UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ROBERTO LEONIE FERREIRA MOREIRA

INDEPENDÊNCIA E DOMINAÇÃO EM CAMINHOS: VARIANTES PARA SITUAÇÕES COTIDIANAS

RIO DE JANEIRO
2024

ROBERTO LEONIE FERREIRA MOREIRA

INDEPENDÊNCIA E DOMINAÇÃO EM CAMINHOS: VARIANTES PARA SITUAÇÕES COTIDIANAS

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Vinícius Gusmão Pereira de Sá, D.Sc.

RIO DE JANEIRO
2024

CIP - Catalogação na Publicação

M838i

Moreira, Roberto Leonie Ferreira
Independência e dominação em caminhos: variantes para situações cotidianas / Roberto Leonie Ferreira Moreira. -- Rio de Janeiro, 2024.
53 f.

Orientador: Vinícius Gusmão Pereira de Sá.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2024.

1. Teoria dos grafos. 2. Otimização. 3. Grafo caminho. 4. Conjunto independente. 5. Conjunto dominante. I. Sá, Vinícius Gusmão Pereira de, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

ROBERTO LEONIE FERREIRA MOREIRA

INDEPENDÊNCIA E DOMINAÇÃO EM CAMINHOS: VARIANTES PARA SITUAÇÕES COTIDIANAS

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 23 de agosto de 2024

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
VINICIUS GUSMAO PEREIRA DE SA
Data: 26/08/2024 23:23:12-0300
Verifique em https://validar.iti.gov.br

Prof. Vinícius Gusmão Pereira de Sá
D.Sc. (UFRJ)

gov.br Documento assinado digitalmente
DANIEL SADOC MENASCHE
Data: 26/08/2024 12:38:16-0300
Verifique em https://validar.iti.gov.br

Prof. Daniel Sadoc Menasché
D.Sc. (UFRJ)

gov.br Documento assinado digitalmente
MARIA HELENA CAUTIERO HORTA JARDIM
Data: 26/08/2024 22:30:37-0300
Verifique em https://validar.iti.gov.br

Prof. Maria Helena Cautiero Horta Jardim
D.Sc. (UFRJ)

À Iraci, por ter desempenhado um papel fundamental no meu desenvolvimento enquanto cidadão. Minha eterna gratidão.

AGRADECIMENTOS

À minha mãe, Lusiete Silva, que sempre me incentivou a nunca abandonar os meus objetivos, demostrando ser meu suporte tanto emocional quanto financeiro, permitindo que eu conseguisse me dedicar aos estudos com maior afinco. Sua abnegação foi vital para a construção deste novo capítulo que se inicia em minha vida.

Ao Prof. Vinícius Gusmão, que assumiu altruisticamente o papel de mentor ao longo de minha graduação e por ter se recusado a desistir de mim. Sua confiança e encorajamento foram fundamentais para minha conclusão de curso e realização deste trabalho.

Ao Prof. João Carlos e à COAA, que compreenderam cada dificuldade e obstáculo que enfrentei até aqui. Sua orientação acadêmica e acompanhamento foram cruciais ao longo desta trajetória.

"Winter never fails to turn to spring."

Daisaku Ikeda

RESUMO

O presente trabalho analisa algumas situações do cotidiano que podem ser modeladas como problemas de otimização de cardinalidade em alguns conjuntos específicos de uma determinada classe de grafos, com ênfase nos grafos caminho. Em particular, serão investigados o conjunto independente e o conjunto dominante. Diversas variantes serão empregadas para abordar o problema de ocupação de espaços, com o objetivo de, em geral, evitar ocupações consecutivas e, assim, maximizar ou minimizar a cardinalidade desses conjuntos em cada uma delas. Em vez de simplesmente encontrar o maior ou menor subgrafo induzido por um conjunto de vértices com características específicas, serão estabelecidas métricas de interesse que correspondem com fenômenos observados no mundo real, como uma quantidade que mede o desconforto que um indivíduo sente com a proximidade de outros indivíduos. Essas métricas conduzirão a variantes que, embora possam apresentar soluções convergentes, podem divergir em eficiência. Dado que tais fenômenos podem ser modelados como problemas de otimização, eles sugerem funções-objetivo e conjunto de restrições originais. As variantes propostas serão comparadas por meio de experimentações computacionais para determinar qual delas é mais eficiente na modelagem de um simulador de preenchimento destes espaços, de modo a mantê-lo fidedigno às situações do mundo real, permitindo, se necessário, o relaxamento da restrição de evitar ocupar espaços consecutivos, conforme o desejo do usuário.

Palavras-chave: teoria dos grafos; otimização; grafo caminho; conjunto independente; conjunto dominante.

ABSTRACT

This work analyzes some everyday situations that can be modeled as cardinality optimization problems in some specific sets in a given class of graphs, with emphasis on path graphs. In particular, the independent set and the dominating set will be investigated. Several variants will be employed to address the space occupation problem, with the goal of, in general, avoiding consecutive occupations and, thus, maximizing or minimizing the cardinality of these sets in each of them. Instead of simply finding the largest or smallest subgraph induced by a set of vertices with specific phenomena, considerations of interest will be made that encompass characteristics observed in the real world, such as a quantity that measures the discomfort an individual feels with the proximity of other individuals. These measurements led to variants that, although they may present convergent solutions, may diverge in efficiency. Given that such phenomena can be modeled as optimization problems, they suggest objective functions and sets of original constraints. The proposed variants will be compared through computational experiments to determine which one is more efficient in modeling its spaces-filling simulator, in order to keep it faithful to the real-world situations, allowing, if necessary, the relaxation of the restriction of avoiding occupying continuous spaces, according to the user's desire.

Keywords: graph theory; optimization; path graph; independent set; dominating set.

SUMÁRIO

1 INTRODUÇÃO

Imagine que indivíduos desejam assistir a um filme no cinema ou utilizar os mictórios de um sanitário público. Quais seriam as formas de organizar a utilização da sala de cinema ou do sanitário para otimizar a experiência dos usuários?

1.1 PROPOSTA

O presente trabalho propõe o estudo de algoritmos de otimização em grafos a fim de solucionar problemas corriqueiros do cotidiano. Conceitos como conjunto dominante e conjunto independente de teoria dos grafos serão abordados de maneira a facilitar o entendimento e a modelagem dos problemas, possibilitando a obtenção de possíveis soluções.

Tais problemas do cotidiano possibilitarão ao leitor visualizar aplicações específicas, de modo a reforçar o aprendizado dos conceitos anteriormente mencionados. O estudo apresentará variantes destes problemas com soluções distintas, chegando a soluções globais a partir de contribuições individuais ou locais de seus agentes, tornando mais interessantes os problemas propostos.

1.2 O PROBLEMA DO MICTÓRIO

Pessoas entram em um sanitário público com o objetivo de utilizar os mictórios disponíveis. A forma mais simples de escolher um mictório neste sanitário seria selecioná-lo aleatoriamente. No entanto, isso não reflete o comportamento observado no mundo real.

Na prática, as pessoas desejam utilizar os mictórios sem estarem imediatamente ao lado de outras. Assim, elas costumam adotar uma estratégia de busca por mictórios mais cuidadosa, priorizando mictórios livres e com ambos os lados desocupados. Caso não existam mictórios disponíveis com essa condição, elas buscam aqueles que tenham apenas uma das adjacências (esquerda e direita) ocupadas. Caso não haja nenhum mictório com essa configuração, elas acabam escolhendo aleatoriamente um mictório cujas duas adjacências estejam ocupadas.

1.3 O PROBLEMA DA SALA DE CINEMA

Considere uma sala de cinema e pessoas que chegam. Ao selecionar uma poltrona, as pessoas adotam uma estratégia cautelosa de busca, avaliando as seguintes condições:

- evitar selecionar poltronas disponíveis imediatamente atrás de poltronas já ocupadas, uma vez que sentar-se imediatamente atrás de alguém bloqueia sua visão, impactando negativamente na sua experiência da sessão;

- buscar selecionar poltronas disponíveis com ambos os lados esquerdo e direito desocupados. Se não existir esse tipo de poltronas, então
- buscar selecionar uma poltrona livre que contenha apenas uma poltrona ao lado ocupada, sendo a poltrona à esquerda livre, enquanto a poltrona à direita está ocupada, ou vice-versa. Se não for possível, então
- selecionar aleatoriamente e de forma uniforme uma poltrona livre entre outras duas poltronas já ocupadas.

De modo a facilitar o entendimento, este texto irá se aprofundar na investigação do problema de salas de cinema unidimensionais (1D), assumindo que as poltronas estão inclinadas ou afastadas o suficiente, de modo que uma pessoa não cause incômodo aos espectadores sentados imediatamente atrás. Assim, a primeira condição de avaliar se a poltrona de interesse de um usuário está imediatamente atrás de uma poltrona já ocupada é descartada, tornando o problema da sala de cinema como exatamente o problema do mictório descrito anteriormente. Para facilitar o entendimento, tanto a sala de cinema quanto o sanitário público serão referidos como instalações, enquanto as poltronas e os mictórios serão chamados de recursos da instalação, ou simplesmente recursos.

Ao longo do texto, serão apresentadas abordagens mais ou menos eficientes para preencher as salas de cinema ou sanitários públicos de forma estática. Isto é, sem permitir a saída das pessoas.

Considere um grafo caminho P_n que represente uma instalação com capacidade igual a n, em que os vértices de P_n representam os recursos, enquanto as arestas denotam as relações de adjacência. Algumas dessas abordagens estarão interessadas em encontrar um conjunto independente em P_n, uma vez que, a princípio, as pessoas evitarão utilizar recursos imediatamente ao lado de recursos ocupados. Um *conjunto independente* é um conjunto de vértices tal que nenhum par de vértices no conjunto é adjacente (BALLARD-MYER, 2019). Apesar do problema de encontrar o conjunto independente ser NP-difícil para grafos em geral, ele é trivial para caminhos caso seja resolvido de forma centralizada. Isto é, se todos os agentes possuem um conhecimento global do cenário e estão com as estratégias alinhadas.

Outras abordagens estarão interessadas em permitir o mínimo de recursos a serem utilizados desde que a chegada de um próximo usuário acarrete no incômodo dele e de outro usuário que já esteja na instalação. Assim, estas abordagens podem desejar encontrar um conjunto dominante dos vértices de P_n. Um conjunto $S \subseteq V$ de vértices em um grafo $G = (V, E)$ é denominado *conjunto dominante* se cada vértice $v \in V$ é ou um elemento de S ou adjacente a um elemento de S. (HAYNES; HEDETNIEMI; SLATER, 1998).

1.4 COEFICIENTES DE INCÔMODO

Admita um *array* A de n posições indexadas de 0 a $n-1$ com valores 0 para mictórios disponíveis e 1 para ocupados que represente a configuração de uma instalação e seus recursos em um dado instante de chegada de usuários. A partir dos paradigmas discutidos anteriormente, pode ser importante considerar um índice de incômodo global para o sanitário, a ser detalhado em breve.

1.4.1 Coeficiente individual de incômodo

Considere o grafo $G = (V, E)$ e um subconjunto $S \subset V(G)$ de vértices abaixo.

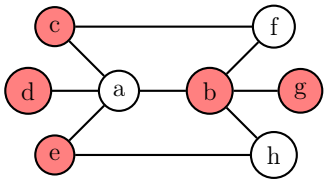

Figura 1 – Um grafo G qualquer e um subconjunto S de vértices em vermelho.

Admita G como uma representação genérica de uma instalação, onde os recursos são representados pelos vértices de $V(G)$. Neste exemplo, os recursos da instalação não precisam estar sequencialmente dispostos como no caminho P_n, uma vez que G permite a existência de ciclos como subgrafo.

Seja S o conjunto dos vértices que representam os recursos em utilização em um determinado instante. Cada usuário que utiliza um recurso representado por um vértice v pode ser incomodado por, no máximo, o grau de v em G. Dessa forma, o grau de v no subgrafo induzido por S indica a quantidade de recursos adjacentes a v que estão ocupados em um dado momento, caracterizando o *coeficiente individual de incômodo* do usuário do recurso representado pelo vértice v. Este coeficiente é denotado pela letra λ.

Para um caminho P_n e um *array* A que representa a ocupação dos recursos, onde $A[v]$ indica se a posição correspondente ao recurso representado pelo vértice v está ocupada ou não, $\lambda(v)$ é definido por

$$\lambda(v) = \begin{cases} A[v+1], \text{ se } v = 0 \\ A[v-1], \text{ se } v = n-1 \\ A[v-1] + A[v+1], \text{ caso contrário,} \end{cases} \tag{1.1}$$

onde v representa um vértice de P_n. É importante ressaltar que o coeficiente mede o incômodo de um indivíduo que utiliza um recurso. Como pode ser facilmente observado, $\lambda(v)$ é limitado superiormente pelo valor 2.

1.4.2 Coeficiente global de incômodo

A ideia central é descobrir uma configuração de S que minimize a soma dos incômodos individuais dos usuários dos recursos representados por cada vértice em S. Essa soma é denominada *coeficiente global de incômodo* da instalação, denotada por Λ e definida por

$$\Lambda(G) = \sum_{v \in G} \lambda(v), \tag{1.2}$$

onde $\lambda(v)$ indica o coeficiente de incômodo individual do indivíduo utilizando o recurso correspondente ao vértice v.

Na figura 1, o conjunto S possui cardinalidade igual a 5. Logo, o grau de um vértice $v \in S$ é limitado superiormente por $|S| - 1$, ou seja, 4. Analisando o grafo G, verifica-se que seu grau máximo $\Delta(G)$ é também igual a 4, o que ocorre nos vértices a e b. Para o subconjunto de vértices $S = \{b, c, d, e, g\}$, os valores de λ são iguais a 0, 0, 0, 1 e 1, respectivamente. Somando esses valores de λ, é obtido o valor $\Lambda(S) = 2$.

Assuma outro subconjunto de vértices $T \subset V(G)$, como ilustrado a seguir.

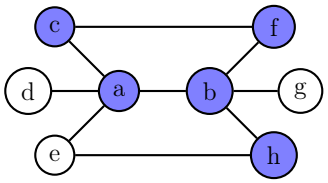

Figura 2 – O grafo G e um subconjunto T de vértices em azul.

Conforme observado na figura 2, o conjunto T também possui cardinalidade igual a 5, limitando o grau máximo de um vértice de T a 4. O conjunto $T = \{a, b, c, f, h\}$ possui valores de λ iguais a 2, 3, 2, 2 e 1, respectivamente. Por conseguinte, o usuário mais incomodado da instalação tem três recursos adjacentes ocupados simultaneamente, enquanto o coeficiente de incômodo global Λ totaliza 10.

Seja k o número de indivíduos que utilizam os recursos da instalação em um dado instante. Formulando o fenômeno observado como um problema de otimização, é obtida a seguinte forma padrão:

$$\begin{array}{ll} \text{minimizar} & \Lambda(S) = \sum_{v \in S} \lambda(v) \\ \text{sujeito a} & |S| = k. \end{array} \tag{1.3}$$

1.5 ORGANIZAÇÃO DO TRABALHO

Após apresentar diferentes estratégias para o preenchimento de uma instalação unidimensional, este trabalho terá como objetivo a comparação, entre as estratégias estudadas, do número de recursos que estarão sendo utilizados sem incômodos até o instante de chegada do primeiro cliente que é obrigado a ocupar um recurso ao lado de alguém. Algumas

estratégias desejarão minimizar este número, enquanto outras desejam sua maximização. Além disso, se buscará calcular o coeficiente de incômodo individual em cada instante de chegada dos usuários, até que o último cliente encontre um recurso livre, mesmo que as adjacências estejam ocupadas.

No capítulo 2 deste texto, serão discutidos os conceitos teóricos fundamentais do estudo apresentado. O capítulo 3 abordará de forma prática as situações mencionadas na introdução, ilustrando diversas estratégias de ocupação de uma sala de cinema ou sanitário público. No capítulo 4, as implementações das soluções propostas para cada estratégia serão comparadas entre si, utilizando recursos computacionais. O capítulo 5 concluirá o estudo e mencionará novas visões que emergem do paradigma unidimensional dos problemas da sala de cinema e do mictório.

2 DEFINIÇÕES E CONCEITOS BÁSICOS

Como mencionado neste texto, serão desenvolvidas aplicações que simulam a utilização de salas de cinema ou sanitários públicos. Essas aplicações visam o estudo da otimização das propriedades de um tipo particular de grafos, os grafos caminho P_n, onde n representa o número de vértices. Antes de avançar, é indispensável a apresentação de alguns conceitos-chave para este trabalho.

2.1 GRAFO CAMINHO

2.1.1 Definição

Formalizando, um grafo caminho P_n é um grafo no qual seus vértices podem ser enumerados em uma sequência $v_0, v_1, \ldots, v_{n-1}$, onde cada aresta é da forma $\{v_i, v_{i+1}\}$, conectando vértices consecutivos, para $i = 0, \ldots, n-2$. Para $n > 2$, P_n possuirá exatamente dois vértices com grau igual a 1 denominados extremidades, enquanto todos os demais vértices terão grau 2. Exemplos de grafos caminho podem ser vistos abaixo.

Figura 3 – Grafo caminho P_1.

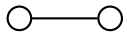

Figura 4 – Grafo caminho P_2.

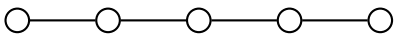

Figura 5 – Grafo caminho P_5.

2.2 CONJUNTO INDEPENDENTE

2.2.1 Definições

Seja $G = (V, E)$ um grafo simples não-direcionado com conjunto de vértices $V = 1, 2, \ldots, n$ e conjunto de arestas E. Um *conjunto independente* (também chamado *conjunto estável* na literatura) de G é um subconjunto S de V tal que dois vértices em S não são adjacentes (KITAEV, 2006). Em suma, cada aresta de G tem no máximo uma extremidade em S. Exemplos de conjunto independente podem ser vistos na imagem a seguir.

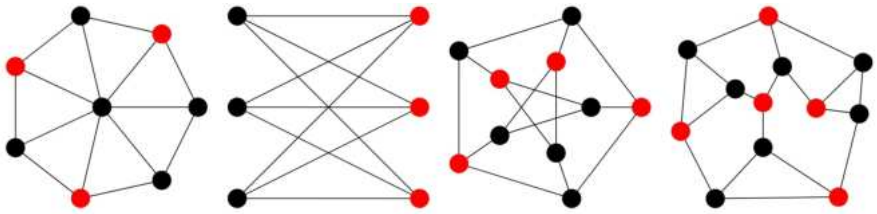

Figura 6 – Conjunto independente (em vermelho) de vários grafos.

Com base na definição de conjunto independente, é pertinente discutir os conceitos de conjunto independente maximal e conjunto independente máximo.

Um *conjunto independente maximal* é um conjunto independente tal que se algum vértice que não esteja no conjunto for adicionado, este não mais será um conjunto independente (BALLARD-MYER, 2019).

Um *conjunto independente máximo* (MIS) é um conjunto independente tal que nenhum dos conjuntos independentes de G é maior (BALLARD-MYER, 2019). O conceito de conjunto independente maximal é relevante para a aplicação que simulará a ocupação de uma sala de cinema ou de um sanitário público.

2.2.2 O problema do conjunto independente

É importante mencionar que o problema de otimização do conjunto independente máximo é NP-difícil para grafos gerais. No entanto, obter o conjunto independente em um grafo caminho de uma forma centralizada é trivial, pois basta escolher posições alternadas a partir da primeira.

2.3 CONJUNTO DOMINANTE

Conceitos como conjunto dominante e conjunto dominante mínimo são valiosos para o desenvolvimento da aplicação que simulará a ocupação de salas de cinema ou de mictórios em um sanitário público. Na próxima subseção, ambos serão discutidos.

2.3.1 Definição

Um *conjunto dominante* D é um conjunto de vértices tal que cada vértice de G ou está em D ou possui pelo menos um vizinho em D (SHUKLA; THAKUR, 2020).

Um conjunto dominante é minimal se a remoção de qualquer vértice desse conjunto violar sua propriedade de dominação. Exemplos de conjunto dominante e conjunto dominante mínimo estão ilustrados na figura abaixo.

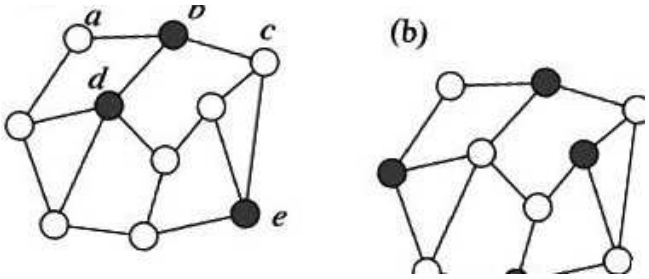

Figura 7 – Exemplos de conjunto dominante mínimo (a) e conjunto dominante (b) em cinza.

2.3.2 O problema do conjunto dominante

O problema do conjunto dominante consiste em determinar se $\gamma(G) \leq k$, onde o grafo G e k são suas entradas, caracterizando-se como um problema de decisão NP-completo. Dessa forma, acredita-se não existir um algoritmo computacionalmente eficiente que calcule $\gamma(G)$ para qualquer grafo G. Entretanto, existem algoritmos aproximativos eficientes e algoritmos exatos para certas classes de grafos.

O problema do conjunto dominante mínimo pode ser visto como uma variante específica de conjuntos *k-ruling*, ou, alternativamente, cada instância de conjuntos *k-ruling* pode ser considerada um relaxamento do problema do conjunto dominante mínimo, sendo, nesse caso, um conjunto 2-*ruling*. Dado um grafo $G = (V, E)$, um conjunto *k-ruling* é um subconjunto $S \subseteq V$ tal que todo vértice em $V \setminus S$ está no máximo distante k de S (KOSTER, 2022).

2.4 RELAÇÕES ENTRE INDEPENDÊNCIA E DOMINAÇÃO EM GRAFOS

Existem propriedades e relações entre os princípios de conjunto independente e conjunto dominante em um grafo que são importantes de serem destacadas.

1. Um conjunto independente é também um conjunto dominante se, e somente se, for um conjunto independente maximal. Isso implica que qualquer conjunto independente maximal de um grafo é necessariamente também um conjunto dominante.

   Prova: Seja S um conjunto independente de um grafo $G = (V, E)$ que também é um conjunto dominante. Isso significa que todo vértice $v \in V(G)$ pertence a S ou tem um vizinho em S. Será demonstrado que S é um conjunto independente maximal.

   Suponha que S não seja maximal. Então existe um vértice $w \in V(G)$ tal que $w \notin S$, mas que pode ser adicionado a S sem violar sua propriedade de independência. No entanto, como S é um conjunto dominante, w deve ter um vizinho u em S, contradizendo a hipótese de que S não é maximal. Portanto, S é um conjunto independente maximal.

Por outro lado, considere S um conjunto independente maximal do grafo G. Será demonstrado que S é também um conjunto dominante.

Suponha que S não seja um conjunto dominante. Logo, existe um vértice $v \in V(G)$ tal que $v \notin S$ e que não é vizinho de nenhum vértice em S. Todavia, dado que S é um conjunto independente maximal, v deve ser adjacente a algum vértice de S, o que leva a uma contradição. Assim, conclui-se que S é um conjunto dominante. Mais detalhes sobre as relações entre conjunto dominante independente e conjunto independente maximal podem ser acessados em (LIU; POON; LIN, 2014).

■

2. Um conjunto dominante independente é um conjunto de vértices que é simultaneamente um conjunto independente e um conjunto dominante de vértices de um grafo $G = (V, E)$. Isso é equivalente a um conjunto independente maximal de vértices [1].

   Prova: Assuma S como um conjunto dominante independente. Logo, S é simultaneamente um conjunto dominante e um conjunto independente. Como S é dominante, cada vértice $v \in V(G)$ pertence a S ou é adjacente a pelo menos um vértice de S. Suponha que S não seja maximal. Isto é, existe um vértice $u \in V(G)$ que pode ser adicionado a S sem violar sua propriedade de independência. Uma vez que S é um conjunto dominante, u deve ser adjacente a pelo menos um vértice w em S. Se u e w são adjacentes, isso viola a propriedade de independência, resultando em uma contradição. Portanto, S é maximal.

   Em contrapartida, considere S como um conjunto independente maximal. Por definição, S é independente. Será demonstrado que S é um conjunto dominante. Para todo vértice $v \in V(G)$, deve existir um vértice $u \in S$ que domina v. Seja v pertencente a S ou dominado por algum vértice em S diferente de v. Se v pertence a S, um vértice é dominado por si mesmo, logo v é dominado. Se v não pertence a S, existe um vértice u em S que domina v, uma vez que S é maximal. Portanto, todo vértice em $V(G)$ é dominado por S, o que significa que S é um conjunto dominante.

   ■

3. O conjunto dominante mínimo de um grafo G não necessariamente é independente. No entanto, a cardinalidade de um conjunto dominante mínimo é sempre menor ou igual à cardinalidade de um conjunto independente maximal mínimo.

   Prova: Um conjunto dominante pode conter vértices que são adjacentes entre si, como nos ciclos ímpar ($C_{2k+1}, k = 0, 1, 2, \dots$), por exemplo. Por outro lado, um conjunto independente não pode conter vértices adjacentes. Um conjunto dominante

[1] Um conjunto que é independente e dominante também é um conjunto independente maximal. Isso ocorre porque se adicionássemos outro vértice ao conjunto, ele seria adjacente a pelo menos um vértice já no conjunto, tornando-o não mais independente (LOPEZ, 2022).

mínimo é o menor conjunto possível que é dominante. Da mesma maneira, um conjunto independente maximal mínimo é o menor conjunto independente maximal possível.

Um conjunto dominante deve abranger todos os vértices que não são adjacentes a outros vértices do conjunto. Como um conjunto independente maximal inclui vértices não adjacentes entre si, a cardinalidade do conjunto dominante mínimo é menor ou igual do que a cardinalidade de um conjunto independente maximal do grafo. No entanto, um conjunto dominante, ainda que mínimo, não necessariamente é independente. Analise a figura a seguir. Embora existam outras possibilidades de conjuntos dominantes mínimos que sejam independentes no grafo ilustrado, este serve como um contraexemplo de conjunto dominante mínimo que não é independente.

■

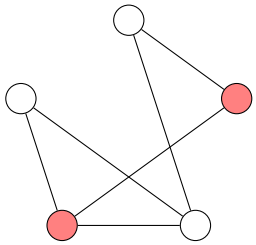

Figura 8 – Em vermelho, exemplo de conjunto dominante mínimo que não é independente.

O próximo capítulo apresentará abordagens voltadas para a otimização da cardinalidade de tipos de conjuntos independente e dominante, além de explorar um problema de minimização da cardinalidade de um subconjunto qualquer em uma outra abordagem.

2.5 OTIMIZAÇÃO

A *Otimização Matemática* é o processo de

(i) formulação e

(ii) solução de um problema de otimização com restrições da forma matemática geral:

$$\underset{\text{em relação a } x}{\text{minimizar}} \; f(x), x = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n$$

sujeito às restrições:

$$g_j(x) \leq 0, \quad j = 1, 2, \ldots, m \tag{2.1}$$

$$h_j(x) = 0, \quad j = 1, 2, \ldots, r \tag{2.2}$$

onde $f(x)$, $g_j(x)$ e $h_j(x)$ são funções escalares do vetor coluna real x (SNYMAN; WILKE, 2018). $f(x)$ é chamada de função-objetivo, enquanto $g_j(x)$ e $h_j(x)$ são denominadas restrições do problema.

A aplicação que simula a ocupação de uma sala de cinema ou de um sanitário público busca otimizar a cardinalidade de um subgrafo induzido em um grafo caminho P_n que representa a instalação, configurando assim um problema de otimização inteira. Esse subgrafo pode ser um conjunto dominante ou independente.

Problemas de otimização inteira podem exigir um árduo esforço computacional para a sua resolução, como nos casos do Problema do Caixeiro Viajante e do Problema da Árvore de Steiner. Para esses problemas, porém, existem outras abordagens, baseadas em programação inteira mista com variáveis binárias, que fazem uso de um número polinomial de restrições e variáveis (MACAMBIRA et al., 2022).

Embora estudos aprofundados sobre otimização em grafos ponderados não sejam utilizados na modelagem do processo de ocupação dos recursos de uma instalação, é possível encontrar caminhos que são subgrafos de grafos gerais através da resolução de problemas de otimização inteira. Mais detalhes sobre programação inteira mista para problemas de otimização em grafos podem ser consultados em (MACAMBIRA et al., 2022).

Dado que se busca configurar o subgrafo de maneira a maximizar ou minimizar sua cardinalidade de acordo com uma abordagem específica, atendendo as restrições de utilização dos recursos da instalação representada, o problema se caracteriza como um problema de otimização combinatória em grafos, mais precisamente como um problema de otimização online.

2.5.1 Otimização online

Um problema de otimização online é um em que a instância do problema é revelada de forma incremental. Decisões podem ou devem ser feitas antes que toda a instância do problema seja revelado. Qualquer algoritmo que dê uma solução para um problema online é denotado como um algoritmo online (WAGNER, 2006).

A otimização online é um tema de grande interesse na ciência da computação e pesquisa operacional devido a sua aplicabilidade em problemas de alocação dinâmica de recursos. Em muitos sistemas do mundo real, os processos a serem otimizados não revelam a todas as informações de forma antecipada, caracterizando o problema como um problema de otimização online, como em sistemas com chegadas contínuas de clientes. Mais detalhes sobre problemas de otimização online podem ser encontrados em (LU, 2013).

3 PROBLEMAS PRÁTICOS

3.1 APRESENTAÇÃO

O presente capítulo apresentará diversas abordagens de ocupação de um *array* unidimensional, simulando o preenchimento de uma instalação, levando em conta as adjacências dos recursos utilizados nestas instalações. Uma posição receberá o valor 1 se o seu recurso correspondente estiver ocupado. Caso contrário, a posição receberá 0.

Inicialmente, será apresentada a abordagem completamente aleatória exatamente igual à busca aleatória por poltronas do problema da sala de cinema, na qual os usuários selecionam recursos de forma completamente aleatória.

Em seguida, considere uma nova abordagem em que o indivíduo escolhe seu recurso ignorando as seleções dos próximos clientes. No entanto, ao escolher entre os recursos disponíveis, os clientes evitam aqueles imediatamente ao lado de recursos já ocupados. Essa abordagem será chamada de individualista. Será visto futuramente que esta abordagem busca encontrar um conjunto dominante independente minimal no grafo caminho que representa a instalação, denotado por P_n, onde n é a capacidade da instalação.

Uma terceira abordagem é chamada de abordagem altruísta. Conforme o nome sugere, nesta abordagem, seus usuários são bem-intencionados ou altruístas e selecionam um recurso considerando as escolhas dos próximos usuários, que também serão altruístas. Essa abordagem visa maximizar a quantidade de usuários simultâneos da instalação sem causar incômodos. Em outras palavras, busca-se identificar o conjunto independente máximo do grafo P_n citado anteriormente, pois deseja-se maximizar seu conjunto independente maximal. Como essa abordagem conta com usuários altruístas que visam a maximização da utilização da instalação, a seleção de recursos pelos usuários precisa ser cuidadosa.

Admita ainda uma abordagem onde, a cada instante em que os indivíduos selecionam recursos, seja calculada a quantidade de opções de recursos da instalação consideradas vantajosas, ou seja, recursos disponíveis sem adjacências ocupadas. Esta abordagem pode ser chamada de individualista-altruísta, pois, de maneira geral, o cliente seleciona recursos disponíveis com adjacências livres, buscando deixar o maior número possível desses recursos disponíveis para os próximos clientes. O objetivo desta abordagem é maximizar, a cada instante, a quantidade de opções de recursos vantajosos disponíveis.

Uma outra abordagem a ser apresentada é conhecida como abordagem do faxineiro. Visando ter o mínimo de esforço na limpeza da instalação, o faxineiro deseja que o menor número possível de indivíduos utilizem os recursos. Assim, essa abordagem busca encontrar o conjunto dominante independente mínimo do grafo P_n que represente a instalação.

É importante ressaltar que as abordagens individualista, altruísta e do faxineiro terão como critério de parada a condição de não haver mais recursos com adjacências livres.

Isto é, quando todos os recursos disponíveis tiverem pelo menos um recurso adjacente ocupado.

Como última abordagem, considere o cenário em que o usuário pode escolher um recurso livre, ainda que este tenha recursos adjacentes ocupados. Ao selecionar um recurso, o usuário busca minimizar seu próprio incômodo, optando assim por um recurso com o maior número possível de adjacências livres, podendo ainda optar por recursos que já tenham alguma adjacência ocupada ou em utilização. Para este paradigma, pode-se avaliar o coeficiente de incômodo global da instalação. Esse coeficiente é obtido por meio do somatório da quantidade de adjacências ocupadas para cada recurso da instalação. A presente abordagem será tratada como um problema de minimização que buscará uma configuração da instalação que minimize o coeficiente de incômodo global da instalação a cada chegada de indivíduos.

No decorrer deste capítulo, serão apresentadas seções dedicadas ao aprofundamento dos conceitos de cada uma dessas abordagens, acompanhadas de pseudocódigos que implementam suas respectivas soluções.

3.2 ABORDAGEM COMPLETAMENTE ALEATÓRIA

Considere a situação a seguir. Logo após a abertura da instalação, o primeiro usuário chega e seleciona aleatoriamente um recurso para utilização. Após a ocupação deste recurso, os próximos usuários chegam e também selecionam aleatoriamente outros recursos disponíveis. Essa abordagem possui como objetivo apenas permitir a seleção aleatória de um recurso por um usuário, se configurando assim como a abordagem mais ingênua de todas a serem exploradas no texto.

3.2.1 Execução

Para facilitar a compreensão do leitor, admita uma instalação inicialmente vazia com cinco recursos numerados de 0 a 4. Em seguida, um *array* de cinco posições é inicializado com cada posição assumindo o valor 0, assim como uma lista auxiliar contendo valores inteiros de 0 a 4, utilizada para a seleção aleatória dos recursos.

0	0	0	0	0

Tabela 1 – Um *array* de cinco posições que representa a instalação inicialmente vazia.

0	1	2	3	4

Tabela 2 – Lista auxiliar utilizada para a seleção aleatória dos recursos.

Esta lista removerá os valores à medida que forem sendo selecionados. Suponha que o primeiro indivíduo seleciona aleatoriamente o valor 2 da lista. Logo, as configurações do *array* de ocupação dos recursos e da lista auxiliar são as seguintes:

0	0	1	0	0

Tabela 3 – O *array* de cinco posições com o recurso 2 em utilização.

0	1	3	4

Tabela 4 – Lista auxiliar sem o elemento 2.

Suponha que um segundo usuário selecione, ao acaso, o recurso número 0. As estruturas são atualizadas como segue:

1	0	1	0	0

Tabela 5 – O *array* de cinco posições com os recursos 0 e 2 sendo utilizados.

1	3	4

Tabela 6 – Lista auxiliar atualizada com a remoção do elemento 0.

Admitindo que um novo usuário seleciona o recurso 3, as estruturas são atualizadas para a forma abaixo.

1	0	1	1	0

Tabela 7 – O *array* de cinco posições com os recursos 0, 2 e 3 sendo utilizados.

1	4

Tabela 8 – Lista auxiliar atualizada após a remoção do elemento 3.

Será visto em breve que as abordagens podem ou não permitir a ocupação de todos os recursos da instalação, dependendo da escolha do usuário. Se o usuário opta pela ocupação total, o algoritmo encerrará quando o *array* que representa a instalação estiver preenchido somente com valores 1.

3.2.2 Pseudocódigo

A entrada do algoritmo que implementa a abordagem completamente aleatória é um número inteiro n, que representa a capacidade da instalação e uma variável booleana que

indica se a abordagem irá permitir a ocupação todos os recursos ou não. Sua saída será o número de passos até o fim da execução e o coeficiente global de incômodo da instalação.

A seguir, é apresentado um pseudocódigo que implementa a abordagem completamente aleatória discutida, acompanhado de uma breve explicação de seu funcionamento.

Algorithm 1 ABORDAGEM COMPLETAMENTE ALEATÓRIA

```
1: função CompletamenteAleatoria(n, ocupaTodos):
2:     instalacao ← [0, 0, ..., 0]
3:     listaIndices ← [0, 1, ..., n-1]
4:     incomodo ← 0
5:     quantPassos ← 0
6:     enquanto listaIndices ≠ [] faça
7:         se ocupaTodos = Verdadeiro então
8:             indice ← Elemento aleatório de listaIndices
9:             instalacao[indice] ← 1
10:            Remova indice de listaIndices
11:            quantPassos ← quantPassos + 1
12:        senão
13:            se [1, 1] não é sublista de instalacao então
14:                indice ← Elemento aleatório de listaIndices
15:                instalacao[indice] ← 1
16:                Remova indice de listaIndices
17:                quantPassos ← quantPassos + 1
18:            senão
19:                Pára
20:            fim se
21:        fim se
22:    fim enquanto
23:    i ← 0
24:    enquanto i < n − 1 faça
25:        se instalacao[i:i+2] = [1, 1] então
26:            incomodo ← incomodo + 1
27:        fim se
28:        i ← i + 1
29:    fim enquanto
30:    devolve quantPassos, incomodo
31: fim função
```

Na linhas 2 e 3, o pseudocódigo inicializa o *array* `instalacao` com n posições nulas e cria uma lista de índices a serem selecionados aleatoriamente, em que o elemento selecionado será removido a cada etapa enquanto a lista de índices contiver elementos, como pode ser observado na linha 6. Além disso, as variáveis `quantPassos` e `incomodo` são inicializadas com o valor 0 nas linhas 4 e 5. `quantPassos` é responsável por armazenar o número de passos do algoritmo.

Conforme já mencionado, é verificado na linha 6 se `listaIndices` contém algum elemento. Enquanto tiver, é verificado na linha 7 se o usuário optou pela ocupação de

todos os recursos. Caso verdade, um elemento de `listaIndices` é sorteado aleatoriamente na linha 8 e a posição correspondente em `instalacao` recebe o valor 1 na linha 9. Em seguida, o elemento sorteado é removido de `listaIndices` na linha 10 e `quantPassos` é incrementado de 1 na linha 11. Caso contrário, se ainda não existirem recursos adjacentes ocupados em `instalacao` como testado na linha 13, a subrotina citada anteriormente também é executada. Ao final da execução a partir da linha 23, o coeficiente global de incômodo é calculado para ser retornado junto do valor da variável `quantPassos`.

3.3 ABORDAGEM INDIVIDUALISTA

Nesta abordagem, os indivíduos que chegam à instalação evitam utilizar recursos imediatamente ao lado dos já ocupados, dificultando a escolha de recursos dos próximos usuários.

Em outras palavras, os usuários optam por recursos sem adjacências ocupadas, sem se importar com a disponibilidade de outros recursos com estas mesmas características disponíveis para os próximos usuários. Assim, esta abordagem individualista possui como objetivo a obtenção de um conjunto dominante independente que minimize, quando possível, os coeficientes individuais de incômodo dos usuários da instalação, ou simplesmente um conjunto dominante independente minimal.

3.3.1 Execução

Como mencionado anteriormente, a abordagem pode ou não encerrar quando não houver mais recursos livres com adjacências também livres, dependendo da escolha do usuário. A lista de índices será inicializada conforme visto anteriormente. Entretanto, ao selecionar uma posição i tal que $0 < i < n - 1$, também serão removidas as posições $i - 1$ e $i + 1$ em cada passo, assegurando que posições adjacentes às ocupadas não sejam selecionadas futuramente. Para $i = 0$, que corresponde à extremidade mais à esquerda da instalação, apenas a posição $i + 1$ da lista de índices será removida. O processo é similar para $i = n - 1$, correspondendo à extremidade mais à direita.

Para garantir a clareza ao leitor, considere novamente a instalação com cinco recursos disponíveis, inicialmente vazios. Suponha que o primeiro usuário seleciona o recurso número 1. As configurações do *array* de ocupação dos recursos e da lista auxiliar de índices são as seguintes:

0	1	0	0	0

Tabela 9 – *Array* com cinco posições, onde o recurso 1 está em uso.

3	4

Tabela 10 – Lista auxiliar sem o elemento sorteado 1 e seus vizinhos 0 e 2.

Observe que três elementos foram removidos da lista de recursos disponíveis, restando somente os recursos de índices 3 e 4 para o próximo indivíduo escolher. Suponha que este próximo indivíduo tenha selecionado o recurso 4. Assim, a lista ficará vazia. A configuração atual do *array* de ocupação de recursos e da lista auxiliar pode ser vista abaixo.

0	1	0	0	1

Tabela 11 – *Array* de cinco posições representando os recursos 1 e 4 em uso.

Tabela 12 – Lista auxiliar completamente vazia.

Considere um grafo caminho P_5 para representar a instalação atual. Admitindo um subconjunto $S \subset P_5$ que contém os vértices que representam os recursos em uso, é obtido o conjunto de vértices representado na figura abaixo. O conjunto S encontrado é um conjunto dominante independente mínimo de P_5, que por sua vez é um conjunto dominante independente minimal.

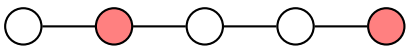

Figura 9 – Em vermelho, um conjunto dominante independente minimal do grafo caminho P_5.

Imagine agora uma instalação com capacidade para 10 recursos. Suponha que o primeiro indivíduo selecione o recurso representado pelo índice 7. O *array* que indica os recursos ocupados e a lista auxiliar de índices para seleção serão atualizados para a configuração a seguir:

0	0	0	0	0	0	0	1	0	0

Tabela 13 – *Array* de dez posições, com o recurso 7 em utilização.

0	1	2	3	4	5	9

Tabela 14 – Lista auxiliar sem o elemento selecionado 7 e seus vizinhos 6 e 8.

Suponha que agora um segundo indivíduo escolha o recurso de índice 4. As estruturas auxiliares foram atualizadas após essa escolha e possuem a configuração abaixo.

0	0	0	0	1	0	0	1	0	0

Tabela 15 – *Array* de dez posições, com os recursos 4 e 7 em utilização.

0	1	2	9

Tabela 16 – Lista auxiliar atualizada sem o elemento selecionado 4 e seus vizinhos 3 e 5.

Considere que o próximo indivíduo que entra na instalação seleciona o recurso 0. As estruturas auxiliares são então atualizadas para a configuração abaixo.

1	0	0	0	1	0	0	1	0	0

Tabela 17 – *Array* de dez posições, com os recursos 0, 4 e 7 em uso.

2	9

Tabela 18 – Lista auxiliar atualizada sem o elemento selecionado 0 e seu vizinho 1.

Note que não foi necessário deletar da lista auxiliar o índice do recurso imediatamente à esquerda do recurso de número 0, pois este recurso não existe, sendo suficiente remover apenas o índice do recurso imediatamente à direita, ou seja, o índice 1.

Admita que o próximo indivíduo selecione o recurso 2. Essa seleção atualizará as estruturas auxiliares da maneira a seguir:

1	0	1	0	1	0	0	1	0	0

Tabela 19 – *Array* de dez posições, com os recursos 0, 2, 4 e 7 em uso.

9

Tabela 20 – Lista auxiliar atualizada com a remoção do índice selecionado 2.

Por fim, resta ao último indivíduo entrar na instalação e selecionar o recurso 9, atualizando as estruturas auxiliares conforme mostrado abaixo. Uma vez que o recurso 9 foi selecionado, o índice correspondente é removido da lista de índices, deixando-a vazia.

1	0	1	0	1	0	0	1	0	1

Tabela 21 – *Array* de dez posições com os recursos 0, 2, 4, 7 e 9 em utilização.

Tabela 22 – Lista auxiliar vazia.

Para representar a instalação com dez recursos, considere o grafo caminho P_{10}. Dessa forma, é possível identificar um conjunto $S \subset P_{10}$ que contém os vértices representados pelo valor 1 no *array* de dez posições, como ilustrado na figura abaixo. Nessa representação, observa-se que S é um conjunto dominante independente minimal, pois a remoção de qualquer vértice de S violaria sua propriedade de dominação.

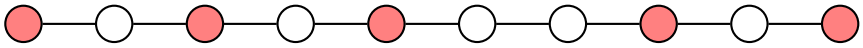

Figura 10 – Em vermelho, o conjunto dominante independente minimal $S \subset P_{10}$.

3.3.2 Pseudocódigo

Na abordagem individualista, o algoritmo recebe como parâmetro de entrada um inteiro n, que representa a capacidade da instalação. Em ambos os exemplos anteriores, o usuário do simulador decidirá através de uma variável booleana se o programa permitirá a ocupação total dos recursos. O algoritmo retorna o número de passos até o término de sua execução ou simplesmente a cardinalidade do conjunto dominante independente minimal encontrado, além do coeficiente global de incômodo calculado. Uma proposta de pseudocódigo que implementa a abordagem individualista pode ser vista a seguir.

Algorithm 2 ABORDAGEM INDIVIDUALISTA

```
1: função INDIVIDUALISTA(n, ocupaTodos):
2:     instalacao ← [0, 0, ..., 0]
3:     listaIndices ← [0, 1, ..., n − 1]
4:     quantPassos ← 0
5:     incomodo ← 0
6:     enquanto listaIndices ≠ [] faça
7:         indice ← Elemento aleatório de listaIndices
8:         instalacao[indice] ← 1
9:         Remova indice de listaIndices
10:        se ocupaTodos = Falso então
11:            se indice > 0 e indice-1 está em listaIndices então
12:                Remova indice-1 de listaIndices
13:            fim se
14:            se indice < n-1 e indice+1 está em listaIndices então
15:                Remova indice+1 de listaIndices
16:            fim se
17:        fim se
18:        quantPassos ← quantPassos + 1
19:    fim enquanto
20:    i ← 0
21:    enquanto i < n − 1 faça
22:        se instalacao[i:i+2] = [1, 1] então
23:            incomodo ← incomodo + 1
24:        fim se
25:        i ← i + 1
26:    fim enquanto
27:    devolve quantPassos, incomodo
28: fim função
```

O algoritmo inicializa nas linhas 2 a 5 as estruturas auxiliares e as variáveis de saída que contarão o número de passos executados e o coeficiente global de incômodo, respectivamente. Como observado na linha 6, enquanto houver elementos na lista auxiliar que permite a seleção de índices dos recursos, um índice é sorteado dessa lista na linha 7 para que seja atribuído o valor 1 à posição correspondente no *array* de ocupação de recursos na linha 8. O índice selecionado é removido de `listaIndices` na linha 9.

Caso o usuário não permita a ocupação total dos recursos da instalação, então a partir do índice selecionado, verifica-se a adjacência do recurso correspondente. Para o teste na linha 11 que verifica se o índice escolhido não corresponde ao recurso mais à esquerda na instalação, é avaliado na mesma linha se o recurso imediatamente à esquerda ainda possui índice em `listaIndices`, ou seja, se está disponível. Se verdadeiro, o índice é removido de `listaIndices` na linha 12. O processo é análogo na linha 14 para o caso do índice não corresponder ao recurso mais à direita na instalação. A variável que conta o número de passos é incrementada de 1 na linha 18, o cálculo do coeficiente global de incômodo é

realizado nas linhas 20 a 26 e ambos são retornados ao fim da execução.

3.4 ABORDAGEM ALTRUÍSTA

A abordagem altruísta parte do princípio de que os usuários dos recursos são bem-intencionados ou altruístas. Cada usuário busca maximizar a quantidade simultânea de recursos em uso, sem ocupar recursos adjacentes e sem causar incômodo a outros usuários.

Representando uma instalação por meio de um grafo caminho P_n, esta abordagem possui como objetivo a obtenção do conjunto independente máximo.

3.4.1 Conceituação

Admitindo que os indivíduos possuam conhecimento total da instalação e dos recursos disponíveis, a determinação do conjunto independente máximo de P_n é realizada de forma trivial. Seja P_n um caminho com n vértices. Para $n = 1$, o conjunto independente máximo de P_1 é claramente o próprio grafo P_1. Para $n = 2$, o caminho P_2 apresenta duas opções de conjunto independente máximo, cada uma contendo uma das extremidades.

Figura 11 – O conjunto independente máximo do caminho P_1.

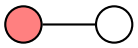

Figura 12 – Um conjunto independente máximo do caminho P_2.

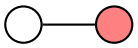

Figura 13 – Um outro conjunto independente máximo do caminho P_2.

Cada aresta de P_n requer que apenas uma de suas extremidades pertença ao conjunto independente máximo $S \subset P_n$, sendo possível que arestas consecutivas compartilhem um vértice v tal que $v \in S$. No entanto, é proibido que ambas as extremidades de uma mesma aresta de P_n estejam em S, pois isso violaria sua propriedade de independência. Assim, os vértices de S não são adjacentes entre si, estando distantes por pelo menos duas arestas ou um vértice que não pertence a S.

Como resultado, pode-se inferir que o conjunto independente máximo S de um caminho P_n é obtido simplesmente alternando por entre seus vértices. Logo, sua cardinalidade é de $\lceil n/2 \rceil$.

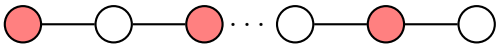

Figura 14 – Um conjunto independente máximo do caminho $P_{2k}, k = 0, 1, 2, \ldots$.

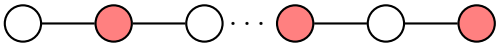

Figura 15 – Outro conjunto independente máximo do caminho $P_{2k}, k = 0, 1, 2, \ldots$.

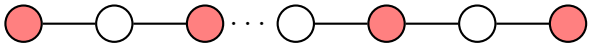

Figura 16 – O conjunto independente máximo do caminho $P_{2k+1}, k = 0, 1, 2, \ldots$.

3.4.2 Execução

Com base nas informações anteriores, o conjunto independente máximo do caminho P_n, que representa a instalação, pode ser armazenado em uma estrutura auxiliar. Essa estrutura auxiliar será a própria lista de índices a serem selecionados, sendo esta a forma mais simples de restringir a seleção dos recursos. Como nas abordagens vistas previamente, selecionar um índice da lista implica na sua remoção.

Considere uma instalação com capacidade ímpar. Para facilitar a compreensão, tome $n = 9$. O *array* com 9 posições a ser ocupado e a lista de índices auxiliar são inicializados com a configuração abaixo.

0	0	0	0	0	0	0	0	0

Tabela 23 – Instalação inicialmente vazia, com nove recursos disponíveis.

0	2	4	6	8

Tabela 24 – Lista auxiliar contendo apenas os índices dos vértices pertencentes ao conjunto independente máximo do grafo P_9.

Suponha que o primeiro usuário selecione o recurso número 0. O elemento 0 é removido da lista auxiliar e o valor 1 é atribuído à posição 0 do *array* que representa a instalação.

1	0	0	0	0	0	0	0	0

Tabela 25 – Instalação com o recurso 0 em uso.

2	4	6	8

Tabela 26 – Lista auxiliar atualizada com a remoção do índice 0.

Um segundo indivíduo seleciona o recurso 6, atualizando as estruturas auxiliares conforme pode ser observado a seguir.

1	0	0	0	0	0	1	0	0

Tabela 27 – Instalação com os recursos 0 e 6 em uso.

2	4	8

Tabela 28 – Lista auxiliar atualizada com a remoção do índice 6.

Um novo usuário escolhe o recurso 2, atualizando as estruturas conforme mostrado abaixo.

1	0	1	0	0	0	1	0	0

Tabela 29 – Instalação com os recursos 0, 2 e 6 em uso.

4	8

Tabela 30 – Lista auxiliar atualizada com a remoção do índice 2.

O penúltimo indivíduo a entrar na instalação seleciona o recurso 4. As estruturas auxiliares são atualizadas da seguinte forma:

1	0	1	0	1	0	1	0	0

Tabela 31 – Instalação com os recursos 0, 2, 4 e 6 em utilização.

8

Tabela 32 – Lista auxiliar atualizada com a remoção do índice 4.

Ao último indivíduo cabe utilizar o recurso de número 8, configurando os estados finais do *array* que representa a instalação e da lista de índices auxiliar conforme abaixo.

1	0	1	0	1	0	1	0	1

Tabela 33 – Instalação com os recursos 0, 2, 4, 6 e 8 em uso.

Tabela 34 – Lista auxiliar vazia.

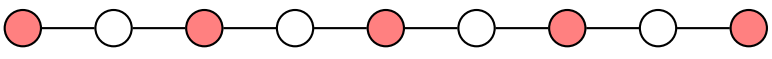

Figura 17 – Caminho P_9 que representa a instalação com nove recursos, onde o conjunto independente máximo (recursos ocupados) está indicado em vermelho.

É importante lembrar que, quando n é par, existem duas possibilidades de um conjunto independente máximo S. S pode ou não incluir o vértice mais à esquerda de P_n. Se S incluir o vértice mais à esquerda, então não pode incluir o vértice mais à direita. Caso contrário, S deve obrigatoriamente conter o vértice mais à direita.

Uma vez que existem duas possibilidades de conjunto independente máximo para $P_{2k}, k = 0, 1, 2, \ldots,$ pode-se sortear uma delas para trabalhar. Suponha que o conjunto independente máximo que inclui o vértice mais à esquerda tenha sido sorteado. Além disso, considere que $k = 4$ ou $n = 2k = 8$, e que o primeiro indivíduo tenha selecionado o recurso 4. Após as atualizações necessárias, as estruturas possuem a seguinte caracterização:

0	0	0	0	1	0	0	0

Tabela 35 – Instalação com oito recursos e recurso 4 ocupado.

0	2	6

Tabela 36 – Lista auxiliar atualizada com a remoção do índice 4.

Dentre os recursos disponíveis, o segundo indivíduo que chega à instalação seleciona o recurso 2. As estruturas são então atualizadas para a forma abaixo.

0	0	1	0	1	0	0	0

Tabela 37 – Instalação com recursos 2 e 4 em uso.

0	6

Tabela 38 – Lista auxiliar atualizada com a remoção do índice 2.

Ao terceiro usuário, resta a seleção entre os recursos de número 0 e 6. Ele opta pelo recurso 0, o recurso mais à esquerda, atualizando as estruturas conforme mostrado a seguir.

1	0	1	0	1	0	0	0

Tabela 39 – Instalação com recursos 0, 2 e 4 em uso.

6

Tabela 40 – Lista auxiliar atualizada com a remoção do índice 0.

O último indivíduo seleciona o recurso restante, configurando o estado final do *array* que representa a instalação e da lista de índices conforme pode ser observado.

1	0	1	0	1	0	1	0

Tabela 41 – Instalação com os recursos 0, 2, 4 e 6 ocupados.

Tabela 42 – Lista auxiliar vazia.

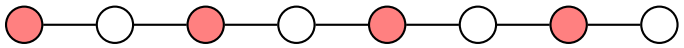

Figura 18 – Caminho P_8 que representa a instalação com oito recursos e conjunto independente máximo (recursos ocupados) em vermelho.

O processo descrito é análogo quando n é par e o conjunto independente máximo contém o vértice mais à direita de P_n.

3.4.3 Pseudocódigo

Assim como nos algoritmos das abordagens anteriores, o algoritmo da abordagem altruísta recebe como entrada um inteiro n que representa o número de recursos da instalação e uma variável booleana que decidirá se a abordagem permitirá a ocupação total da instalação. A saída do algoritmo também será o número de passos ou a cardinalidade do conjunto independente máximo de P_n encontrado. A seguir, uma proposta de pseudocódigo para a abordagem altruísta, acompanhada de uma breve descrição.

Algorithm 3 ABORDAGEM ALTRUÍSTA

```
1: função ALTRUISTA(n, ocupaTodos):
2:     instalacao ← [0, 0, ..., 0]
3:     listaIndices ← []
4:     quantPassos ← 0
5:     incomodo ← 0
6:     se ocupaTodos = Falso então
7:         se n é par então
8:             inicio ← Número inteiro entre 0 e 1
9:             listaIndices ← [inicio, inicio+2, ..., inicio+n-2]
10:        senão
11:            listaIndices ← [0, 2, ..., n − 1]
12:        fim se
13:    senão
14:        listaIndices ← [0, 1, ..., n − 1]
15:    fim se
16:    enquanto listaIndices ≠ [] faça
17:        indice ← Elemento aleatório de listaIndices
18:        instalacao[indice] ← 1
19:        Remova indice de listaIndices
20:        quantPassos ← quantPassos + 1
21:    fim enquanto
22:    i ← 0
23:    enquanto i < n − 1 faça
24:        se instalacao[i:i+2] = [1, 1] então
25:            incomodo ← incomodo + 1
26:        fim se
27:        i ← i + 1
28:    fim enquanto
29:    devolve quantPassos, incomodo
30: fim função
```

A principal diferença entre o pseudocódigo da abordagem altruísta e o da abordagem individualista reside na ausência de testes de adjacência na lista de índices quando ao selecionar um elemento, como visto na linha 17. Na abordagem individualista, os índices dos vizinhos do recurso selecionado também são removidos da lista de índices, o que não ocorre na abordagem altruísta.

Conforme visto nas abordagens anteriores, o algoritmo inicializa nas linhas 2 e 4 o *array* `instalacao` com todas as posições iguais a 0 e `quantPassos` com 0, respectivamente. Além disso, `incomodo` também é inicializado com 0 na linha 5. No entanto, na abordagem altruísta, a `listaIndices` é inicializada na linha 3 como uma lista vazia e atualizada conforme a paridade de n e se o usuário permite a ocupação integral da instalação. Esses testes são realizados nas linhas 6 e 7 do pseudocódigo. Caso a ocupação integral da instalação não seja permitida, a paridade da entrada n é avaliada a partir da linha 7.

Na linha 8, se n for par, um inteiro entre 0 e 1 é sorteado para definir qual das

duas opções de conjunto independente máximo será escolhida. Se o valor 0 for sorteado, será escolhido o conjunto que contém o vértice mais à esquerda. Caso contrário, será escolhido o conjunto que contém o vértice mais à direita (linha 9). Na linha 10, se n for ímpar, `listaIndices` simplesmente recebe a lista $[0, 2, \ldots, n-1]$, que representa o único conjunto independente máximo possível.

Se `ocupaTodos` recebe valor verdadeiro, `listaIndices` é inicializada como $[0, 1, \ldots, n-1]$ na linha 14, abrangendo todas as posições da instalação. Na linha 16, enquanto houver elementos na lista, um índice dela é sorteado na linha 17 para atribuir o valor 1 à respectiva posição no *array* `instalacao` na linha 18. Em seguida nas linhas 19 e 20, o índice sorteado é removido de `listaIndices` e o contador de passos do algoritmo é incrementado de 1. Nas linhas 22 a 28, o coeficiente global de incômodo é calculado e retornado junto com o número de passos do programa.

3.5 ABORDAGEM INDIVIDUALISTA-ALTRUÍSTA

A abordagem recebe este nome pelo fato de que um usuário, ao chegar, escolhe recursos disponíveis sem adjacências ocupadas, enquanto mantém a maior quantidade possível de recursos também sem adjacências ocupadas para os próximos usuários.

Conceitos como independência e dominação em grafos não são relevantes para esta abordagem. Portanto, esta abordagem não busca obter um subconjunto específico de um caminho P_n que represente a instalação. Seu objetivo é obter uma configuração da instalação anule seu coeficiente global de incômodo.

3.5.1 Execução

O objetivo central desta abordagem é permitir que um indivíduo escolha um recurso disponível sem adjacências ocupadas, de modo a maximizar a quantidade de recursos disponíveis restantes, também sem adjacências ocupadas, para os próximos usuários.

Considere uma instalação com 10 recursos, representada pelo *array* a seguir.

0	0	0	0	0	0	0	0	0	0

Tabela 43 – Instalação com 10 recursos inicialmente vazia.

Considere também a lista auxiliar dos índices de recursos a serem selecionados, conforme pode ser visto abaixo.

0	1	2	3	4	5	6	7	8	9

Tabela 44 – Lista de auxiliar de índices inicializada.

Inicialmente, todos os recursos são considerados *bons*, ou seja, disponíveis e sem adjacências ocupadas. Suponha que o primeiro usuário esteja pensando em utilizar o recurso

2, conforme representado na figura a seguir, embora ainda não o tenha realmente selecionado. Recursos livres, mas com alguma adjacência ocupada ou recursos *ruins* estão representados por X.

0	X	1	X	0	0	0	0	0	0

Tabela 45 – A instalação se o recurso 2 fosse o único ocupado.

Nesta configuração hipotética, percebe-se um decremento de 3 no número de recursos bons. Todavia, esta escolha não maximiza o número de recursos bons disponíveis para os próximos usuários, nem permite ocupar o recurso com o menor número possível de adjacências que podem ser ocupadas. Vale ressaltar que, para os recursos livres com duas adjacências também livres, o decremento seria o mesmo.

Como consequência, os dois primeiros usuários optam pelos recursos localizados nas extremidades à esquerda e à direita da instalação, como ilustrado nas atualizações a seguir.

1	X	0	0	0	0	0	0	0	0

Tabela 46 – Instalação com o recurso 0 em uso.

1	2	3	4	5	6	7	8	9

Tabela 47 – Lista de índices atualizada após a remoção do elemento 0.

1	X	0	0	0	0	0	0	X	1

Tabela 48 – Instalação com os recursos 0 e 9 em uso.

1	2	3	4	5	6	7	8

Tabela 49 – Lista de índices atualizada após a remoção do elemento 9.

Após a ocupação do recurso 0, o segundo usuário poderia optar pelo recurso 2, o que implicaria na redução em 2 do número de recursos bons. No entanto, ao escolher o recurso mais à direita, esse usuário terá no máximo uma adjacência ocupada, em vez de duas.

Para um terceiro indivíduo que chega à instalação, ele deverá escolher um recurso bom, se ainda existir. Observa-se que a escolha de um recurso no centro de uma sequência $[0, 0, 0]$ implica na diminuição em 3 do número de recursos bons. Logo, escolher um recurso bom no meio da instalação é ineficiente, pois reduz o número de recursos bons em 3.

Dessa forma, o terceiro indivíduo deve escolher um recurso imediatamente ao lado de um recurso ruim, com redução do número de recursos bons em apenas 2. Assuma que este indivíduo tenha selecionado o recurso 2, atualizando a instalação para a forma abaixo.

1	X	1	X	0	0	0	0	X	1

Tabela 50 – Instalação com os recursos 0, 2 e 9 em uso.

1	3	4	5	6	7	8

Tabela 51 – Lista de índices atualizada após a remoção do elemento 2.

Suponha que o próximo cliente tenha escolhido o recurso 7, resultando nas configurações a seguir.

1	X	1	X	0	0	X	1	X	1

Tabela 52 – Instalação com os recursos 0, 2, 7 e 9 em uso.

1	3	4	5	6	8

Tabela 53 – Lista de índices atualizada após a remoção do elemento 7.

Seguindo a mesma lógica, ao selecionar o recurso 4, até que metade da instalação esteja ocupada, o estado da instalação e da lista de índices serão os seguintes:

1	X	1	X	1	X	X	1	X	1

Tabela 54 – Instalação com os recursos 0, 2, 4, 7 e 9 em uso.

1	3	5	6	8

Tabela 55 – Lista de índices atualizada após a remoção do elemento 4.

Nesse ponto, todos os recursos disponíveis são considerados ruins. Apesar disso, se existir uma sequência $[X, X]$ no *array* da instalação, qualquer dos recursos representados por esta sequência podem ser escolhidos pelo próximo cliente, visto que, ainda que temporariamente, o cliente tenha apenas um recurso ocupado ao lado. Supondo que o cliente escolha o recurso 6, o estado atualizado da instalação e da lista de índices serão:

1	X	1	X	1	X	1	1	X	1

Tabela 56 – Instalação com os recursos 0, 2, 4, 6, 7 e 9 em uso.

1	3	5	8

Tabela 57 – Lista de índices atualizada após a remoção do elemento 6.

A partir desta etapa, a escolha dos recursos pode ser feita aleatoriamente entre os índices representados na lista auxiliar, caso tenha sido permitida à abordagem a ocupação integral da instalação, pois todos os recursos livres têm ambas as adjacências ocupadas.

Devido ao grande número de linhas de código requeridas para implementar a abordagem individualista-altruísta, não é viável apresentar a proposta completa de pseudocódigo nesta seção. O código escrito em *Python* que implementa a abordagem pode ser visto no método `individualist_altruistic()` da classe `Approaches()`. O leitor pode acessar o código no seguinte link: https://github.com/robertoleonie/OccupationSimulator/blob/main/approaches.py.

3.6 ABORDAGEM DO FAXINEIRO

Esta abordagem recebe este nome devido à presença de um responsável pela limpeza ou faxineiro na instalação. Este profissional deseja limpar o menor número possível de recursos, assegurando que os usuários da instalação não sejam incomodados pela ocupação de recursos vizinhos.

Assim, através de um caminho P_n que representa a instalação, esta abordagem possui como objetivo a obtenção do conjunto dominante independente mínimo. É importante dizer que a abordagem do faxineiro é a única a proibir a ocupação total dos recursos, já que o objetivo do faxineiro é minimizar o esforço necessário na limpeza da instalação.

3.6.1 Conceituação

Primeiramente, é necessário determinar o conjunto dominante independente mínimo do caminho P_n. Seja este conjunto D. Para $n = 1$, é notório que o subgrafo induzido por D será exatamente P_1.

Para $n > 1$, com o objetivo de minimizar o conjunto dominante independente encontrado em P_n, uma estratégia eficaz consiste em alternar por entre seus vértices exatamente como na abordagem altruísta, mantendo sua propriedade de independência. Isso se deve ao fato de que o conjunto dominante independente de um grafo é também um conjunto independente maximal, como discutido anteriormente.

Portanto, como na abordagem anterior, para n par, o conjunto dominante independente mínimo D apresenta duas possibilidades, conforme visto a seguir.

Dado que o conjunto dominante independente é mínimo, é desejável minimizar a adição de vértices a ele. Para n ímpar, considere P_n como um caminho composto por um número par de vértices, com um vértice adicionado a uma de suas extremidades. Para simplificar,

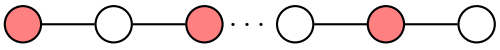

Figura 19 – Em vermelho, uma opção de conjunto dominante independente mínimo do caminho $P_{2k}, k = 0, 1, 2, \ldots$.

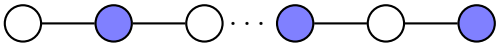

Figura 20 – Em azul, outra opção de conjunto dominante independente mínimo do caminho $P_{2k}, k = 0, 1, 2, \ldots$.

assuma essa extremidade como sendo a mais à direita. Assim, o vértice adicionado se torna o vértice mais à direita de P_n.

Visto que já foram exibidas duas possibilidades para D no grafo P_{2k}, existem duas hipóteses para o conjunto dominante independente mínimo de $P_{2k+1}, k = 0, 1, 2, \ldots$, utilizando a estratégia de alternância entre os vértices do caminho. As hipóteses podem ser vistas a seguir.

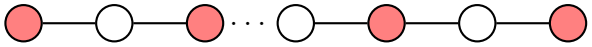

Figura 21 – Em vermelho, uma hipótese de conjunto dominante independente mínimo $D(H_1)$ no caminho $P_{2k+1}, k = 0, 1, 2, \ldots$.

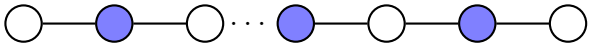

Figura 22 – Em azul, outra hipótese de conjunto dominante independente mínimo $D(H_2)$ no caminho $P_{2k+1}, k = 0, 1, 2, \ldots$.

A união dos conjuntos de vértices das duas hipóteses é igual a $V(P_n)$. Além disso, $D(H_2)$ possui exatamente um elemento a menos do que $D(H_1)$. Logo, $D = D(H_2)$, consolidando-se assim como o único conjunto dominante independente mínimo do caminho $P_{2k+1}, k = 0, 1, 2, \ldots$. Como resultado, o conjunto D não inclui nenhum dos vértices mais à esquerda e à direita de P_{2k+1}.

3.6.2 Execução

Para obter a representação do conjunto dominante independente mínimo em um *array* que representa a instalação, os índices a serem selecionados serão restritos aos índices dos vértices desse conjunto, em conformidade com as abordagens anteriores. Esses índices serão selecionados através de uma lista auxiliar.

A paridade da capacidade da instalação determinará os índices disponíveis para seleção. Se n for par, a lista auxiliar de índices terá duas opções de escolha no início da rotina da abordagem. Essas opções estão ilustradas abaixo.

0	2	4	6	...	$n-4$	$n-2$

Tabela 58 – Primeira opção de lista auxiliar de índices a serem escolhidos.

1	3	5	7	...	$n-3$	$n-1$

Tabela 59 – Segunda opção de lista auxiliar de índices a serem escolhidos.

Até aqui, para n par, a lista auxiliar de índices possui inicialização idêntica à vista na abordagem altruísta. Todavia, a inicialização da lista difere quando n é ímpar. Os índices que eram proibidos de serem sorteados na abordagem altruísta são agora os únicos disponíveis. Segue uma comparação da inicialização da lista auxiliar de índices nas duas abordagens:

0	2	4	6	...	$n-3$	$n-1$

Tabela 60 – Estado inicial da lista auxiliar de índices na abordagem altruísta quando n é ímpar.

1	3	5	7	...	$n-4$	$n-2$

Tabela 61 – Estado inicial da lista auxiliar de índices na abordagem do faxineiro quando n é ímpar.

Os demais passos permanecem inalterados: os elementos selecionados são removidos da lista de índices, o *array* que representa a instalação é atualizado na posição correspondente e o número de passos a ser retornado ao término da execução é incrementado de 1.

Inicialmente, assuma que a capacidade da instalação seja de 6. As estruturas são inicializadas da seguinte forma:

0	0	0	0	0	0

Tabela 62 – Instalação inicialmente vazia, representada por um *array*.

0	2	4

Tabela 63 – Estado inicial da lista auxiliar de índices.

Supondo que o primeiro usuário tenha selecionado o recurso 2, as estruturas são atualizadas para a forma abaixo.

0	0	1	0	0	0

Tabela 64 – Instalação com o recurso 2 sendo utilizado.

0	4

Tabela 65 – Lista auxiliar de índices atualizada com a remoção do elemento 2.

Admita que o próximo usuário seleciona o recurso número 4, atualizando as estruturas conforme a seguir.

0	0	1	0	1	0

Tabela 66 – Instalação com os recursos 2 e 4 em uso.

0

Tabela 67 – Lista auxiliar de índices atualizada com a remoção do elemento 4.

Restando ao último usuário utilizar o recurso 0, as estruturas terão seus estados finais conforme mostrado abaixo.

1	0	1	0	1	0

Tabela 68 – Estado final da instalação, com os recursos 0, 2 e 4 ocupados.

Tabela 69 – Lista auxiliar de índices vazia.

O processo é análogo se o recurso mais à direita da instalação tiver sido liberado pelo faxineiro para utilização, em vez do recurso mais à esquerda.

Considere agora $n = 7$. É importante lembrar que, para n ímpar, os dois recursos mais à esquerda e mais à direita são bloqueados para utilização, pois o conjunto dominante independente correspondente é mínimo, caracterizando a inicialização das estruturas como pode ser visto a seguir.

0	0	0	0	0	0	0

Tabela 70 – Instalação com sete recursos inicialmente vazia, representada por um *array*.

1	3	5

Tabela 71 – Lista auxiliar de índices no seu estado inicial.

Suponha que o primeiro indivíduo que entra na instalação tenha selecionado o recurso 5, atualizando as estruturas conforme mostrado abaixo.

0	0	0	0	0	1	0

Tabela 72 – Instalação com o recurso 5 em uso.

1	3

Tabela 73 – Lista de índices atualizada com a remoção do elemento 5.

Admitindo que os próximos recursos selecionados foram os de índices 1 e 3, nessa ordem, as estruturas são atualizadas até o término da execução, como ilustrado nos estados a seguir:

0	1	0	0	0	1	0

Tabela 74 – Instalação com os recursos 1 e 5 em uso.

3

Tabela 75 – Lista de índices atualizada com a remoção do elemento 1.

0	1	0	1	0	1	0

Tabela 76 – Instalação com os recursos 1, 3 e 5 ocupados.

Tabela 77 – Lista auxiliar de índices vazia.

3.6.3 Pseudocódigo

Uma proposta de pseudocódigo para a abordagem do faxineiro é apresentada a seguir. O pseudocódigo retorna o número de passos ou a cardinalidade do conjunto dominante independente mínimo encontrado no caminho P_n, além do coeficiente global de incômodo da instalação.

Algorithm 4 ABORDAGEM DO FAXINEIRO

```
1: função FAXINEIRO(n, ocupaTodos):
2:     instalacao ← [0, 0, ..., 0]
3:     listaIndices ← []
4:     quantPassos ← 0
5:     incomodo ← 0
6:     se n é par então
7:         inicio ← Número inteiro entre 0 e 1
8:         listaIndices ← [inicio, inicio+2, ..., inicio+n-2]
9:     senão
10:        se n = 1 então
11:            listaIndices ← [0]
12:        senão
13:            listaIndices ← [1, 3, ..., n − 2]
14:        fim se
15:    fim se
16:    enquanto listaIndices ≠ [] faça
17:        indice ← Elemento aleatório de listaIndices
18:        instalacao[indice] ← 1
19:        Remova indice de listaIndices
20:        quantPassos ← quantPassos + 1
21:    fim enquanto
22:    i ← 0
23:    enquanto i < n − 1 faça
24:        se instalacao[i:i+2] = [1, 1] então
25:            incomodo ← incomodo + 1
26:        fim se
27:        i ← i + 1
28:    fim enquanto
29:    devolve quantPassos, incomodo
30: fim função
```

O algoritmo inicializa entre as linhas 2 a 5 o *array* que representa a instalação e as variáveis que calculam o número de passos e o coeficiente global de incômodo, ambos com valor zero. Entretanto, na linha 3, `listaIndices` é novamente inicializada como uma lista vazia.

Na linha 6, se o tamanho da instalação n for par, um inteiro entre 0 e 1 é sorteado na linha 7 para decidir se o conjunto dominante independente mínimo de interesse incluirá o vértice mais à esquerda ou mais à direita. Caso n seja ímpar, é avaliado na linha 10 se a instalação contém somente um recurso. Caso verdadeiro, somente o índice 0 pode ser escolhido (linha 11). Caso contrário, o único conjunto dominante mínimo possível é diretamente representado na `listaIndices`, que é então atualizada na linha 13.

Na linha 16, enquanto `listaIndices` contiver elementos, o procedimento de seleção e remoção de um índice da lista nas linhas 17 a 19 é executado conforme nas abordagens anteriores, atualizando a variável `quantPassos` na linha 20. Ao término da execução, o

coeficiente global de incômodo é calculado entre as linhas 22 e 28 e retornado juntamente com `quantPassos`.

3.7 ABORDAGEM DE MINIMIZAÇÃO DO INCÔMODO

Com a introdução de uma nova estrutura auxiliar, a abordagem de minimização do incômodo possui como objetivo a obtenção de um subgrafo induzido por um subconjunto de vértices de P_n que minimize o coeficiente global de incômodo da instalação representada por P_n.

3.7.1 Execução

Nesta abordagem, a princípio, todos os recursos estão disponíveis para utilização. Assim, a lista de índices é completa, abrangendo todos os elementos de 0 a $n-1$, permitindo a seleção de qualquer índice. Além disso, essa abordagem irá incorporar novas informações, como o número de clientes que utilizarão a instalação.

Considere uma instalação com cinco recursos, que serão utilizados por quatro indivíduos. O *array* que representa a instalação e a lista auxiliar de índices serão inicializados conforme mostrado abaixo.

0	0	0	0	0

Tabela 78 – Instalação inicialmente vazia, representada por um *array* de cinco posições.

0	1	2	3	4

Tabela 79 – Estado inicial da lista auxiliar de índices.

Para minimizar o coeficiente de incômodo global, é interessante explorar a concepção de uma nova estrutura auxiliar intitulada como *mapa de adjacências*. Como o nome sugere, este mapa regula o número máximo de adjacências de cada recurso que podem ser ocupadas. Com cinco recursos na instalação, a configuração inicial do mapa de adjacências está ilustrada a seguir.

1	2	2	2	1

Tabela 80 – Mapa de adjacências inicializado para cinco recursos.

Para uma instalação com recursos dispostos de forma linear e sequencial, é evidente que as extremidades terão no máximo um recurso adjacente ocupado, enquanto as outras posições terão no máximo dois recursos adjacentes em uso.

A cada instante, é desejável maximizar a soma das adjacências disponíveis na instalação, sendo alcançado ao maximizar o somatório dos valores do mapa de adjacências.

Em outras palavras, é desejável minimizar, a cada momento, o somatório dos valores no *array* que representa a instalação. Observe que o somatório inicial dos valores do mapa de adjacências é $1 + 2 \cdot (n - 2) + 1 = 2n - 2$. Para $n = 5$, o somatório é igual a 8.

Como os clientes desejam minimizar seu próprio incômodo, os primeiros indivíduos certamente utilizarão os recursos localizados nas extremidades esquerda e direita da instalação. Assim, suponha que o primeiro indivíduo tenha escolhido o recurso mais à direita e que o segundo escolheu o recurso mais à esquerda. Após a alocação do primeiro usuário, as estruturas são atualizadas como pode ser observado abaixo.

0	0	0	0	1

Tabela 81 – Instalação com o recurso 4 em uso.

0	1	2	3

Tabela 82 – Lista de índices atualizada após a remoção do elemento 4.

1	2	2	1	1

Tabela 83 – Mapa de adjacências atualizado com a ocupação do recurso 4.

Observe que a posição imediatamente ao lado do recurso 4 é decrementada de 1 no mapa de adjacências. Após a alocação do segundo cliente, as estruturas apresentam o estado que pode visualizado a seguir.

1	0	0	0	1

Tabela 84 – Instalação com os recursos 0 e 4 em uso.

1	2	3

Tabela 85 – Lista de índices atualizada após a remoção do elemento 0.

1	1	2	1	1

Tabela 86 – Mapa de adjacências atualizado após a ocupação do recurso 0.

Assim como a abordagem individualista-altruísta, a abordagem de minimização do incômodo implica na ocupação dos recursos mais à esquerda e à direita nos primeiros instantes de chegada dos usuários. A utilização do recurso 0 resultou no decremento de 1 na posição adjacente à sua no mapa de adjacências. Como apenas o recurso 2 possui

duas adjacências livres, e portanto, um coeficiente individual de incômodo nulo, o terceiro indivíduo deve utilizar este recurso, atualizando as estruturas como ilustrado em seguida.

1	0	1	0	1

Tabela 87 – Instalação com os recursos 0, 2 e 4 em utilização.

1	3

Tabela 88 – Lista de índices atualizada após a remoção do elemento 2.

1	0	2	0	1

Tabela 89 – Mapa de adjacências atualizado com a ocupação do recurso 2.

Os vizinhos do recurso 2 tiveram suas posições correspondentes no mapa de adjacências decrementadas de 1. Embora a posição 2 tenha o maior valor possível do mapa de adjacências, esta corresponde a um recurso indisponível, restringindo as próximas escolhas aos índices 1 e 3. Como ambos os recursos apresentam o mesmo número de adjacências livres, a escolha entre eles é indiferente. Admitindo que o próximo usuário tenha selecionado o recurso 3, as estruturas possuirão os estados abaixo.

1	0	1	1	1

Tabela 90 – Instalação com cinco recursos, todos ocupados, exceto o recurso 1.

1

Tabela 91 – Lista de índices atualizada após a remoção do elemento 3.

1	0	1	0	0

Tabela 92 – Mapa de adjacências atualizado após o recurso 3 ficar indisponível.

Como o número previsto de usuários da instalação foi atingido, a abordagem não admitirá novas chegadas, chegando assim ao fim com estados finais iguais aos ilustrados acima.

Apesar desta abordagem não propor um conjunto característico de vértices em um grafo caminho que represente uma instalação, ela permite obter um simulador de ocupação de recursos próximo o suficiente dos cenários encontrados no mundo real.

Para um pseudocódigo proposto para a abordagem, o número de passos produzido na saída corresponde ao número de indivíduos que utilizam a instalação. O simulador obtido

possui como critério de parada o momento em que esse número é alcançado. Devido à extensão deste algoritmo, a apresentação de seu pseudocódigo nesta seção foi considerada inviável. O código em *Python* que implementa a abordagem de minimização do incômodo está disponível no método `discomfort_minimization()` da classe `Approaches()`. O código pode ser acessado em: https://github.com/robertoleonie/OccupationSimulator/blob/main/approaches.py.

3.8 RESUMO DE ABORDAGENS E OBJETIVOS

Para facilitar a compreensão do leitor, o resumo abaixo descreve os objetivos de cada abordagem.

- **abordagem completamente aleatória:** permite a seleção aleatória de um recurso por um usuário;
- **abordagem individualista:** obter o conjunto dominante independente minimal de P_n;
- **abordagem altruísta:** obter o conjunto independente máximo de P_n;
- **abordagem individualista-altruísta:** obter uma configuração da instalação anule seu coeficiente global de incômodo;
- **abordagem do faxineiro:** obter o conjunto dominante independente mínimo de P_n;
- **abordagem de minimização do incômodo:** obter um subgrafo de P_n que minimize seu coeficiente global de incômodo.

4 RESULTADOS COMPUTACIONAIS

Neste capítulo, são apresentadas análises computacionais sobre alguns aspectos de cada abordagem, além do coeficiente global de incômodo por instante de chegada em cada abordagem.

4.1 ANÁLISES GERADAS

A fim de analisar as particularidades de cada abordagem, foi desenvolvido um *benchmark* que possibilita avaliar aspectos como coeficientes de incômodo por instantes de chegada de usuários e os instantes em que algum usuário se sente incomodado ao utilizar um recurso.

É importante mencionar que a geração das análises comparativas das abordagens se mostrou computacionalmente muito mais custosa do que a simulação da ocupação dos recursos de uma instalação em cada uma das abordagens. Os testes foram conduzidos em uma máquina com as seguintes especificações:

- **processador:** Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz;
- **RAM instalada:** 16.0 GB (utilizável: 15.8 GB);
- **tipo de sistema:** sistema operacional de 64 bits, processador baseado em x64;
- **sistema operacional:** Windows 11 Enterprise Versão 23H2;
- **compilação do SO:** 22631.3737;
- **experiência:** Windows Feature Experience Pack 1000.22700.1009.0.

4.1.1 Comparação de incômodos para o preenchimento total dos recursos

Considerando a ocupação integral da instalação para a comparação dos coeficientes de incômodo globais nos instantes de chegada dos usuários, é importante destacar que a abordagem do faxineiro, que deseja limpar a menor quantidade de recursos, não será incluída na comparação. Além disso, devido ao interesse em observar o comportamento de cada abordagem ao longo do tempo, será realizada apenas uma rodada para cada uma delas. Os dados completos sobre os incômodos ao longo do tempo estão disponíveis em https://github.com/robertoleonie/OccupationSimulator/blob/main/discomforts.json. Para facilitar a interpretação, a tabela abaixo indica os instantes em que os coeficientes globais de incômodo de cada abordagem deixam de ser nulos.

n	**Abordagens**				
	compl. aleatória	individualista	altruísta	ind.-alt.	otim. do incômodo
2	1	1	1	1	1
3	1	1	1	2	2
5	3	2	2	3	3
10	3	3	1	5	4
20	1	5	3	10	9
30	8	8	3	15	13
50	9	5	3	25	22
100	9	11	4	50	42
200	9	16	15	100	85
300	4	11	16	150	129
500	8	28	25	250	214
1000	23	23	24	500	440

Tabela 93 – Instantes em que algum usuário fica incomodado.

Observa-se que a abordagem individualista-altruísta sempre proporciona o maior período sem usuários incomodados, sendo seguida pela abordagem de minimização do incômodo. As demais abordagens apresentaram resultados relativamente insatisfatórios.

Para $n = 1000$, as três primeiras abordagens da tabela apresentam períodos de ausência de incômodos muito curtos. Dessa forma, conclui-se que a abordagem individualista-altruísta possui o maior período sem usuários incomodados, oferecendo um período com ausência de incômodos igual a $\lceil n/2 \rceil$.

4.1.2 Coeficientes de incômodo por usuário

Nesta subseção, é comparado o coeficiente de incômodo por usuário, que é obtido dividindo o coeficiente de incômodo global pelo número de usuários em um determinado instante. Para efeitos de visualização gráfica neste texto, os valores selecionados para comparação foram $n = \{2, 3, 5, 10\}$. Os coeficientes de incômodo por usuário são apresentados nas tabelas a seguir, onde cada tabela lista os coeficientes correspondentes a cada instante. Os dados completos podem ser acessados em https://github.com/robertoleonie/OccupationSimulator/blob/main/discomfort_per_time.json.

Abordagens	**Coeficientes**
completamente aleatória	$\{0, 2\}$
individualista	$\{0, 2\}$
altruísta	$\{0, 2\}$
ind.-alt.	$\{0, 2\}$
min. do incômodo	$\{0, 2\}$

Tabela 94 – Coeficientes por usuário para $n = 2$.

Abordagens	**Coeficientes**
completamente aleatória	$\{0, 2, 2\}$
individualista	$\{0, 2, 2\}$
altruísta	$\{0, 2, 2\}$
ind.-alt.	$\{0, 0, 2\}$
min. do incômodo	$\{0, 0, 2\}$

Tabela 95 – Coeficientes por usuário para n = 3.

Abordagens	**Coeficientes**
completamente aleatória	$\{0, 0, 1, 1.333, 2\}$
individualista	$\{0, 0, 1, 1.333, 2\}$
altruísta	$\{0, 0, 0, 1.333, 2\}$
ind.-alt.	$\{0, 0, 0, 1.333, 2\}$
min. do incômodo	$\{0, 0, 0, 1.333, 2\}$

Tabela 96 – Coeficientes por usuário para n = 5.

Abordagens	**Coeficientes**
completamente aleatória	$\{0, 0, 0, 0.666, 1.5, 1.2, 1.333, 1.429, 1.75, 2\}$
individualista	$\{0, 0, 1, 1.333, 1.5, 2, 1.667, 1.714, 1.75, 2\}$
altruísta	$\{0, 0, 0, 0, 1, 1.2, 1.333, 1.429, 1.75, 2\}$
ind.-alt.	$\{0, 0, 0, 0, 0, 0.4, 1, 1.429, 1.75, 2\}$
min. do incômodo	$\{0, 0, 0, 0, 0, 0.4, 1, 1.429, 1.75, 2\}$

Tabela 97 – Coeficientes por usuário para n = 10.

Como observado, as abordagens individualista-altruísta e de minimização do incômodo se sobressaíram em relação às demais, apresentando coeficientes de incômodo por usuário mais baixos e período sem usuários incomodados mais prolongado.

Portanto, se o objetivo é modelar uma solução centralizada para a ocupação dos recursos, as abordagens individualista-altruísta e de minimização do incômodo se apresentam como as mais vantajosas.

5 CONCLUSÃO E TRABALHOS FUTUROS

Utilizando conceitos de teoria dos grafos e otimização, este estudo investigou cenários do cotidiano ao apresentar exemplos simples de alocação de recursos e ocupação de espaços compartilhados entre indivíduos. Esses exemplos iniciais possibilitam um vasto campo de casos similares a ser explorado.

O trabalho delineou os princípios de conjunto independente e conjunto dominante de um grafo, fundamentais para a formalização e implementação direta de abordagens. Essas abordagens estabelecem heurísticas a serem otimizadas através da busca por tais conjuntos em estruturas de dados que representam uma instalação, especialmente nos grafos caminho, possibilitando uma ocupação otimizada dos recursos.

Para avaliar a eficiência de cada abordagem, foram conduzidos testes e análises comparativas, como a medição do número de usuários incomodados na instalação a cada instante de chegada, ao longo de sucessivas execuções. Esses testes incluíram a medição do Identificar as abordagens mais eficientes e viáveis é crucial para determinar qual delas é a mais adequada para o desenvolvimento de um simulador de ocupação dos recursos de uma instalação. Os códigos implementados para cada abordagem analisada e para as comparações de desempenho estão disponíveis em https://github.com/robertoleonie/OccupationSimulator.

No futuro, planeja-se aprofundar este estudo para cenários em que a chegada dos indivíduos a uma instalação siga um processo *Poisson* e o tempo de serviço de cada usuário seja exponencial, permitindo a modelagem do sistema como uma fila $M/M/k$ com restrições de uso, como a de evitar a ocupação de dois servidores consecutivos quando possível. Vale ressaltar que, nesse cenário, os clientes terão um tempo de utilização dos recursos, caracterizando o uso destes recursos dinâmico em vez de estático.

Além disso, pretende-se explorar cenários em dimensões maiores, como salas de cinema bidimensionais (2D). Nesse contexto, o desconforto de um espectador não é apenas influenciado pelas posições lateralmente adjacentes ocupadas, mas também pela presença de um espectador na fileira à frente. Essa situação pode impactar negativamente sua experiência durante a sessão, pois obstruiria sua visão do filme, tornando o problema mais realista.

Portanto, o escopo do trabalho se concentraria na investigação de salas de cinema, com o foco na pesquisa de grafos grade em vez de grafos caminho.

REFERÊNCIAS

BALLARD-MYER, J. C. Deterministic greedy algorithm for maximum independent set problem in graph theory. p. 14, 2019.

HAYNES, T. W.; HEDETNIEMI, S. T.; SLATER, P. J. **Fundamentals of domination in graphs**. New York: Marcel Dekker, Inc., 1998. 47 p.

KITAEV, S. Independent sets on path-schemes. **Journal of Integer Sequences**, Reykjavík University, Ofanleiti 2, IS-103 Reykjavík, Iceland, v. 9, 2006. Disponível em: https://cs.uwaterloo.ca/journals/JIS/VOL9/Kitaev/kitaev45.pdf. Acesso em: 19 jun.2024.

KOSTER, M. On relaxation of dominant sets. **Cornell University**, 2022.

LIU, C.-H.; POON, S.-H.; LIN, J.-Y. Independent dominating set problem revisited. **Theoretical Computer Science**, 2014.

LOPEZ, R. Independent dominating sets in unicyclic graphs. **Montclair State University**, 2022.

LU, X. Online optimization problems. **Massachusetts Institute of Technology**, 2013.

MACAMBIRA, A. F. U. et al. **Tópicos em Otimização Inteira**. [S.l.]: Editora UFRJ, 2022. 258 p.

SHUKLA, S.; THAKUR, V. Domination and it's type in graph theory. **Journal of Emerging Technologies and Innovative Research (JETIR)**, Raman University Kargi Road Kota, Bilaspur (C.G.), India., 2020. Disponível em: https://www.jetir.org/papers/JETIR2003225.pdf. Acesso em: 19 jun.2024.

SNYMAN, J. A.; WILKE, D. N. **Practical Mathematical Optimization**. [S.l.]: Springer International Publishing AG, 2018. 388 p.

WAGNER, M. R. Online optimization in routing and scheduling. **MASSACHUSETTS INSTITUTE OF TECHNOLOGY**, 2006.