obter soluções ótimas antes do fim do tempo dedicado para cada instância. A oitava coluna de cada tabela informa a média da quantidade de soluções inteiras viáveis computadas para cada instância de cada classe. A nona e décima coluna de cada tabela informam a média da quantidade de linhas e colunas no programa formado pelo ILP, no qual a décima primeira coluna de cada tabela informa a quantidade de variáveis não-zero. A décima segunda coluna de cada tabela informa a quantidade média de nós computados no processamento do ILP. A décima terceira e décima quarta colunas de cada tabela representam, respectivamente, a média dos valores de limite inferior e limite superior para os resultados ótimos, e a décima quinta coluna a média dos percentuais representados pela diferença entre as soluções obtidas e o limite inferior. A décima sexta coluna de cada tabela informa o tempo médio, em segundos, para processar cada instância de cada classe até a obtenção de uma solução ótima. Apenas instâncias

$\|P\|$	$\|I\|$	K	M	#otm	#LM	#LT	#int	#linhas	#colunas	#naozeros	#nodos	LI	LS	%*gap*	tempo(s)
100	100	1	1	10	0	0	3.600	2.0E+04	3.3E+05	8.5E+05	1.000	14.900	14.900	0.000	17.456
100	100	2	1	10	0	0	1.600	2.0E+04	3.3E+05	8.4E+05	1.000	29.200	29.200	0.000	8.292
100	100	2	2	10	0	0	1.600	2.0E+04	3.3E+05	8.5E+05	1.000	28.900	28.900	0.000	12.603
100	100	3	1	10	0	0	1.700	2.0E+04	3.2E+05	8.3E+05	1.000	43.800	43.800	0.000	5.077
100	100	3	2	10	0	0	1.500	2.0E+04	3.3E+05	8.4E+05	1.000	43.300	43.300	0.000	11.996
100	100	3	3	10	0	0	1.200	2.0E+04	3.3E+05	8.4E+05	1.000	45.600	45.600	0.000	14.370
100	300	1	1	9	0	1	2.400	6.0E+04	2.5E+06	7.2E+06	1.000	12.400	12.400	0.000	1104.429
100	300	2	1	10	0	0	2.400	6.0E+04	2.6E+06	7.2E+06	1.600	24.900	24.900	0.000	708.677
100	300	2	2	9	0	1	2.300	6.0E+04	2.5E+06	7.2E+06	1.000	24.800	24.800	0.000	809.473
100	300	3	1	10	0	0	2.200	6.0E+04	2.5E+06	7.2E+06	1.000	37.500	37.500	0.000	388.021
100	300	3	2	10	0	0	1.900	6.0E+04	2.5E+06	7.1E+06	1.000	37.900	37.900	0.000	621.008
100	300	3	3	7	0	3	1.700	6.0E+04	2.5E+06	7.1E+06	1.000	37.600	37.600	0.000	783.940
100	500	1	1	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	2	1	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	2	2	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	3	1	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	3	2	0	10	0	0.000	-	-	-	-	-	-	-	-
100	500	3	3	0	10	0	0.000	-	-	-	-	-	-	-	-
200	100	1	1	10	0	0	2.800	4.1E+04	8.2E+05	1.9E+06	1.000	16.600	16.600	0.000	45.479
200	100	2	1	10	0	0	1.500	4.1E+04	7.8E+05	1.8E+06	1.000	33.600	33.600	0.000	19.913
200	100	2	2	10	0	0	2.700	4.1E+04	8.1E+05	1.8E+06	1.000	33.400	33.400	0.000	57.869
200	100	3	1	10	0	0	1.600	4.1E+04	7.9E+05	1.8E+06	1.000	49.700	49.700	0.000	14.096
200	100	3	2	10	0	0	1.400	4.1E+04	8.0E+05	1.8E+06	1.000	50.300	50.300	0.000	21.521
200	100	3	3	10	0	0	1.500	4.1E+04	7.7E+05	1.8E+06	1.000	50.900	50.900	0.000	30.044
200	300	1	1	0	8	2	0.000	1.2E+05	5.5E+06	1.5E+07	1.000	6.667	7.000	-	-
200	300	2	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	300	2	2	0	9	1	0.000	1.2E+05	5.4E+06	1.4E+07	1.000	27.818	28.000	-	-
200	300	3	1	1	9	0	1.000	1.2E+05	5.5E+06	1.5E+07	1.000	42.000	42.000	0.000	2597.422
200	300	3	2	0	8	2	0.000	1.2E+05	5.6E+06	1.5E+07	1.000	42.500	42.500	-	-
200	300	3	3	0	9	1	0.000	1.2E+05	5.3E+06	1.4E+07	1.000	40.500	41.000	-	-
200	500	1	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	2	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	2	2	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	3	1	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	3	2	0	10	0	0.000	-	-	-	-	-	-	-	-
200	500	3	3	0	10	0	0.000	-	-	-	-	-	-	-	-

Tabela 2: Resultados do ILP fluxo-único

em que foi possível obter soluções ótimas são consideradas na décima quinta e décima sexta colunas. As tabelas mostram o símbolo $-$ em células que representam classes e colunas onde não foi possível obter resultados para nenhuma instância.

Na maioria dos experimentos, não foi necessário computar nós no algoritmo *branch-and-bound* além do nó inicial. Na décima segunda coluna das tabelas 1 e 2, frequentemente a quantidade média de nós explorados em cada classe é 1. A décima terceira e décima quarta colunas das tabelas evidenciam que, frequentemente, as médias dos valores dos limites inferiores e superiores da função objetivo numa mesma classe têm o mesmo valor. Assim, em muitos casos, os limites estimados pelo algoritmo *branch-and-bound* tiveram valores similares aos resultados ótimos.

Como pode ser visto na quinta coluna de cada uma das tabelas 1 e 2, não foi possível possível computar soluções ótimas para várias instâncias em classes com 300 ou 500 sensores. A sexta coluna de cada tabela tem seus maiores valores nas classes de instâncias com 500 sensores, o que indica que não foi possível representar instâncias desse tamanho com a quantidade de memória disponível no ambiente de execução do experimento. A sétima coluna de cada tabela

tem seus maiores valores em instâncias de 100 e 300 sensores, o que evidencia que é possível representar essas instâncias com a quantidade de memória disponível no ambiente de execução do experimento, mas o tempo de processamento não foi suficiente para computar soluções ótimas. Assim, para computar instâncias com 500 sensores ou mais, o ambiente de execução precisaria de mais memória e seria necessário mais tempo de processamento.

Os resultados do algoritmo configurado com os dois ILPs é similar nos casos em que foi possível obter soluções ótimas com ambos para a mesma instância. Contudo, foi possível solucionar mais instâncias com uso do ILP fluxo-único. Como pode ser visto nas nonas, décimas e décimas primeiras colunas das tabelas 1 e 2, o uso do ILP fluxo-único requer menos memória do que o uso do ILP multi-fluxo, uma vez que o algoritmo *branch-and-bound* computa matrizes com mais linhas e colunas de valores não-zero. Como pode ser visto na décima sexta coluna das tabelas 1 e 2, o tempo necessário para computar soluções ótimas com uso do ILP fluxo-único é menor do que com o uso do ILP multi-fluxo. Assim, é menos provável que a computação de uma instância seja interrompida prematuramente, por motivos de falta de memória ou tempo, com o uso do ILP fluxo-único do que com o ILP multi-fluxo.

5.2 Avaliação das Heurísticas

O principal experimento deste trabalho consiste em executar os artefatos de software implementados para cada heurística com cada instância gerada. Em todos os casos, o algoritmo *branch-and-bound* foi configurado para utilizar o ILP fluxo-único. Em todos os casos, foi possível obter soluções ao final da execução do algoritmo *branch-and-bound*, em nenhum caso o processamento foi interrompido devido a limites de tempo e memória. Os resultados do experimento estão na tabela 3.

As primeiras quatro colunas da tabela 3 indicam as quantidades de POIs, sensores e os valores de K e M de todas as instâncias de uma mesma classe. A quinta coluna mostra a média do percentual representado pelos sensores no conjunto Δ computados pela heurística DKOV nas instâncias da classe. Esse valor representa o percentual dos sensores que é processado pelo algoritmo *branch-and-bound*. A sexta, sétima e oitava colunas têm valores análogos aos da quinta coluna, contudo referentes, respectivamente, às heurísticas REUSE, BREADTH e FEWER. A nona coluna mostra a média de tempo necessário para processamento da heurística DKOV em cada instância de cada classe. A décima, décima primeira e décima segunda colunas têm valores análogos aos da nona coluna, contudo referentes, respectivamente, às heurísticas REUSE, BREADTH e FEWER. A décima terceira coluna mostra a média dos *gaps* entre a melhor solução computada pela heurística DKOV e as soluções ótimas para cada instância, quando foi possível computar as soluções ótimas nos experimentos preliminares. Os valores são expressos como

				Δ				tempo(s)				%*gap*			
$\|P\|$	$\|I\|$	K	M	DKOV	REUSE	BREADTH	FEWER	DKOV	REUSE	BREADTH	FEWER	DKOV	REUSE	BREADTH	FEWER
100	100	1	1	31.1	**25.3**	30.7	**25.3**	**0.01**	**0.01**	0.05	0.07	15.33	15.78	**10.47***	15.78
100	100	2	1	40.2	**37.8**	40.1	**37.8**	**0.01**	**0.01**	0.05	0.07	11.04	**10.05**	10.31	10.28
100	100	2	2	50.5	**43.2**	47.5	**43.2**	0.41	**0.02**	0.45	0.09	10.97	12.02	**7.34**	12.02
100	100	3	1	55.2	55.8	55.3	**53.9**	**0.01**	**0.01**	0.05	0.07	7.56	**5.98**	6.02	7.18
100	100	3	2	59.8	54.2	57.7	**54.0**	0.51	**0.02**	0.45	0.19	7.83	7.21	**6.47**	6.84
100	100	3	3	70.2	61.9	66.8	**61.7**	2.32	1.13	2.06	**1.11**	3.95	6.86	**2.73***	5.91
100	300	1	1	35.2	**24.5**	36.4	**24.5**	**0.02**	0.03	0.41	0.37	26.66	29.02	**20.21**	29.02
100	300	2	1	44.2	**37.8**	45.1	**37.8**	**0.02**	0.03	0.32	0.38	21.12	20.61	**16.92**	20.61
100	300	2	2	58.8	**44.9**	61.8	**44.9**	1.73	**0.05**	2.71	0.39	19.96	21.91	**14.09**	21.91
100	300	3	1	57.2	57.9	56.9	**54.7**	**0.12**	0.34	0.52	0.37	18.43	16.13	**14.65**	17.23
100	300	3	2	69.7	**56.7**	69.6	**56.7**	2.13	**0.16**	2.13	0.62	19.88	21.18	**16.06**	21.18
100	300	3	3	83.7	**65.1**	82.4	**65.1**	8.44	**1.48**	7.33	2.16	20.95	22.21	**15.77**	22.21
100	500	1	1	37.1	**26.5**	39.9	**26.5**	**0.03**	0.06	0.95	0.95	-	-	-	-
100	500	2	1	45.7	**37.9**	49.7	**37.9**	0.14	**0.06**	1.25	0.95	-	-	-	-
100	500	2	2	65.7	**45.6**	69.8	**45.6**	3.65	**0.10**	6.17	1.02	-	-	-	-
100	500	3	1	58.7	59.0	57.3	**55.9**	0.44	**0.26**	0.95	1.15	-	-	-	-
100	500	3	2	73.0	**56.8**	78.8	**56.8**	2.25	**0.30**	6.08	1.23	-	-	-	-
100	500	3	3	94.2	**67.3**	94.4	**67.3**	12.17	**1.74**	12.50	3.02	-	-	-	-
200	100	1	1	37.7	**29.2**	37.7	**29.2**	0.62	**0.03**	0.40	0.14	15.93	17.62	**13.16***	17.62
200	100	2	1	47.2	43.7	47.3	**43.6**	0.52	**0.13**	0.68	0.23	10.35	10.78	**8.96**	10.53
200	100	2	2	60.3	**50.2**	59.5	**50.2**	4.93	**1.75**	4.51	1.78	7.92	10.64	**6.1**	10.64
200	100	3	1	64.4	64.3	63.1	**61.8**	1.92	1.83	1.69	**1.64**	**4.22**	4.38*	5.09	4.53*
200	100	3	2	68.0	**62.9**	68.1	**62.9**	3.23	**2.25**	3.11	2.28	7.5	7.98	**6.64**	7.98
200	100	3	3	76.6	**68.8**	75.8	**68.8**	8.64	**4.67**	8.92	4.72	4.85	6.44	**4.33***	6.44
200	300	1	1	45.1	**30.0**	48.9	**30.0**	1.84	**0.07**	4.01	0.72	-	-	-	-
200	300	2	1	55.5	**45.1**	55.1	**45.1**	2.04	**0.57**	2.72	1.32	-	-	-	-
200	300	2	2	76.4	**55.9**	81.4	**55.9**	15.36	**2.92**	22.37	3.55	-	-	-	-
200	300	3	1	71.0	73.0	70.8	**68.6**	**2.84**	3.97	3.64	3.35	**17.65**	**17.65**	**17.65**	**17.65**
200	300	3	2	84.7	**65.1**	89.5	**65.1**	12.56	**2.92**	18.47	3.26	-	-	-	-
200	300	3	3	102.0	**76.3**	109.2	**76.3**	50.79	**8.27**	70.00	9.05	-	-	-	-
200	500	1	1	46.2	**30.7**	51.7	**30.7**	2.56	**0.11**	5.90	1.89	-	-	-	-
200	500	2	1	58.0	**43.9**	63.5	**43.9**	2.07	**0.41**	5.35	2.03	-	-	-	-
200	500	2	2	80.6	**54.0**	93.0	**54.0**	22.60	**2.39**	51.62	4.31	-	-	-	-
200	500	3	1	74.3	73.3	77.6	**70.8**	**3.47**	3.72	6.23	5.02	-	-	-	-
200	500	3	2	89.8	**66.7**	99.0	**66.7**	16.60	**3.09**	25.85	4.65	-	-	-	-
200	500	3	3	113.1	**79.6**	129.2	**79.6**	80.14	**11.38**	136.31	13.64	-	-	-	-

Tabela 3: Resultados das heurísticas

uma porcentagem da melhor solução obtida. Assim, um valor 3.95 na décima terceira coluna significa que a heurística DKOV obteve soluções com 3.95% sensores, em média, a mais que a solução ótima. A décima quarta, décima quinta e décima sexta colunas têm valores análogos aos da décima terceira, contudo referentes, respectivamente, às heurísticas REUSE, BREADTH e FEWER.

Em ao menos seis casos, as heurísticas computaram resultados ótimos. Na tabela 3, valores da décima quarta, décima quinta e décima sexta colunas marcados com um asterisco ressaltam que a heurística utilizada computou resultados ótimos para exatamente uma instância dentre as dez da classe. As heurísticas computaram resultados sabidamente ótimos em cinco instâncias. É possível que resultados ótimos tenham sido computados pelas heurísticas em mais instâncias, mas, como não foram computados pelos algoritmos exatos, não é possível verificar se os resultados são ótimos. A heurística BREADTH computou resultados ótimos em quatro instâncias de quatro classes diferentes. A heurística REUSE computou o resultado ótimo em uma única instância, a mesma na qual a heurística FEWER obteve resultado ótimo. Assim, resultados ótimos foram computados pelas heurísticas em seis casos distintos, para cinco instâncias diferentes. No total, as heurísticas computaram resultados ótimos em 2.84% das 176 instâncias em que os algoritmos exatos obtiveram resultados.

A heurística FEWER requer menos esforço computacional na etapa de otimização que as demais heurísticas. Isso porque FEWER computa conjuntos Δ de tamanho igual ou menor aos conjuntos computados pelas demais heurísticas. Isso pode ser observado na oitava coluna da tabela 3, em que os valores são sempre iguais ou menores aos da quinta, sexta e sétima colunas.

A heurística BREADTH computa soluções heurísticas mais próximas da solução ótima do que as demais heurísticas. Nos seis casos em que as heurísticas computaram resultados sabidamente ótimos, quatro foram computados pela heurística BREADTH. Na décima quinta coluna da tabela 3, os valores são frequentemente menores que os das décima terceira, décima quarta e décima sexta colunas. Isso se deve ao fato da heurística incluir mais sensores no conjunto Δ que as heurísticas REUSE e FEWER, e, em vários casos, esses fazem parte de soluções melhores. Por vezes, a heurística DKOV computa conjuntos Δ com mais sensores que os conjuntos computados na heurística BREADTH, mas, ainda assim, seus resultados têm valores mais distantes dos resultados ótimos. Isso se deve ao fato de que os sensores do conjunto Δ computado pela heurística BREADTH participam de mais caminhos, sendo mais provavelmente pertencentes a soluções melhores.

Por vezes, as heurísticas BREADTH, DKOV e REUSE podem ser processadas em menos tempo que a heurística FEWER, mesmo computando um conjunto Δ maior. Contudo, esses casos são mais comuns em classes com menores quantidades de POIs e sensores. Na tabela 3, a décima primeira e décima segunda colunas informam, respectivamente, que a heurística BREADTH pode ser processada em menos tempo que a heurística FEWER em três classes com 100 POIs e 100 sensores, em uma classe com 100 POIs e 300 sensores, e em uma classe com 100 POIs e 500 sensores. Em classes com 200 POIs, não foram observados casos em que a heurística BREADTH pode ser processada mais rapidamente que a heurística FEWER. Isso pode ser explicado pelo fato da etapa de fixação poder ser processada mais rapidamente na heurística BREADTH do que na FEWER. Como o custo computacional da etapa de otimização cresce mais rapidamente com a quantidade de POIs e sensores em uma instância que o custo computacional da etapa de fixação, em instâncias maiores, a heurística FEWER pode ser processada mais rapidamente.

5.2.1 Comparação estatística das heurísticas

Comparamos as heurísticas DKOV, REUSE, BREADTH e FEWER pelo método de Carvalho [7]. As primeiras quatro colunas da tabela 4 indicam as quantidades de POIs, sensores e os valores de K e M de todas as instâncias de uma mesma classe. A quinta, sexta, sétima e oitava colunas indicam o *rank* médio das heurísticas DKOV, REUSE, BREADTH e FEWER, respectivamente, nas instâncias de cada classe. A heurística BREADTH tem, em média, *rank*

$\|P\|$	$\|I\|$	K	M	DKOV	REUSE	BREADTH	FEWER
100	100	1	1	**1.7**	2.7	**1.7**	3.9
100	100	2	1	**2.0**	**2.0**	2.6	3.4
100	100	2	2	2.3	2.5	**1.5**	3.7
100	100	3	1	2.5	**1.7**	2.0	3.8
100	100	3	2	3.0	**1.8**	2.3	2.9
100	100	3	3	2.2	3.2	**1.4**	3.2
100	300	1	1	**1.6**	2.9	**1.6**	3.9
100	300	2	1	2.3	2.5	**1.6**	3.6
100	300	2	2	2.5	2.4	**1.7**	3.4
100	300	3	1	2.6	2.2	**1.7**	3.5
100	300	3	2	3.0	2.3	**1.4**	3.3
100	300	3	3	2.8	2.6	**1.0**	3.6
100	500	1	1	1.9	3.0	**1.1**	4.0
100	500	2	1	2.9	2.1	**1.8**	3.2
100	500	2	2	2.6	2.7	**1.0**	3.7
100	500	3	1	2.6	2.3	**1.8**	3.3
100	500	3	2	2.7	2.7	**1.1**	3.5
100	500	3	3	2.1	3.0	**1.1**	3.8
200	100	1	1	**2.1**	2.2	2.2	3.5
200	100	2	1	2.4	**2.1**	2.6	2.9
200	100	2	2	2.0	2.7	**1.8**	3.5
200	100	3	1	**2.0**	2.4	2.9	2.7
200	100	3	2	2.3	2.3	**2.2**	3.2
200	100	3	3	2.3	2.8	**1.7**	3.2
200	300	1	1	2.4	2.7	**1.2**	3.7
200	300	2	1	2.9	**1.7**	2.6	2.8
200	300	2	2	2.2	3.1	**1.2**	3.5
200	300	3	1	3.0	2.3	**1.7**	3.0
200	300	3	2	2.7	3.1	**1.1**	3.1
200	300	3	3	**1.6**	3.0	2.0	3.4
200	500	1	1	2.8	2.3	**1.6**	3.3
200	500	2	1	3.1	**1.8**	2.1	3.0
200	500	2	2	2.3	2.7	**1.3**	3.7
200	500	3	1	3.2	**1.9**	2.0	2.9
200	500	3	2	2.2	2.5	**1.8**	3.5
200	500	3	3	2.3	2.9	**1.1**	3.7
Ranking médio				2.42	2.47	1.71	3.40

Tabela 4: *Ranking* médio das heurísticas por classe de instâncias.

1.708, a heurística DKOV tem *rank* médio 2.419, a heurística REUSE tem em média *rank* 2.475 e a heurística FEWER tem em média *rank* 3.397.

O teste de Friedman pode detectar se uma ou mais heurísticas frequentemente têm *rank* melhor do que as demais. A hipótese-nula (H_0) do teste supõe não haver diferenciação significativa na frequência em que cada heurística tem melhor *rank*, e a hipótese alternativa (H_1) é de que algumas das heurísticas tem *rank* melhor que as demais em frequência suficiente para que $p < 0.05$. No resultado observado do teste, $p = 4.5 * 10^{-}67$, o que rejeita a hipótese-nula. Isso indica que uma ou mais heurísticas frequentemente têm *rank* melhores do que as demais. O teste *post-hoc* de Nemenyi pode ser utilizado para determinar quais são essas heurísticas.

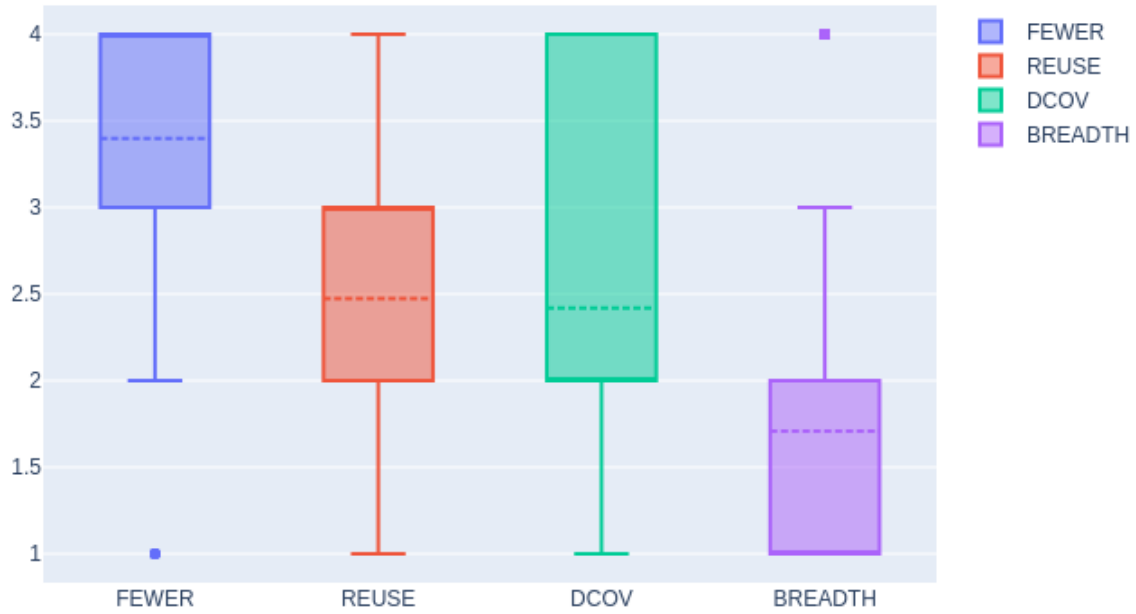
A hipótese nula do teste *post-hoc* de Nemenyi supõe que não há diferenças significativas na média dos valores de *rank* de um dado par de heurísticas. Se rejeitada em favor da hipótese alternativa, as médias dos valores de *rank* de cada heurística do par são significativamente dife-

	DCOV	REUSE	BREADTH	FEWER
DCOV	-	0.900	0.001	0.001
REUSE	0.900	-	0.001	0.001
BREADTH	0.001	0.001	-	0.001
FEWER	0.001	0.001	0.001	-

Tabela 5: Resultados do teste de Nemenyi para cada par de heurísticas.

rentes. Na tabela 5, cada coluna e linha representam um par de heurísticas e cada célula contém o resultado do teste de Nemenyi para o par. As células da diagonal principal da tabela, em que uma heurística é comparada com si mesma, estão preenchidas com o símbolo $-$. Os resultados do teste sugerem que os *ranks* do par de heurísticas $\langle DCOV, REUSE \rangle$ não são significativamente diferentes pois $p > 0.05$, como pode ser visto na primeira e segunda colunas da primeira e segunda linhas da tabela 5. Nas demais linhas e colunas da tabela 5, os resultados sugerem que os *ranks* das heurísticas BREADTH e FEWER são significativamente diferentes das demais, inclusive no par $\langle BREADTH, FEWER \rangle$.

Figura 3: Visualização da distribuição de *rankings* de cada heurística.



Fonte: Autores

Na figura 3, estão representados os rankings médios de cada heurística por meio de gráficos *box-plot*. Os resultados estão organizados em quatro colunas, sendo que, na primeira, temos a distribuição dos valores de *rank* da heurística FEWER, na segunda coluna, a distribuição dos valores da heurística REUSE, na terceira, os da heurística DCOV e, na quarta, o da heurística BREADTH. Em cada coluna, temos um retângulo, sendo que metade dos valores de *rank* da heurística representada na coluna estão entre o limite superior e inferior do retângulo. A média dos valores é representada por uma linha pontilhada. Cada retângulo pode ter uma linha vertical acima e outra abaixo, finalizadas em linhas horizontais. O intervalo entre uma das linhas

horizontais e o retângulo representa 1.5 vezes a distância entre a extremidade mais próxima do retângulo e a linha pontilhada da média. Valores de *rank* acima ou abaixo do intervalo entre as linhas horizontais são considerados pouco representativos da distribuição. Pode-se observar que os valores de *rank* do par de heurísticas REUSE e DCOV estão distribuídos em intervalos que se sobrepõem mais do que os intervalos do par FEWER e REUSE. Pelo teste de Nemenyi, a sobreposição da distribuição de valores de *rank* no par de heurísticas FEWER e REUSE é tão pouca que as médias podem ser consideradas significativamente diferentes, o que sustenta a afirmação que a heurística FEWER tem, em média, resultados piores que os das heurísticas REUSE, DCOV e BREADTH. A maioria dos valores de *rank* da heurística BREADTH estão distribuídos num intervalo que não sobrepõe a maioria dos valores das heurísticas REUSE e DCOV. Pelo teste de Nemenyi, a média dos valores de *rank* da heurística BREADTH é significativamente diferente das médias das heurísticas REUSE, DKOV e FEWER. Por essa razão, pode-se afirmar que a heurística BREADTH tem, em média, resultados melhores que os das demais heurísticas.

Capítulo 6

Conclusões e Trabalhos Futuros

Todas as heurísticas apresentadas neste trabalho utilizam a meta-heurística *fix-and-optimize* para computar rapidamente soluções aproximadas para o problema KCMC. Neste trabalho, são apresentados dois ILPs polinomiais que podem ser resolvidos por um algoritmo *branch-and-bound* para obter soluções ótimas ou para compor a segunda etapa da meta-heurística *fix-and-optimize*. Os ILPs elaborados neste trabalho consideram a propriedade de M-Conectividade tal qual o teorema de Menger, que não exige que o grafo de comunicação de uma instância seja regular. A heurística BREADTH é a melhor heurística apresentada neste trabalho, pois tem, em média, os melhores valores de *rank* e a diferença para as médias das demais heurísticas é estatisticamente significativa.

Trabalhos futuros podem explorar o conceito de *proximidade* entre soluções heurísticas, considerando serem mais próximas as soluções que são compostas por vários caminhos em comum, ou caminhos com mais sensores em comum. Nesse caso, seria possível utilizar métodos como algoritmos genéticos e variação de busca em vizinhanças na etapa de fixação da meta-heurística *fix-and-optimize*. Seria possível também obter resultados sem utilizar a meta-heurística *fix-and-optimize*.

Outros trabalhos futuros podem desenvolver novos ILPs que considerem outros fatores envolvidos em situações reais. Neste trabalho, a comunicação entre sensores é *simétrica*, ou seja, se o sensor i_0 consegue transmitir dados para o sensor i_1, então o sensor i_1 consegue transmitir dados para o sensor i_0. Neste trabalho, a adição de sensores a uma WSN não compromete as propriedades de K-Cobertura e M-Conectividade da rede, o que pode acontecer em situações reais devido a fatores como interferência eletromagnética. Neste trabalho, a limitação de carga de bateria em sensores sem fio é desconsiderada, contudo, a natureza versátil e de baixo custo dos dispositivos torna esse um fator relevante em muitas situações reais. Trabalhos futuros podem desenvolver ILPs que minimizem o consumo de energia em sensores que podem se comunicar de forma assimétrica e interferir na operação de outros sensores na WSN.

Referências

[1] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[2] Jamal N Al-Karaki and Amjad Gawanmeh. The optimal deployment, coverage, and connectivity problems in wireless sensor networks: revisited. *IEEE Access*, 5:18051–18065, 2017.

[3] Ali Jameel Al-Mousawi and Haider K AL-Hassani. A survey in wireless sensor network for explosives detection. *Computers & Electrical Engineering*, 72:682–701, 2018.

[4] Anish Arora, Prabal Dutta, Sandip Bapat, Vinod Kulathumani, Hongwei Zhang, Vinayak Naik, Vineet Mittal, Hui Cao, Murat Demirbas, Mohamed Gouda, et al. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.

[5] M Azharuddin, Pratyay Kuila, and Prasanta K Jana. Energy efficient fault tolerant clustering and routing algorithms for wireless sensor networks. *Computers & Electrical Engineering*, 41:177–190, 2015.

[6] Wafa Barkhoda and Hemmat Sheikhi. Immigrant imperialist competitive algorithm to solve the multi-constraint node placement problem in target-based wireless sensor networks. *Ad Hoc Networks*, page 102183, 2020.

[7] Iago A Carvalho. On the statistical evaluation of algorithmic's computational experimentation with infeasible solutions. *Information Processing Letters*, 143:24–27, 2019.

[8] Iago A Carvalho, Thiago F Noronha, Christophe Duhamel, Luiz FM Vieira, and Vinicius F dos Santos. A fix-and-optimize heuristic for the minmax regret shortest path arborescence problem under interval uncertainty. *International Transactions in Operational Research*, 30(2):1120–1143, 2023.

[9] Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173(33):12, 2005.

[10] Yefim Dinitz. Dinitz'algorithm: The original version and even's version. *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 218–240, 2006.

[11] Yefim A Dinitz. An algorithm for the solution of the problem of maximal flow in a network with power estimation. In *Doklady Akademii nauk*, number 4, pages 754–757. Russian Academy of Sciences, 1970.

[12] Mohammed Farsi, Mostafa A Elhosseini, Mahmoud Badawy, Hesham Arafat Ali, and Hanaa Zain Eldin. Deployment techniques in wireless sensor networks, coverage and connectivity: A survey. *IEEE Access*, 7:28940–28954, 2019.

[13] George HG Fonseca and Túlio AM Toffolo. A fix-and-optimize heuristic for the itc2021 sports timetabling problem. *Journal of Scheduling*, 25(3):273–286, 2022.

[14] Marcelo W Friske, Luciana S Buriol, and Eduardo Camponogara. A relax-and-fix and fix-and-optimize algorithm for a maritime inventory routing problem. *Computers & Operations Research*, 137:105520, 2022.

[15] Amitabha Ghosh and Sajal K Das. Coverage and connectivity issues in wireless sensor networks: A survey. *Pervasive and Mobile Computing*, 4(3):303–334, 2008.

[16] Vitali Gintner, Natalia Kliewer, and Leena Suhl. Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *Or Spectrum*, 27:507–523, 2005.

[17] Govind P Gupta and Sonu Jha. Biogeography-based optimization scheme for solving the coverage and connected node placement problem for wireless sensor networks. *Wireless Networks*, 25(6):3167–3177, 2019.

[18] Suneet Kumar Gupta, Pratyay Kuila, and Prasanta K Jana. Genetic algorithm approach for k-coverage and m-connected node placement in target based wireless sensor networks. *Computers & Electrical Engineering*, 56:544–556, 2016.

[19] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. *Available at https://www.gurobi.com*, 2021.

[20] C Jehan and D Shalini Punithavathani. Potential position node placement approach via oppositional gravitational search for fulfill coverage and connectivity in target based wireless sensor networks. *Wireless Networks*, 23(6):1875–1888, 2017.

[21] Dionisis Kandris, Christos Nakas, Dimitrios Vomvas, and Grigorios Koulouras. Applications of wireless sensor networks: an up-to-date survey. *Applied System Innovation*, 3(1):14, 2020.

[22] Wei-Chieh Ke, Bing-Hong Liu, and Ming-Jer Tsai. Constructing a wireless sensor network to fully cover critical grids by deploying minimum sensors on grid points is np-complete. *IEEE Transactions on Computers*, 56(5):710–715, 2007.

[23] Gaurav Khanna and Sanjay Kumar Chaturvedi. A comprehensive survey on multi-hop wireless networks: milestones, changing trends and concomitant challenges. *Wireless Personal Communications*, 101(2):677–722, 2018.

[24] Ailsa H Land and Alison G Doig. *An automatic method for solving discrete programming problems*. Springer, 1960.

[25] Marcelo Caggiani Luizelli, Weverton Luis da Costa Cordeiro, Luciana S Buriol, and Luciano Paschoal Gaspary. A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining. *Computer Communications*, 102:67–77, 2017.

[26] Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

[27] Segun O Olatinwo and Trudi-H Joubert. Energy efficiency maximization in a wireless powered iot sensor network for water quality monitoring. *Computer Networks*, 176:107237, 2020.

[28] Chiara Petrioli, Michele Nati, Paolo Casari, Michele Zorzi, and Stefano Basagni. Alba-r: Load-balancing geographic routing around connectivity holes in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):529–539, 2014.

[29] Maher Rebai, Hichem Snoussi, Faicel Hnaien, Lyes Khoukhi, et al. Sensor deployment optimization methods to achieve both coverage and connectivity in wireless sensor networks. *Computers & Operations Research*, 59:11–21, 2015.

[30] Divyansh Thakur, Yugal Kumar, Arvind Kumar, and Pradeep Kumar Singh. Applicability of wireless sensor networks in precision agriculture: A review. *Wireless Personal Communications*, 107(1):471–512, 2019.

[31] Abhishek Tripathi, Hari Prabhat Gupta, Tanima Dutta, Rahul Mishra, KK Shukla, and Satyabrat Jit. Coverage and connectivity in wsns: A survey, research issues and challenges. *IEEE Access*, 6:26971–26992, 2018.

[32] Laurence A Wolsey. *Integer programming*. Chapter 7, John Wiley & Sons, 1998.

[33] Qun Zhao and Mohan Gurusamy. Connected k-target coverage problem in wireless sensor networks with different observation scenarios. *Computer Networks*, 52(11):2205–2220, 2008.

[34] Chuan Zhu, Chunlin Zheng, Lei Shu, and Guangjie Han. A survey on coverage and connectivity issues in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2):619–632, 2012.