Ronan Pardo Soares

Procura em Grafos

Fortaleza

2010

Ronan Pardo Soares

Procura em Grafos

Dissertação de mestrado apresentada como requisito para obtenção do título de mestre em Ciência da Computação pela Universidade Federal do Ceará.

Orientador:
Cláudia Linhares Sales

Universidade Federal do Ceará
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação
ParGO - Paralelismo, Grafos e Otimização

Fortaleza

2010

Profa. Dra. Cláudia Linhares Sales
Orientador

Prof. Dr. Manoel Campêlo
Universidade Federal do Ceará

Prof. Dr. Rudini Sampaio
Universidade Federal do Ceará

Dr. Nicolas Nisse
INRIA Sophia-Antipolis

Resumo

Neste trabalho, o problema da Procura em Grafos e o problema da Limpeza de Vizinhança são abordados, considerando, não apenas o número de agentes, mas também o número de passos que a estratégia construída necessita para resolver tais problemas. É demonstrado, neste trabalho, que o problema da Limpeza de Vizinhança, quando é levado em consideração o número de passos de uma estratégia de limpeza, é *NP-completo*. Um algoritmo para resolver uma variação da Limpeza de Vizinhança na qual os agentes não podem se movimentar é apresentado. Teoremas e algoritmos para grafos direcionados sem circuitos e caminhos disjuntos também são apresentados neste trabalho. Na Procura em Vértices, um caso particular da Procura em Grafos, é apresentado um teorema relacionando o número de agentes necessários para limpar um grafo e um subgrafo. Modelos matemáticos de programação linear inteira foram desenvolvidos para a Procura em Vértices e para a Limpeza de Vizinhança.

PALAVRAS-CHAVE: Procura em Grafos, procura em vértices, procura em arestas, procura mista, limpeza de vizinhança, busca em grafos.

Abstract

In this work, we consider the problem of Graph Seaching and the problem of calculating the Process Number where we do not only want to minimize the number of agents necessary to clear the graph, but the number of steps that we need to clear the entire graph. We demonstrate that the problem of Process Number with the number of steps as an added constraint is still *NP-complete*. We also give an algorithm to solve the variation of Process Number where the agents cannot move. For the Vertex Search problem, a particular case of Graph Searching, we present a theorem relating the number of agents necessary to clear a graph and the number of agents needed to clear a subgraph. We also give mathematical models of integer linear programming for Vertex Search and for Process Number.

KEYWORDS: Graph Seaching, vertex search, edge search, mixed search, process number.

Agradecimentos

Agradeço enormemente à minha orientadora, professora Cláudia Linhares Sales, por sua dedicação, paciência e sabedoria. Sua presença foi fundamental para a realização deste trabalho. Seus conselhos, não somente relacionados a esse trabalho, me ajudaram em vários aspectos da vida. Espero seguir seu exemplo como professora, pesquisadora e orientadora.

Agradeço também ao David Coudert, ao Nicolas Nisse e ao Dorian Mazauric pela imensa ajuda neste trabalho. Eles me auxiliaram a entender as ideias que originaram os teoremas mais complexos, também facilitando a busca por artigos para compor uma boa referência bibliográfica.

Agradeço à minha família, especialmente aos meus pais, Alberto Melo Soares e Miriam Carmen Pardo Soares, pelo apoio e incentivo aos estudos, por me fornecerem um excelente ambiente de estudo em casa, por serem um exemplo de perseverança, por estarem sempre ao meu lado me apoiando e aconselhando.

Agradeço à Tássia Gabrielle Ponte Carneiro, minha namorada, por toda dedicação, carinho e paciência, por sempre me incentivar e apoiar na realização dos meus sonhos, por me ajudar durante a elaboração do texto e, finalmente, por todos os momentos felizes que já passamos.

Diversas outras pessoas foram importantes para a concretização deste sonho. Não posso citar todas, mas elas ficarão guardadas em minha memória.

Finalmente, à CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - por financiar a bolsa de estudos para a realização do presente trabalho.

Sumário

1 Introdução p. 8

2 Definições p. 11

2.1 Conceitos Básicos . . . . . p. 11

2.2 Classes de Grafos . . . . . p. 14

2.3 Decomposições . . . . . p. 17

2.4 Outros conceitos . . . . . p. 19

3 Elementos da Procura em Grafos p. 21

3.1 Procura em Vértices . . . . . p. 21

3.2 Procura em Arestas . . . . . p. 24

3.3 Procura Mista . . . . . p. 25

3.4 Terminologia . . . . . p. 26

4 Resultados Clássicos p. 29

4.1 Complexidade . . . . . p. 29

4.2 Procura em Vértices . . . . . p. 30

4.3 Procura em Arestas . . . . . p. 31

4.4 Procura Mista . . . . . p. 33

4.5 Procura em Grafos Direcionados . . . . . p. 35

5 Limpeza de Vizinhança p. 36

6 Procuras Rápidas p. 40

6.1 Procura em Grafos Rápida . . . . . p. 40
6.1.1 Procura em Arestas . . . . . p. 41
6.1.2 Procura em Vértices . . . . . p. 43
6.2 Limpeza de Vizinhança . . . . . p. 46

7 Limpeza de Vizinhança Rápida p. 48
7.1 Complexidade . . . . . p. 48
7.2 Limpeza de Vizinhança com Agentes Fixos . . . . . p. 50
7.3 Cobertura e Limpeza de Vizinhança . . . . . p. 53
7.4 Resultados para Classes de Grafos . . . . . p. 53
7.4.1 Grafos direcionados sem circuitos (DAGs) . . . . . p. 53
7.4.2 Caminhos Direcionados Disjuntos . . . . . p. 57
7.4.3 Subdivisão de Estrela Direcionada . . . . . p. 61
7.4.4 Árvores Direcionadas . . . . . p. 63
7.5 Modelos Lineares Inteiros . . . . . p. 64

8 Procura em Vértices Rápida p. 67
8.1 Resultados Gerais . . . . . p. 67
8.2 Caminhos . . . . . p. 68
8.3 Modelos Lineares Inteiros . . . . . p. 69

9 Conclusão p. 72

Lista de Figuras p. 73

Lista de Tabelas p. 76

Índice Remissivo p. 77

Referências p. 79

1 Introdução

Procura em Grafos é um problema bastante estudado em Teoria dos Grafos [1]. A primeira formulação matemática desse problema foi feita por Torrence Parsons em 1976 [2], inspirado por um artigo de Breish [3], o qual retratava o problema de encontrar um explorador perdido em um complicado sistema de cavernas escuras. Para resolver tal problema, deve-se descobrir uma estratégia de procura utilizando o menor número de pessoas e garantindo que o explorador será resgatado independentemente das suas ações, ou seja, a procura deve ter sucesso, mesmo que o explorador perdido tente evitar o resgate. Por esse motivo, é comum chamar o explorador perdido de fugitivo ou ladrão, e as pessoas encarregadas da procura de policiais ou agentes.

Uma outra aplicação para Procura em Grafos é o problema de descontaminação. Neste caso, tem-se um complexo sistema de tubulações contaminado por algum gás tóxico (arma química ou biológica) e pretende-se descontaminá-lo garantindo que, ao final, todas as tubulações estarão livres do gás. Pode-se modelar o sistema de tubulações como um grafo, no qual cada junção de dois ou mais tubos é um vértice do grafo, e dois vértices u e v são adjacentes quando ao passar da junção u para a junção v, não se atravessa nenhuma outra junção. Se existir uma estratégia de procura utilizando k policiais para esse grafo, pode-se, então, descontaminar a tubulação utilizando k agentes.

A Procura em Grafos também pode modelar o problema de encontrar um carro roubado em uma cidade utilizando o menor número de policiais. Modelando as ruas da cidade como um grafo, cada vértice representaria um cruzamento; e as arestas, as ruas entre esses cruzamentos. Se há uma estratégia de procura que utilize k policiais para o problema de Procura em Grafos nesse grafo, então pode-se utilizar a mesma estratégia de procura com k policiais para procurar o carro roubado na cidade.

Existem formulações matemáticas contínuas e discretas para esse problema, ambas equivalentes, como foi provado por Golovach [4]. Neste trabalho, é estudada a formulação discreta do problema.

Formalmente, dado um grafo $G = (V, E)$, uma *estratégia de procura* em G é uma sequência de posições de policiais $S = \langle A_0, A_1, A_2, \ldots, A_k \rangle$, com $A_i \subseteq V$, para todo $i \in [0, k-1]$, de forma que para qualquer $i \in [0, k-1]$ uma das seguintes afirmações é verdadeira:

1. $A_i \subset A_{i+1}$, significando que policiais foram adicionados à procura;
2. $A_{i+1} \subset A_i$, significando que policiais foram removidos da procura;
3. $A_i \Delta A_{i+1} = \{u, v\}$ com $e = \{u, v\} \in E$, $u \in A_i$ e $v \in A_{i+1}$, significando que o policial atravessou a aresta (u, v) de u para v.

Cada conjunto A_i também é chamado de etapa ou de passo da procura. Em cada etapa, o conjunto das arestas do grafo pode ser particionado em dois: a partição das arestas contaminadas e a partição das arestas limpas. As arestas contaminadas são as que podem ser alcançadas pelo fugitivo. O conjunto das arestas limpas é o conjunto complementar. Inicialmente, todas as arestas do grafo estão contaminadas. Um vértice $v \in V$ é dito limpo se todas as arestas que incidem nele estão limpas ou se ele está ocupado por um policial. Deve-se atravessar uma aresta a partir de uma extremidade limpa para descontaminar essa aresta. Afirma-se que uma aresta limpa é re-contaminada, ou seja, voltou a pertencer ao conjunto de arestas contaminadas, se, em algum momento da procura, existe um caminho livre de policiais entre essa aresta e uma aresta contaminada. Ao final da estratégia de procura, não podem existir arestas contaminadas. O número de policiais utilizados em uma procura é dado por $\max_{i \in [0,k]}\{|A_i|\}$. O problema da Procura em Grafos consiste em encontrar uma estratégia de procura que utilize o menor número de policiais, esse problema é *NP-difícil* [5]. A classe de problemas *NP-difíceis* é caracterizada por problemas nos quais é possível verificar uma solução em tempo hábil, porém não se sabe se também é possível encontrar tal solução em tempo hábil. Outro fato interessante é a equivalência desse problema com outros parâmetros como a Decomposição em Caminhos e a Separação de Vértices de um grafo.

Neste trabalho, apresenta-se um estudo aprofundado sobre as inúmeras variações do problema de Procura em Grafos, mostrando os resultados e algoritmos existentes na literatura para cada um deles. Nas versões mais difundidas, o objetivo é tecer uma estratégia que minimize o número de agentes utilizados. Aqui é destacada uma outra abordagem para o problema: minimizar o número de passos de uma estratégia, dado um número fixo de agentes. Essa variação do problema é chamada de Procura em Grafos Rápida. Para a Procura em Vértices Rápida, um caso particular da Procura em Grafos Rápida, é

apresentado, nesta dissertação, um teorema relacionando o número de agentes necessários para limpar um grafo e um subgrafo.

O problema da Limpeza de Vizinhança foi introduzido por N. Jose e A.K. Somani [6]. Esse problema também é *NP-difícil* [7, 8] na sua formulação original. Nesta dissertação, mostra-se que o problema da Limpeza de Vizinhança mesmo restringindo o número de passos ou o número de agentes ainda é *NP-difícil*. Além disso, apresenta-se um algoritmo para resolver uma variação da Limpeza de Vizinhança na qual os agentes não podem se movimentar. Teoremas e algoritmos para grafos direcionados sem circuitos e caminhos disjuntos também são apresentados neste trabalho. Modelos matemáticos de programação linear inteira foram desenvolvidos para encontrar parâmetros relacionados à Procura em Vértices Rápida e à Limpeza de Vizinhança. Tais resultados foram obtidos conjuntamente com David Couldert, Dorian Mazauric e Nicolas Nisse durante um estágio feito junto a Equipe Mascotte do INRIA Sophia Antipolis.

Este texto está organizado da seguinte maneira: serão apresentadas a terminologia e as definições no Capítulo 2, seguindo no Capítulo 3 com um resumo das principais versões do problema de procura. No Capítulo 4, são exibidos os resultados mais importantes dessas versões. A Limpeza de Vizinhança é apresentada no Capítulo 5. O foco dessa dissertação é discutido com mais detalhes no Capítulo 6. No Capítulo 7, os resultados obtidos para a Limpeza de Vizinhança são apresentados e, no Capítulo 8, os resultados obtidos para a Procura em Vértices Rápida são apresentados. Finalmente, as conclusões e trabalhos futuros são apresentados no Capítulo 9.

2 Definições

Neste capítulo, são introduzidas as definições e notações de Teoria dos Grafos necessárias para a compreensão do texto. Para mais definições, é indicada a leitura de [9].

2.1 Conceitos Básicos

Um *grafo* $G = (V, E)$, não orientado, é definido por um conjunto, não vazio, V $(V(G))$ de elementos chamados *vértices*, E $(E(G))$ um conjunto de *arestas* e $\Phi : E \to (V \times V)$ uma função que atribui a cada aresta um par não ordenado de vértices de G. Para simplificar a notação, se $e \in E(G)$ é uma aresta de G e $\Phi(e) = (u, v)$ então, caso não haja ambiguidade, e será denotada por $e = (u, v)$ ou $e = uv$. Diz-se que os vértices u e v são *adjacentes* e são as *extremidades* da aresta $e = (u, v)$. Uma aresta *incide* em um vértice se ele é uma de suas extremidades. Duas arestas que compartilham um mesmo vértice são *adjacentes*. Se uma aresta possui extremidades iguais, então chama-se essa aresta de *laço*. Se duas ou mais arestas possuem as mesmas extremidades, então elas são chamadas *arestas múltiplas*. Um grafo, $G = (V, E)$, sem laços, sem arestas múltiplas e com $V(G)$ finito é chamado de *grafo simples*. Nesse caso, a função Φ pode ser omitida na descrição do grafo.

Quando o conjunto de arestas é formado por pares ordenados, diz-se que o grafo é um *grafo orientado*, *grafo direcionado* ou *digrafo* e denota-se por $\vec{e} = u \to v$ $(\vec{e} = (u, v))$ uma aresta de um vértice u para um vértice v em um grafo orientado simples. Para grafos orientados, arestas múltiplas são arestas que possuem extremidades iguais e mesmo sentido.

O *grau* de um vértice v, ou $d(v)$, é o número de arestas que incidem nesse vértice, cada laço contando duas vezes. O menor (maior) grau de um vértice no grafo G é denotado por $\delta(G)$ (respectivamente $\Delta(G)$). A *vizinhança* de um vértice v em um grafo $G = (V, E)$ é o conjunto $N_G(v) = \{u|(u, v) \in E(G)\}$. Diz-se que u é *vizinho* de v se $u \in N_G(v)$.

A *vizinhança de um conjunto* $S \subseteq V$ de um grafo $G = (V, E)$ é o conjunto $N_G(S) = \bigcup_{v \in S} N_G(v)$.

Em um grafo orientado $G = (V, E)$, o *grau de saída* de um vértice v, ou $d^+(v)$, é dado por $d^+(v) = |\{v \to u \in E\}|$. Da mesma forma, o *grau de entrada* de um vértice v, ou $d^-(v)$, é dado por $|\{u \to v \in E\}|$. De forma similar a de um grafo não orientado, a *vizinhança de saída* (*vizinhança de entrada*) de um vértice v é dada por $N_G^+(v) = \{u | v \to u \in E\}$ (respectivamente $N_G^-(v) = \{u | u \to v \in E\}$). Um grafo orientado é *simétrico* se para cada aresta $u \to v \in E$, tem-se $v \to u \in E$. O *grafo subjacente* de um grafo orientado $G = (V, E)$ é o grafo não orientado $G' = (V, E)$ de forma que $V(G') = V(G)$ e $E(G') = \{(u, v) | u \neq v \text{ e } (u \to v \in E(G) \text{ ou } v \to u \in E(G))\}$.

Um vértice u de um grafo orientado G que possua vizinhança de saída vazia é chamado de *sumidouro*. Uma *fonte* é um vértice que possui vizinhança de entrada vazia.

Sejam $G = (V, E)$ e $G' = (V', E')$ grafos. Diz-se que G' é *subgrafo* de G se $V' \subseteq V$ e $E' \subseteq E \backslash \{(u, v) \in E | u \notin V' \text{ ou } v \notin V'\}$. Nesse caso, diz-se também que G é um *supergrafo* de G'. Se G' é subgrafo de G e $V' = V$, então G' é um *subgrafo gerador* de G. Dado um subconjunto V' dos vértices do grafo G, o grafo $G[V'] = (V', E')$ é um *subgrafo induzido* (pelos vértices) de G, se $E' = \{(u, v) | u, v \in V' \text{ e } (u, v) \in E(G)\}$. Da mesma forma, define-se o subgrafo induzido (pelas arestas) de um subconjunto E' das arestas de G como $G[E'] = (V', E')$ no qual $V' = \{v | (u, v) \in E'\}$.

Sejam A e B dois conjuntos tais que $A \subseteq B$. O *complemento* de A é $\overline{A} = B \backslash A$. O *inverso* de um subgrafo G' de um grafo G é a união dos conjuntos $\overline{V(G')}$ e $\overline{E(G')}$. Nota-se que o inverso de um grafo não é, necessariamente, um grafo.

Um grafo simples $G = (V, E)$ é *completo* quando para todo $u, v \in V$, tem-se que $(u, v) \in E$. Um grafo completo com n vértices é denotado por K_n. Um grafo é *vazio* se $E(G) = \emptyset$. Uma *clique* de um grafo simples G é um subconjunto V' de V tal que $G[V']$ é completo. Denota-se por $\omega(G)$ a cardinalidade da maior clique de G. Um *conjunto estável*, ou *conjunto independente*, é um subconjunto V' de V tal que $G[V']$ é vazio.

Uma *trilha* $T = \langle v_1, e_1, v_2, e_2, \ldots, e_{p-1}, v_p \rangle$ é uma sequência alternada de vértices e de arestas distintas, $p \geq 1$ tal que $e_i = (v_i, v_{i+1})$, para todo $i \in [1, p-1]$. O *comprimento de uma trilha* é a quantidade de arestas nessa trilha. Se o grafo G é simples, qualquer trilha $T = \langle v_1, e_1, v_2, e_2, \ldots, e_{p-1}, v_p \rangle$ pode ser determinada apenas pelos seus vértices, já que existe somente uma aresta entre cada par de vértices. Dada uma trilha $T = \langle v_1, e_1, v_2, e_2, \ldots, e_{p-1}, v_p \rangle$, suas *extremidades* são o primeiro e o último elementos da

sequência, e seus *vértices internos* são os demais.

Um *caminho* é uma trilha na qual todos os vértices são distintos. A *distância* entre dois vértices é o comprimento do menor caminho entre eles. Um *ciclo* em um grafo é definido de forma similar a uma trilha, com a exceção de que as suas extremidades são iguais. Diz-se que um grafo G contém um P_n (C_n) se ele possui um caminho (ciclo) de comprimento $n-1$ (n) como subgrafo. Uma *corda* em um ciclo é uma aresta entre pares de vértices não adjacentes no ciclo. Um ciclo sem cordas de comprimento pelo menos 4 é chamado de *buraco*.

Em um grafo orientado, uma trilha $T = \langle v_1, e_1, v_2, e_2, \ldots, e_{p-1}, v_p \rangle$ (de v_1 a v_p, ou entre v_1 e v_p) é uma sequência alternada de vértices e de arestas distintas, $p \geq 1$ e $e_i = v_i \rightarrow v_{i+1}$ para todo $i \in [1, p-1]$. Um caminho, ou caminho orientado, é uma trilha na qual todos os vértices são distintos. Um *circuito* em um grafo orientado é definido de forma similar a uma trilha, com exceção de que somente as suas extremidades são iguais. Um ciclo em um grafo orientado é uma sequência de vértices e arestas $C = \langle v_1, e_1, v_2, e_2, \ldots, e_{p-1}, v_p \rangle$ no qual $v_p = v_1$ e $e_i = v_i \rightarrow v_{i+1}$ ou $e_i = v_i \leftarrow v_{i+1}$ para todo $i \in [1, p-1]$.

O *diâmetro* de um grafo $G = (V, E)$ é o comprimento do maior caminho, considerando apenas os menores caminhos entre todos os pares de vértices de G. A *cintura* de um grafo G é o tamanho do ciclo de menor comprimento de G. Caso G seja um grafo orientado, consideram-se apenas os caminhos orientados para o *diâmetro* e a *cintura* de G.

A *borda* de um subconjunto de arestas E' de um grafo $G = (V, E)$ é o conjunto de vértices de $G[E']$, os quais são extremidades de pelo menos uma aresta em E' e uma aresta em $\overline{E'}$.

Em um grafo não orientado, dois vértices, u e v, são *conectados*, $u \sim v$, se existe um caminho entre eles. Um grafo G é dito *conexo* se todos os pares de vértices são conectados. Um subgrafo conexo maximal de um grafo G é chamado de *componente* de G. Se entre cada par de vértice de G existem pelo menos k caminhos disjuntos, então diz-se que G é um grafo *k-conexo*. Um grafo G é *desconexo* se existe mais de uma componente em G. Dado um subconjunto de vértices X de G, chama-se de *X-componente* o conjunto de componentes conexas de $G[V \backslash X]$.

Dado um grafo $G = (V, E)$ e dois subconjuntos de vértices V_1 e V_2, se $G[V_1 \cup V_2]$ é conexo, então os conjuntos V_1 e V_2 se *tocam*.

Um *corte de vértices* é um subconjunto de vértices S de um grafo conexo $G = (V, E)$ tal que $G[V \backslash S]$ é desconexo. De forma similar, um *corte de arestas* é um subconjunto de

arestas E' de um grafo conexo $G = (V, E)$ tal que $G' = (V, \overline{E'})$ é desconexo.

Uma *cobertura de vértices* de um grafo G é um subconjunto, S, dos vértices de G tal que para cada aresta $e = (u, v)$ de $E(G)$, tem-se que $u \in S$ ou $v \in S$.

O *complemento* de um grafo $G = (V, E)$ é o grafo $\bar{G} = (\bar{V}, \bar{E})$ tal que $\bar{V} = V$ e $\bar{E} = \{(u, v) \in (V \times V) | u \neq v \text{ e } (u, v) \notin E\}$.

Uma *partição*, Σ, de um conjunto S é uma família de subconjuntos, também chamados de classes, $\Sigma = \{S_i \mid i \in I\}$, tal que $S_i \subseteq S$, $\bigcup S_i = S$ e $S_i \cap S_j = \emptyset$, para todos $i, j \in I$ com $ineqj$. Diz-se que um conjunto possui uma *r-partição* se $|\Sigma| = r$.

Dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são ditos *isomorfos* se existem bijeções $f : V_1 \to V_2$ e $g : E_1 \to E_2$ tais que $v \in V_1$ é incidente a $e \in E_1$ se e somente se $f(v)$ é incidente a $g(e)$. Observa-se que, se G_1 e G_2 forem simples, um isomorfismo pode ser representado apenas por uma função $f : V_1 \to V_2$ e tem-se que $(u, v) \in E_1$ se e somente se $(f(u), f(v)) \in E_2$.

Uma *coloração* de um grafo G é uma função que atribui a cada vértice de G uma cor. Se em uma coloração de um grafo G as extremidades das aresta de G recebem cores distintas, então diz-se que a coloração é *própria*. O *número cromático*, $\chi(G)$, de um grafo G é o menor inteiro k tal que há uma coloração própria de G utilizando k cores distintas.

2.2 Classes de Grafos

Um grafo $G = (V, E)$ é dito *r-partido*, ou *r-particionado*, se existe uma r-partição do conjunto de vértices, tal que toda aresta possui extremidades em classes diferentes da partição. Em particular, todo grafo 2-partido é também chamado de *bipartido*.

Uma *floresta* é um grafo acíclico. Uma *árvore* é uma floresta conexa. Os vértices de uma árvore são chamados de *nós*. As *folhas* são os nós da árvore que possuem grau 1. Os nós que não são folhas são chamados de *internos*. Um grafo direcionado sem circuitos é chamado de *DAG*.

Pode-se destacar um vértice r de uma árvore $T = (V, E)$ chamando-o de *raiz* da árvore. Nesse caso a árvore é *enraizada* em r. Dada uma árvore $T = (V, E)$ enraizada em r, os *descendentes* de um nó $v \in V$ da árvore são todos os vértices u tais que v é um nó interno do caminho de u até r na árvore. Diz-se que v é um *ancestral* de u nesse caso. Os *filhos* de um vértice v são todos os nós que dele descendem e que, a ele, são adjacentes. Se u é filho de v, então v é *pai* de u.

Uma *estrela* com n vértices é um grafo, G, tal que $V(G) = \{v_1, \ldots, v_n\}$ e $E(G) = \{(v_1, v_i) | 1 < i \leq n\}$. O vértice v_1 é chamado de centro e os demais são chamados de periféricos.

Um grafo G é dito *planar* se existe uma representação gráfica de G em um plano na qual cada vértice é representado por ponto e cada aresta é representada por um segmento de reta ligando suas duas extremidades e, nessa representação, as arestas só se intersectam nas suas extremidades. Dada uma representação gráfica de um grafo G em um plano, uma *face* é uma região do plano limitada por arestas de G. Um vértice (aresta) *incide em* uma face f se ele (ela) toca a face f. A *face externa* é a região do plano que não é limitada por arestas. Um grafo é dito *periplanar* se ele é planar, e existe uma representação gráfica no plano de forma que todos os vértices são adjacentes à face externa. O *dual fraco* de um grafo periplanar é um grafo obtido a partir de uma representação gráfica do grafo em um plano de forma que cada face dessa representação é um vértice do dual, com exceção da face externa, e dois vértices são adjacentes se as duas faces associadas tocam uma mesma aresta ou vértice da representação.

Um grafo é *unicíclico* se ele possui apenas um ciclo.

Os *grafos "split"* são todos os grafos em que é possível particionar o conjunto de vértices em duas partições sendo uma delas uma clique e a outra um conjunto independente do grafo.

Um *co-grafo* é qualquer grafo que pode ser construído a partir de aplicações sucessivas das seguintes regras:

1. K_1 é um co-grafo;
2. Se G é um co-grafo então $\overline{G}$ é um co-grafo;
3. Se G e H são co-grafos então o grafo $G' = (V(G) \cup V(H), E(G) \cup E(H))$ é um co-grafo.

Os *grafos de interseção* são grafos que podem ser obtidos a partir de uma família de conjuntos e suas interseções. Se $S = \{S_1, \ldots, S_n\}$ é uma família de conjuntos então o grafo $G = (V, E)$ tal que $V = \{v_1, \ldots, v_n\}$ e $E = \{(v_i, v_j) | S_i \cap S_j \neq \emptyset\}$ é o grafo de interseção obtido a partir da família S.

Um grafo, G, é dito de *intervalo* se cada vértice, v, pode ser associado a um intervalo $[s_v, e_v] \subset \mathbb{R}$ tal que uma aresta $(u, v) \in E(G)$ se e somente se $[s_v, e_v] \cap [s_u, e_u] \neq \emptyset$.

O grafo de interseção de uma família de segmentos de reta que conectam duas retas paralelas em um espaço euclidiano é um *grafo de permutação*.

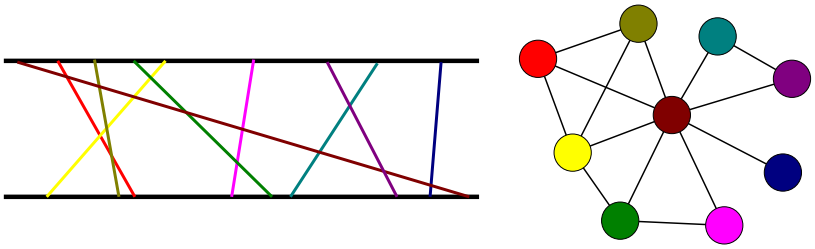


Figura 1: Exemplo de grafo de permutação. As cores representam a bijeção entre os vértices do grafo e os segmentos de reta.

Um *grafo trapezoidal* é um grafo de interseção de uma família de trapézios que se situam entre duas retas paralelas em um espaço euclidiano. Cada trapézio deve possuir, necessariamente, uma base em cada uma das retas paralelas.

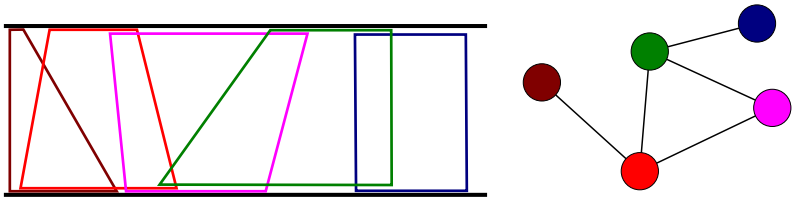


Figura 2: Exemplo de grafo trapezoidal. As cores representam a bijeção entre os vértices do grafo e os trapézios.

Um grafo é *cordal* se ele não possui buracos. Esses grafos podem ser caracterizados da seguinte forma: Um grafo G é cordal se e somente se existe uma árvore T e uma família de subgrafos de T, $S = \{T_0, \ldots, T_{V(G)}\}$, tal que existe uma bijeção entre os vértices de G e S satisfazendo a propriedade de que dois vértices de G são adjacentes se e somente se os subgrafos associados a esses vértices possuem vértices de T em comum. Sabe-se que é sempre possível construir essa árvore T de forma que exista uma bijeção entre cada vértice de T com uma clique maximal de G, e, além disso, tem-se que um subgrafo da família S, T_i, estar associado a um vértice $v_i \in V(G)$ implica em T_i conter todos os vértices de T que representam cliques maximais contendo v_i, ou seja, se T_i está associado com v_i, então para todo $u \in V(T)$, tal que a clique maximal de G associada a u contem v_i, tem-se que $u \in V(T_i)$.

Os *grafos "starlike"* são grafos cordais tais que uma das árvores T das cliques maximais é uma estrela. Seja v_0 o vértice central de T, v_i, $0 < i \leq r$, os vértices periféricos de T e X_i a clique maximal associada ao vértice v_i. Caso seja possível afirmar que o valor

$max_{i=1}^{r}\{|X_i \backslash X_0|\}$ é igual a um inteiro k, então diz-se que esse grafo é um *grafo "k-starlike"*. É interessante notar que todos os grafos split são "1-starlike".

Uma *k-árvore* é definida de forma recursiva da seguinte maneira: uma k-árvore com $k+1$ vértices consiste de uma clique com $k+1$ vértices ($k+1$-clique); dada uma k-árvore T_n com n vértices, $n \geq k+1$, constrói-se uma k-árvore com $n+1$ vértices acrescentando um novo vértice v a T_n e fazendo-o adjacente a cada vértice de uma k-clique de T_n e não adjacente aos $n-k$ vértices restantes.

2.3 Decomposições

As decomposições em árvore e em caminho de um grafo, intuitivamente, indicam quão parecido esse grafo é de uma árvore ou caminho respectivamente.

Definição 2.1. *Uma* decomposição em árvore *de um grafo $G = (V, E)$ é um par $D = (\mathcal{X}, T)$, sendo $T = (I, F)$ uma árvore e $\mathcal{X} = \{X_i, i \in I\}$ uma família de subconjuntos de V, um para cada nó de T, tais que:*

1. $\bigcup_{i \in I} X_i = V$;
2. *para toda aresta $(u, v) \in E$, existe $i \in I$ tal que $u, v \in X_i$;*
3. *para todos $i, j, k \in I$, se j está no caminho entre i e k em T, então $X_i \cap X_k \subseteq X_j$.*

A *largura em árvore*, $LA_G(D)$, de uma decomposição $D = (\mathcal{X} = \{X_i, i \in I\}, T = (I, F))$ é definida como $\max_{i \in I}\{|X_i| - 1\}$. A largura em árvore de um grafo, $LA(G)$, é a largura mínima dentre todas as possíveis decomposições em árvore desse grafo. Chama-se cada X_i de *sacola* da decomposição. A Figura 3 mostra uma decomposição em árvore de um grafo.

Uma decomposição em árvore D é *própria*, se D satisfaz a seguinte afirmação: para todos índices k, tal que k está em um caminho entre i e j em T, tem-se que $|X_i \cap X_j| \leq |X_k| - 2$. A *largura em árvore própria de um grafo*, $LA_p(G)$, é a largura mínima dentre todas as possíveis decomposições em caminho próprias desse grafo.

Uma decomposição em caminho é uma decomposição em árvore na qual a árvore da decomposição é um caminho. A seguir, é apresentada a definição de decomposição em caminho utilizando uma notação mais simples.

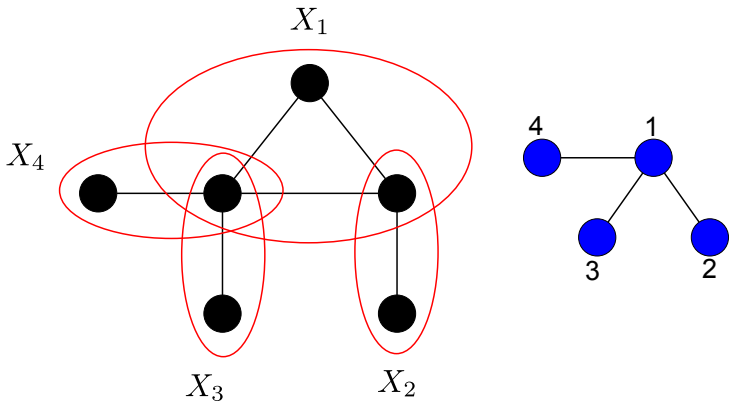


Figura 3: Decomposição em árvore de um grafo. Em vermelho estão representadas as sacolas da decomposição e em azul a árvore dessa decomposição.

Definição 2.2. *Uma* decomposição em caminho *de um grafo* $G = (V, E)$ *é uma sequência* $P = \langle X_0, X_1, X_2, \ldots, X_k \rangle$*, na qual cada* X_i*, com* $0 \leq i \leq k$*, é um subconjunto de* V*, satisfazendo as seguintes propriedades:*

1. $\bigcup_{i \in [0,k]} X_i = V$;
2. *para toda aresta* $(u, v) \in E$*, existe* $i \in [0, k]$ *tal que* $u, v \in X_i$;
3. *para todos* $i, j, k \in [0, k]$*, se* $i \leq j \leq k$*, então* $X_i \cap X_k \subseteq X_j$.

A *largura em caminho*, $LC_G(P)$, de uma decomposição $P = \langle X_0, X_1, X_2, \ldots, X_k \rangle$ é definida como $\max_{i \in [0,k]} \{|X_i| - 1\}$. A largura em caminho de um grafo, $LC(G)$, é a largura mínima dentre todas as possíveis decomposições em caminho desse grafo. Da mesma forma, chama-se cada X_i de *sacola* da decomposição. A Figura 4 mostra uma decomposição em caminho de um grafo.

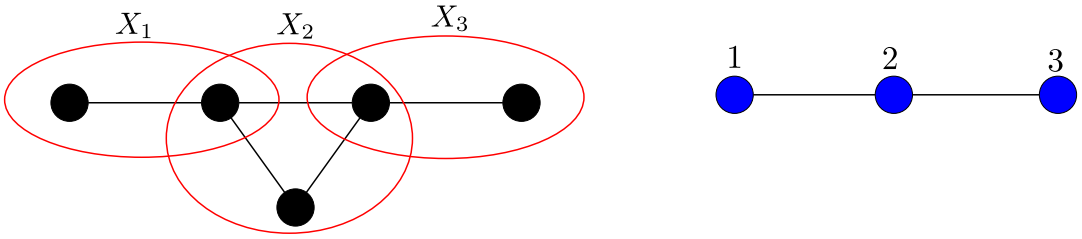


Figura 4: Decomposição em caminho de um grafo. Em vermelho estão representadas as sacolas da decomposição e em azul o caminho associado.

Uma decomposição em caminho P é *própria*, se P satisfaz a seguinte afirmação: para todos índices $i < k < j$ tem-se que $|X_i \cap X_j| \leq |X_k| - 2$. A *largura em caminho própria*

de um grafo, $LC_p(G)$, é a largura mínima dentre todas as possíveis decomposições em caminho próprias desse grafo.

A *decomposição em caminho direcionada* é uma decomposição em caminho, considerando um grafo direcionado. A diferença é a substituição da regra 2 da decomposição em caminho por:

2'. para toda aresta $u \to v \in E$, existe $i, j \in [0, k]$ com $i < j$ tal que $u, v \in X_i$ ou $u \in X_i$ e $v \in X_j$.

Como nas outras decomposições, a *largura em caminho direcionada* de uma decomposição em caminho direcionada de um grafo G é dada por $\max\{|X_i| : i \in [0, k]\}$. A *largura em caminho direcionada* de um grafo é a menor largura em caminho dentre todas as decomposições em caminho direcionadas desse grafo.

Se em um grafo $G = (V, E)$ existe uma família de subconjuntos $B = \{V_0, \ldots, V_k\}$ de $V(G)$ tal que para todo par $(i, j) \in [0, k]^2$ os conjuntos V_i e V_j se tocam, então B é um *arbusto* de G.

2.4 Outros conceitos

Uma *ordenação linear* de um grafo $G = (V, E)$ é uma bijeção $L : V \to \{1, 2, 3, \ldots, |V|\}$. Seja $V_L(i) = \{x | L(x) \leq i$ e $L(y) > i$ para algum $y \in V$ tal que $(x, y) \in E\}$. Uma *separação de vértices* de um grafo $G = (V, E)$ com respeito a uma ordenação linear L, denotada por $vs_L(G)$ é definida por:

$$vs_L(G) = \max\{|V_L(i)||1 \leq i \leq |V|\}$$

Adicionalmente, a *separação de vértices* do grafo G é definida por:

$$vsep(G) = min\{vs_L(G)|\text{L é uma ordenação linear de } G\}$$

A *largura de banda* de um grafo $G = (V, E)$ sobre uma ordenação linear, L, é o valor $b(G, L) = \max_{(u,v)\in E(G)} |L(v) - L(u)|$. A *largura de banda* de um grafo é a menor das larguras de banda sobre uma ordenação linear qualquer do grafo. A *largura de banda topológica* de um grafo G é a menor largura de banda de um grafo G', considerando todos os grafos G' que podem ser obtidos a partir de G subdividindo as arestas de forma independente um número arbitrário de vezes.

Dado um grafo não orientado $G = (V, E)$. Um *transversal de ciclos* é um subconjunto de vértices S de G que intersecta todos os ciclos de G. Logo $G - S$ é acíclico.

A *ordenação linear das arestas* de um grafo G é uma bijeção $L : E \to \{1, \ldots, |E|\}$. A *largura de uma ordenação linear* de um grafo G é o menor inteiro k tal que para todo número inteiro $i \in [1, |E| - 1]$ há no máximo k vértices que são extremidades de arestas de ambos os conjuntos $\{L^{-1}(1), L^{-1}(2), \ldots, L^{-1}(i)\}$ e $\{L^{-1}(i + 1), L^{-1}(i + 2), \ldots, L^{-1}(|E|)\}$ para uma ordenação linear, L, das arestas de G. A *largura linear* de um grafo G é a menor largura de uma ordenação linear de G dentre todas as possíveis ordenações lineares.

A menos que seja dito o contrário, todos os grafos considerados neste texto são grafos simples.

3 Elementos da Procura em Grafos

Neste capítulo, são apresentados os principais conceitos relacionados ao problema da *Procura em Grafos*. Inicialmente, alguns termos apresentados no Capítulo 1 serão redefinidos para uma melhor formalização do problema.

3.1 Procura em Vértices

No Capítulo 1, uma estratégia de procura foi definida. Nessa seção, é apresentada uma outra definição, equivalente, mais precisa. Nos capítulos e seções subsequentes, será esta a definição adotada.

Em uma das formulações clássicas do problema de Procura em Grafos, proposta por Kirousis e Papadimitriou [10], e chamada de *Procura em Vértices*, deseja-se desenvolver uma estratégia para capturar um fugitivo que se esconde em um vértice do grafo, porém sua posição é desconhecida, ele é arbitrariamente rápido e sempre faz a melhor decisão quanto à sua movimentação.

Definição 3.1. *Dado um grafo $G = (V, E)$, uma* estratégia de procura*, ou simplesmente* procura*, é uma sequência $S = \langle (A_0, C_0), (A_1, C_1), \ldots, (A_s, C_s) \rangle$. Cada (A_i, C_i) é chamado de* passo *ou* etapa *da procura. No passo i da procura, o conjunto A_i representa as posições dos policiais e o conjunto C_i o subgrafo livre do fugitivo. Nessa sequência, tem-se que $C_0 = (\emptyset, \emptyset)$, $A_0 = \emptyset$, $C_s = G$ e, para todo $i \in [0, s-1]$, uma das operações seguintes é executada:*

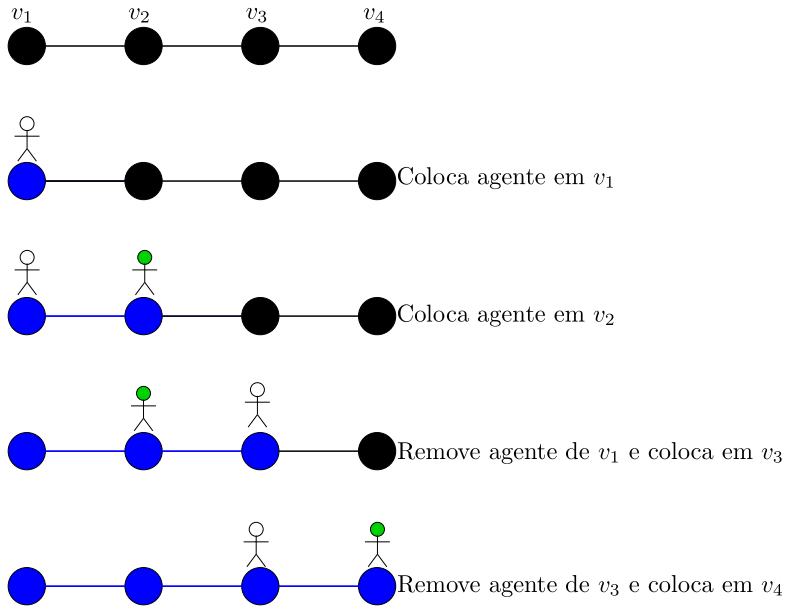
Adicionar: *$A_i \subset A_{i+1}$, significando que policiais são adicionados à procura, $C_{i+1} = G[V(C_i) \cup A_{i+1}]$. Intuitivamente, C_{i+1} é a união do subgrafo limpo no passo i com os vértices ocupados no passo $i+1$ e as arestas entre os vértices de A_{i+1} ou C_i; ou*

Remover: *$A_{i+1} \subset A_i$, significando que policiais são removidos da procura, $V(C_{i+1}) =$*

$V(C_i)\backslash\{u \in V(C_i)|$ *existe um caminho livre de agentes no passo* $i+1$ *entre* u *e* $v, v \in \overline{C_i}\}$ *e* $E(C_{i+1}) = E(C_i)\backslash\{(u,v)|$ *existe um caminho livre de agentes no passo* $i+1$ *entre* u *ou* v *e* $w, w \in \overline{C_i}\}$. *É importante observar que, ao remover agentes da procura, tanto os vértices quanto as arestas que podem ser alcançados pelo fugitivo deixam de ser "livres do fugitivo".*

Diz-se que os vértices e as arestas em C_i estão *limpos* no passo i. Os vértices, ou arestas, que não estão limpos em uma etapa i são chamados de *contaminados*. Se existem índices i e j com $i < j$ tal que para algum vértice $v \in V(C_i)$ e $v \notin V(C_j)$, então o vértice v está *recontaminado* no passo j. Da mesma forma, uma aresta (u, v) está recontaminada no passo j se $(u, v) \in E(C_i)$ e $(u, v) \notin E(C_j)$. O número de agentes utilizados em uma procura é dado por $max_{i\in[0,s]}\{|A_i|\}$.

A Figura 5 mostra um exemplo de estratégia utilizando dois policiais para limpar um caminho, pois são necessários pelo menos dois agentes para limpar grafos com uma ou mais arestas.



Abaixo a mesma estratégia de procura na notação matemática:

$C_0 = (\emptyset, \emptyset), C_1 = (\{v_1\}, \emptyset), C_2 = (\{v_1, v_2\}, \{(v_1, v_2)\}), C_3 = C_2$
$A_0 = \emptyset, A_1 = \{v_1\}, A_2 = \{v_1, v_2\}, A_3 = \{v_2\}, A_4 = \{v_2, v_3\}, A_5 = \{v_3\}, A_6 = \{v_3, v_4\}$
$C_4 = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3)\}), C_5 = C_4, C_6 = G$

Figura 5: Uma estratégia para limpar um caminho utilizando 2 policiais. Os dois policiais são diferenciados pelas cores branca e verde. Os conjuntos C_i estão representados pela cor azul.

Uma das questões que surgem naturalmente a respeito desse problema é qual o menor

número de agentes necessários para que exista uma procura em um grafo. Responder a essa questão é um problema *NP-difícil*, já que responder a pergunta: "dado um grafo G e um inteiro k, existe uma procura que utilize no máximo k agentes?" é um problema *NP-completo* [11, 12].

Dado um grafo $G = (V, E)$, o *número de Procura em Vértices*, $vs(G)$ (do inglês *vertex search*), é o menor inteiro k tal que existe uma estratégia de procura em vértices no grafo G com k agentes.

Um primeiro resultado importante relaciona a largura em caminho de um grafo G com o número de Procura em Vértices desse grafo, como é mostrado no Teorema 3.1.

Teorema 3.1 (Kirousis, Papadimitriou [12, 13])**.** *Seja $G = (V, E)$ um grafo e $P = \langle X_0, X_1, X_2, \ldots, X_k \rangle$ a menor decomposição em caminho desse grafo com respeito ao tamanho da maior sacola. Tem-se:*

$$vs(G) = LC_G(P) + 1$$

Prova: Primeiro, mostra-se que $vs(G) \leq LC_G(P) + 1$. Dada a decomposição P, será construída uma estratégia a partir dessa decomposição. A estratégia será: para cada $i \in [0, k-1]$, de forma sucessiva, ocupar todos os vértices de uma sacola X_i e em seguida remover os agentes dos vértices em $X_i - X_{i+1}$, ou seja, apenas os vértices em $X_i \cap X_{i+1}$ continuam ocupados. No último passo, ocupa-se todos os vértices da sacola X_k.

Já que P é uma decomposição em caminho, não existem arestas (u, v) entre vértices de $X_i \Delta X_{i+1}$ com $i \in [0, k-1]$, $u \in X_i$ e $v \in X_{i+1}$. Então não existe recontaminação entre uma fase de ocupação e uma fase de remoção. Portanto, ao final da procura, todas as arestas estarão limpas. O maior número de agentes usado em um passo dessa procura é igual o tamanho da maior sacola da decomposição P.

Para mostrar que $LC_G(P) + 1 \leq vs(G)$, transforma-se uma estratégia de procura em uma decomposição em caminho. Sabe-se que existem estratégias nas quais não há recontaminação utilizando exatamente $vs(G)$ agentes [11]. Seja S uma procura em G com essa propriedade e minimal no número de agentes utilizados em cada passo. Seja $P = \langle A_0, A_1, \ldots, A_s \rangle$. Mostrar-se-á que P é uma decomposição em caminho de G.

Observando que S é uma estratégia de procura que, por definição, limpa todas as arestas do grafo, chega-se a conclusão de que para todo vértice $v \in V(G)$ existe $i \in [0, s]$ tal que $v \in A_i$ (satisfazendo a regra 1 da Definição 2.2). Também se conclui que só se pode limpar uma aresta $(u, v) \in E(G)$ se u e v forem ocupados simultaneamente. Assim, para

toda aresta $(u, v) \in E(G)$ os vértices u e v estão em A_i para algum $i \in [0, s]$ (satisfazendo a regra 2 da Definição 2.2). Suponha que existem índices $i < k < j$, i o maior possível e j o menor possível, tais que $v \in A_i \cap A_j$ e $v \notin A_k$. Como não há recontaminação em S, ocupar o vértice v no passo i deve resultar em todas as arestas incidentes à v estão limpas no passo $k-1$, pois do contrário haveria recontaminação do vértice v no passo k. Como a procura é minimal no número de agentes utilizados em cada passo, se todas as arestas incidentes à v foram limpas no passo i, então existe uma S' ótima utilizando um menor número de agentes no passo j (basta não ocupar o vértice v). Isso seria um absurdo, pois S é minimal no número de agentes usados em cada passo (satisfazendo a regra 3 da Definição 2.2). Logo P é uma decomposição em caminho. Como $LC_G(P) = \max_i\{A_i\} - 1$, $LC_G(P) + 1 = vs(G)$. □

3.2 Procura em Arestas

Uma das versões do problema de Procura em Grafos é a Procura em Arestas. Nessa variação o fugitivo se esconde tanto nos vértices quanto nas arestas do grafo e não é possível limpar as arestas do grafo ocupando simultaneamente as suas extremidades. É necessária a utilização de uma operação "atravessar" para fazer isso. Essa operação reflete um policial movendo-se entre dois vértices adjacentes, enquanto verifica a existência do fugitivo na aresta que interliga esses dois vértices.

Uma *estratégia de procura em arestas* pode ser caracterizada por uma sequência $S = \langle(A_0, C_0), (A_1, C_1), \ldots, (A_s, C_s)\rangle$. Entretanto, nesse tipo de estratégia, os policiais devem "caminhar" por uma aresta para limpá-la e as arestas só podem ser limpas dessa forma, também é possível que mais de um agente ocupe um mesmo vértice. Tem-se que $C_0 = (\emptyset, \emptyset)$, $A_0 = \emptyset$, $C_s = G$ e, para todo $i \in [0, s-1]$, uma das seguintes operações é executada:

Adicionar: $A_i \subset A_{i+1}$, significando que policiais foram adicionados à procura, $V(C_{i+1}) = V(C_i) \cup A_{i+1}$. Diferentemente de uma estratégia de procura em vértices, nenhuma aresta pode ser limpa dessa forma; ou

Remover: $A_{i+1} \subset A_i$, significando que policiais foram removidos da procura, $V(C_{i+1}) = V(C_i)\backslash\{u \in C_i |$ existe um caminho livre de agentes entre u e $v \in \overline{C_i}$ em $V(G)\backslash A_{i+1}\}$ e $E(C_{i+1}) = E(C_i)\backslash\{(u, v) |$ existe um caminho livre de agentes entre u ou v e $w \in \overline{C_i}$ em $V(G)\backslash A_{i+1}\}$; ou

Atravessar: $A_i \Delta A_{i+1} = \{w, y\}$, com $w \in A_i$, $y \in A_{i+1}$ e $(w, y) \in E(G)$, signifi-

cando que um policial "caminhou" sobre uma aresta (w, y) na direção de w para y. $V(C_{i+1}) = V(C_i) \cup \{y\} \backslash \{u \in C_i |$ existe um caminho livre de agentes entre u e $v \in \overline{C_i}$ em $V(G) \backslash A_{i+1}\}$ e $E(C_{i+1}) = E(C_i) \cup \{(w, y)\} \backslash \{(u, v) |$ existe um caminho livre de agentes entre u ou v e $w \in \overline{C_i}$ em $V(G) \backslash Ai + 1\}$. É importante observar que o fugitivo pode utilizar as extremidades de (w, y) para se locomover "enquanto" o policial "caminha" sobre essa aresta.

Dado um grafo $G = (V, E)$, o *número de Procura em Arestas*, $es(G)$ (do inglês *edge search*), é o menor inteiro k tal que existe uma estratégia de procura em arestas no grafo G com k agentes.

É possível construir uma estratégia de procura em arestas em um caminho que utilize apenas um policial, diferentemente da estratégia de procura em vértices que necessita de pelo menos dois agentes, pois precisamos de dois agentes para limpar uma aresta na procura em vértices. A Figura 6 mostra uma procura em arestas de um fugitivo em um caminho.

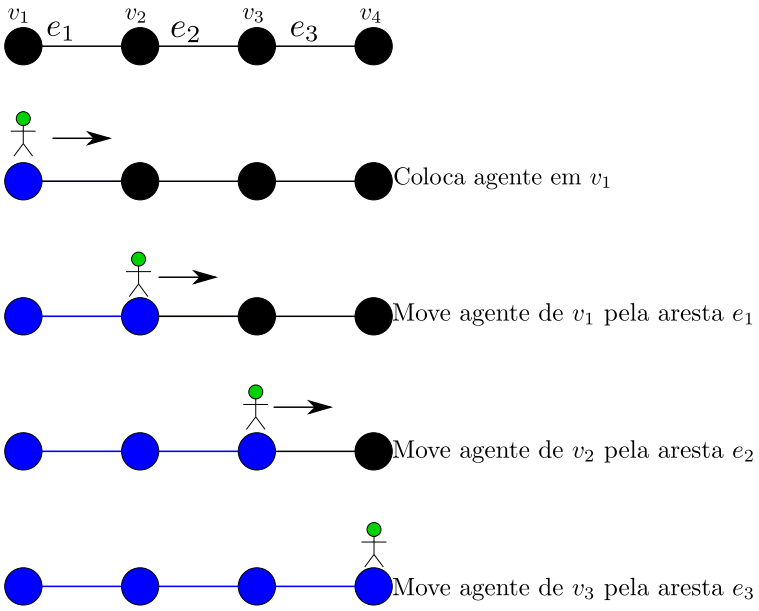


Figura 6: Uma estratégia para limpar um caminho utilizando apenas um policial. Os conjuntos C_i estão representados em azul.

3.3 Procura Mista

O problema da Procura Mista proposto por Bienstock e Seymour em 1991 [11] mescla os problemas da Procura em Vértices e da Procura em Arestas. Nesse problema o fugitivo

se esconde tanto nos vértices quanto nas arestas do grafo e as arestas podem ser limpas ocupando simultaneamente suas extremidades ou por uma operação "atravessar".

Uma *estratégia de procura mista* é uma sequência $S = \langle (A_0, C_0), (A_1, C_1), \ldots, (A_s, C_s) \rangle$. Nesse tipo de estratégia, as arestas podem ser limpas ocupando simultaneamente suas extremidades, ou utilizando a operação "atravessar". Tem-se que $C_0 = A_0 = \emptyset$, $C_s = G$ e para todo $i \in [0, s-1]$:

Adicionar: $A_i \subset A_{i+1}$,significando que policiais são adicionados à procura, $C_{i+1} = G[V(C_i) \cup A_{i+1}]$, da mesma forma que na estratégia de procura em vértices, podemos limpar arestas dessa forma; ou

Remover: $A_{i+1} \subset A_i$, significando que policiais foram removidos da procura, $V(C_{i+1}) = V(C_i) \backslash \{u \in C_i |$ existe um caminho livre de agentes entre u e $v \in \overline{C_i}$ em $V(G) \backslash A_{i+1}\}$ e $E(C_{i+1}) = E(C_i) \backslash \{(u, v) |$ existe um caminho livre de agentes entre u ou v e $w \in \overline{C_i}$ em $V(G) \backslash A_{i+1}\}$; ou

Atravessar: $A_i \Delta A_{i+1} = \{w, y\}$, com $w \in A_i$, $y \in A_{i+1}$ e $(w, y) \in E(G)$, significando que um policial "caminhou" sobre uma aresta (w, y) na direção de w para y. $V(C_{i+1}) = V(C_i) \cup \{y\} \backslash \{u \in C_i |$ existe um caminho livre de agentes entre u e $v \in \overline{C_i}$ em $V(G) \backslash A_{i+1}\}$ e $E(C_{i+1}) = E(C_i) \cup \{(w, y)\} \backslash \{(u, v) |$ existe um caminho livre de agentes entre u ou v e $w \in \overline{C_i}$ em $V(G) \backslash A_{i+1}\}$.

O problema relacionado a encontrar uma estratégia de procura mista do grafo com o menor número de agentes possível é o *Procura Mista*. Dado um grafo $G = (V, E)$, o *número de Procura Mista* de um grafo G, $ms(G)$ (do inglês *mixed search*), é o menor inteiro k tal que existe uma estratégia de procura mista no grafo G com k agentes.

3.4 Terminologia

Nesta seção, será introduzida a terminologia usada nos problemas de Procura em Grafos. Quando uma procura $S = \langle (A_0, C_0), \ldots, (A_s, C_s) \rangle$ em um grafo $G = (V, E)$ é livre de recontaminação, ou seja, C_i é subgrafo de C_{i+1} para todo $i \in [0, s-1]$, diz-se que essa é uma estratégia de procura *monotônica*. Para a Procura em Arestas e a Procura Mista, ainda é necessário que uma aresta "atravessada" no passo i esteja limpa no passo i, pois é possível que uma aresta "atravessada" seja imediatamente recontaminada. Uma procura em que C_i é um subgrafo conexo para todo $i \in [0, s-1]$ é *conexa*.

O fugitivo é *invisível* quando a sua localização no grafo é desconhecida. Caso contrário, ele é considerado *visível*. Quando o fugitivo é visível, se em um passo i são adicionados agentes à procura, então C_i é a união de C_{i-1} com todas as componentes A_i-componente (que é uma componente de $G[V(G) - A_i]$), com exceção da componente na qual o fugitivo se encontra. A Figura 7 exemplifica uma procura em arestas de um fugitivo visível em uma árvore.

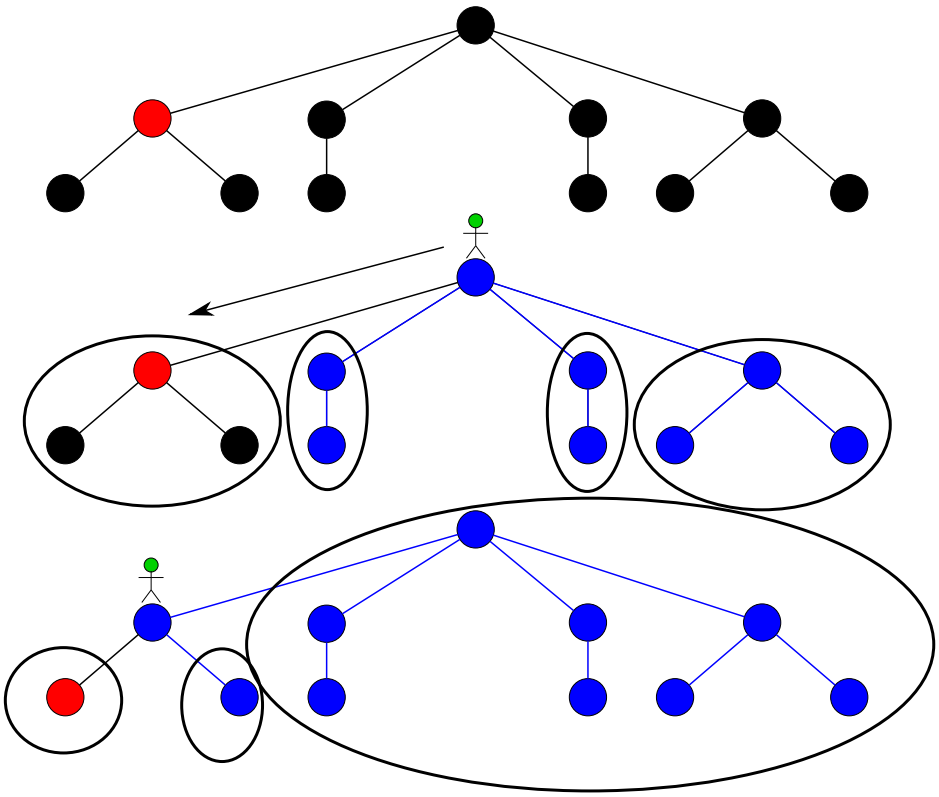


Figura 7: Uma estratégia de procura em arestas de um fugitivo visível em uma árvore utilizando apenas um policial. O fugitivo é representado pelo vértice em vermelho. As elipses representam as componentes conexas do grafo induzido pelos vértices não ocupados pelos agentes. Na primeira etapa, todas as arestas estão contaminadas. Na segunda etapa, coloca-se um agente na raiz da árvore. Nas etapas subsequentes, o policial atravessa a aresta ainda contaminada incidente ao vértice que ele ocupa.

Um fugitivo que só pode se mover quando um policial tentar ocupar o mesmo vértice em que ele se encontra é chamado de *preguiçoso*, caso contrário o fugitivo é *ativo*.

Uma estratégia de procura em arestas, ou mista, na qual a operação "remover" não é realizada é chamada de *interna*, significando que os agentes, após ocuparem um vértice, só podem se movimentar seguindo pelas arestas do grafo. Em uma estratégia de procura em vértices interna, a única solução é ocupar todos os vértices do grafo com agentes, portanto é trivial calcular esse parâmetro. Essa restrição é importante quando se está interessado

Tabela 1: Parâmetros

Restrições	Tipo de Procura		
	Procura em Vértice	Procura em Aresta	Procura Mista
Monotônica	$mvs(G)$	$mes(G)$	$mms(G)$
Visível	$\overline{vs}(G)$	$\overline{es}(G)$	$\overline{ms}(G)$
Conexa	$cvs(G)$	$ces(G)$	$cms(G)$
Interna	$ivs(G) = \|V(G)\|$	$ies(G)$	$ims(G)$
Preguiçosa	$lvs(G)$	$les(G)$	$lms(G)$

em modelar problemas nos quais os agentes não podem sair do grafo em um ponto para entrar em outro como, por exemplo, o problema de encontrar o explorador perdido no sistema de cavernas.

Como introduzidos antes, os parâmetros $vs(G), es(G), ms(G)$ são, respectivamente, o número de Procura em Vértices, o número de Procura em Arestas e o número de Procura Mista. Definir-se-á outros parâmetros para a inclusão dos conceitos de monotonicidade, internalidade, visibilidade, conectividade e atividade. A Tabela 1 mostra os parâmetros relacionados a cada tipo de procura quando é considerado apenas estratégias que satisfaçam as restrições correspondentes. O valor desses parâmetros é sempre o menor número de agentes necessários para que exista uma estratégia de procura referente à célula da tabela.

Pode-se, também, considerar estratégias que obedeçam a mais de uma das restrições, por exemplo, o menor número de agentes necessários para realizar uma procura em vértices visível e conexa em um grafo G é dado pelo parâmetro $\overline{cvs}(G)$.

4 Resultados Clássicos

Neste capítulo, são apresentados os resultados mais importantes sobre os problemas de Procura em Grafos.

4.1 Complexidade

Megido et al. [5] mostraram que é *NP-difícil* responder, para um grafo e um inteiro k, se existe uma estratégia de procura em arestas que limpe o grafo utilizando no máximo k policiais. Pela propriedade da monotonicidade da Procura em Arestas [14], conclui-se que esse problema é *NP-completo*. Os problemas de decisão relacionados à *Procura em Vértices* e à *Procura Mista* também são *NP-completos* [11,12], porém, quando o número de agentes é uma constante responde-se a essas perguntas em tempo linear [15, 16]. Isso é possível, pois há algoritmos capazes de encontrar uma decomposição em caminho de um grafo em tempo linear, quando um limite para a largura de decomposição é conhecido. Dessa forma, é fácil traduzir essa decomposição em caminho em uma estratégia de procura em vértices do grafo como foi mostrado no Teorema 3.1. Para os problemas da Procura em Arestas e da Procura Mista com o número de agentes fixo, é possível construir uma estratégia de procura baseada em uma ordenação linear das arestas do grafo. É interessante notar que um algoritmo para calcular a largura linear de um grafo G [15] é conhecido, pois a princípio as provas de que esses parâmetros podiam ser calculados em tempo linear não eram construtivas e, portanto, não geravam algoritmos para a resolução desses problemas [17].

Uma das maneiras usuais de provar que o problema de decisão associado a uma variação do problema de Procura em Grafos é *NP-completo* é mostrar que a restrição de monotonicidade não torna o problema mais difícil, ou seja, existem procuras ótimas monotônicas que utilizam o mesmo número de agentes que uma procura ótima sem essa restrição. A partir desse resultado, tenta-se formular uma decomposição do grafo que reflita uma procura monotônica dessa variação do problema de Procura, para então provar que decidir se essa decomposição satisfaz alguma propriedade é um problema *NP-completo*. A

Procura em Vértices, a Procura em Arestas e a Procura Mista são monotônicas [11,14,18], significando que existe uma estratégia livre de recontaminação utilizando $vs(G)$, $es(G)$, $ms(G)$, respectivamente, policiais.

Mesmo com as restrições de internalidade e/ou conectividade, o problema de decisão relacionado à Procura em Arestas é *NP-completo* [5]. É importante notar que nem todos os problemas de Procura em Grafos são monotônicos. A Procura em Arestas, quando restrita a estratégias conexas, de um fugitivo visível ou invisível não é monotônica [19,20], ou seja, existem grafos G tais que $mces(G) > ces(G)$ ou $\overline{mces}(G) > \overline{ces}(G)$.

Não é surpreendente que a Procura Mista generalize a Procura em Vértices e a Procura em Arestas [11, 21]. Uma estratégia de procura em vértices pode ser convertida em uma estratégia de procura mista, o mesmo acontece com uma estratégia de procura em arestas.

Takahashi, Ueno e Kajitani [21] mostraram a relação entre o número de Procura Mista, de Procura em Vértices e de Procura em Arestas. Observa-se que $vs(G) - 1 \leq ms(G) \leq vs(G)$, uma vez que qualquer Procura Mista pode ser convertida em uma Procura em Vértices, trocando cada passo "atravessar" uma aresta $u \to v$ por adicionar um agente em v e remover o agente de u. Logo $vs(G) \leq ms(G) + 1$. Utilizando a mesma ideia, eles demonstraram que $es(G) - 1 \leq ms(G) \leq es(G)$.

4.2 Procura em Vértices

A monotonicidade da Procura em Vértices foi generalizada por Mazoit e Nisse [14]. Eles consideraram o caso em que o fugitivo é invisível, mas pode-se perguntar a um oráculo a posição atual do fugitivo ao final de qualquer etapa da procura. Existe um limite pré-determinado, no início da procura, de quantas vezes é possível questionar o oráculo quanto à posição do fugitivo e esse número pode variar de 0 a ∞. Quando esse número é igual a 0 esse problema é idêntico à Procura em Vértices invisível. Quando esse número é ∞, o problema se torna equivalente à Procura em Vértices visível.

Para todo grafo G, sabe-se que $LC(G) - 1 = vs(G)$ (Teorema 3.1). Há também outras decomposições em grafos relacionadas à Procura em Vértices. A separação de vértices de um grafo é igual a largura em caminho desse mesmo grafo [12, 13]. Logo, dado um grafo G, tem-se que $vsep(G) - 1 = vs(G)$. Papadimitriou et al. mostraram que o número de Procura em Vértices é inferior ou igual a largura de banda topológica de um grafo G [22], mostrando ainda que esses números são idênticos, quando $\Delta(G)$ é menor ou igual a três.

Ao analisar o caso em que o fugitivo é visível, deve-se utilizar uma decomposição que reflita essa informação extra. Seymour e Thomas mostraram que o número de Procura em Vértices de um grafo G, considerando um fugitivo visível, é equivalente a largura em árvore menos um, $\overline{vs}(G) = LA(G) - 1$ [23]. Eles ainda mostraram que a Procura em Vértices de um fugitivo invisível e preguiçoso (definida em [24]) é equivalente a capturar um fugitivo visível e ativo, ou seja, $lvs(G) = \overline{vs}(G)$ [23,24]. Um outro conceito equivalente à Procura em Vértices é o de arbusto, pois se $vs(G) = k$ então G possui um arbusto, B, tal que nenhum conjunto X com $|X| \leq k$ intercepta todos os conjuntos de B.

É importante ressaltar que existe um algoritmo para construir uma decomposição em árvore de um grafo G com largura mínima se a largura em árvore de G é limitada por alguma constante [25]. Assim, quando $\overline{vs}(G)$ é limitado, é possível encontrar uma estratégia de procura em vértices para G mínima em tempo linear.

Existem várias classes de grafos nas quais calcular o número mínimo de agentes para que exista uma estratégia de procura em vértices é NP-difícil. Dentre essas classes, é possível citar: grafos planares com grau máximo três, grafos bipartidos, grafos co-bipartidos e grafos "starlike" [26]. Por outro lado, também existem classes de grafos nas quais calcular esse número leva um tempo polinomial como por exemplo: árvores, co-grafos, grafos de permutação, grafos trapezoidais, grafos "k-starlike" e k-árvores para um k fixo [26]. Para uma visão mais geral da complexidade da procura em vértices em diversas classes de grafos é recomendada a leitura da Tabela 2.

4.3 Procura em Arestas

Barrière, Fraigniaud, Santoro e Thilikos [27,28] relacionaram os vários tipos de estratégia de procura em arestas como é mostrado nos Teoremas 4.1 e 4.2.

Teorema 4.1 ([27,28])**.** *Dado um grafo conexo G, tem-se:*

$$es(G) = ies(G) = mes(G) \leq mies(G) \leq ces(G) = ices(G) \leq mces(G) = mices(G).$$

É fácil ver que $ies(G) = es(G)$, pois a operação de remoção de um agente de um vértice v seguida da adição de um agente em um vértice u pode ser substituída por várias operações de atravessar com esse agente por um caminho entre u e v. Esse resultado também mostra que, ao exigir conectividade de uma estratégia de procura em arestas, o número de agentes necessários pode subir. O Teorema 4.2 mostra o relacionamento entre os parâmetros do Teorema 4.1 quando apenas árvores são consideradas.

Teorema 4.2 ([27, 28]). *Dado uma árvore T, tem-se:*

$$es(T) = ies(T) = mes(T) \leq mies(T) = ces(T) = ices(T) = mces(T) = mices(T) \leq 2es(T)-2.$$

O Teorema 4.2 mostra que, em uma árvore, o número de agentes de uma procura em arestas conexa está limitado a um fator de duas vezes o de uma estratégia sem essa restrição.

Nisse demostrou que, para um grafo cordal conexo G, $ces(G)/es(G) \leq O(LA(G))$ [29], apresentando também um algoritmo para construir uma estratégia de procura em arestas conectada para um grafo cordal conexo G que utiliza no máximo $(LA(G)+2)(es(G)-1)$ agentes.

Fomin, Thilikos e Todinca [30] mostraram que se G é um grafo periplanar e T seu dual fraco então $ces(G) \leq 2ces(T) + 1$. Caso G seja um grafo 2-conexo periplanar, eles também construíram um algoritmo 4-aproximativo para calcular $ces(G)$.

É um problema em aberto determinar se para um grafo geral G, $ces(G) \leq c*es(G)$, no qual c é uma constante qualquer [28]. Porém é possível demonstrar que $ces(G)/es(G) \leq \log|V(G)|+1$ [31]. Caso essa conjectura se mostre verdadeira, a existência de um algoritmo aproximativo com fator c para encontrar $mies(G), ces(G), ices(G), mces(G), mices(G)$ em classes de grafos com largura em caminho limitada é provada pelo Teorema 4.1.

Há classes de grafos nas quais calcular o número de agentes para que exista uma estratégia de procura em arestas é NP-difícil como, por exemplo, grafos planares nos quais o grau máximo é três e grafos "starlike" [26]. Por outro lado, também existem classes de grafos nas quais calcular esse número leva um tempo polinomial, como por exemplo: árvores, grafos de intervalo, grafos "split", grafos "k-starlike" para um k fixo [26] e grafos unicíclicos [32]. Para uma visão mais geral da complexidade da Procura em Arestas em diversas classes de grafos é recomendada a leitura da Tabela 2.

Limites inferiores e superiores para o número de procura em arestas de um grafo foram estudados por Alspach et al [33]. Eles melhoraram o limite inferior anteriormente conhecido para grafos com $\delta(G) \geq 3$, que era $\delta(G) + 1$ [20], para $\delta(G)g(G) - 2$ no qual $g(G)$ é o tamanho da cintura do grafo G. Outro resultado obtido é: sendo G é um grafo conexo então $\chi(G) - 1 \leq es(G)$.

4.4 Procura Mista

Takahashi, Ueno e Kajitani [21] mostraram que a decomposição em caminho própria de um grafo G é igual ao número de Procura Mista de G, $LC_p(G) = ms(G)$. Um outro resultado mostra um algoritmo em tempo polinomial para calcular $LC_p(G)$, se G é uma árvore. Porém, como foi citado anteriormente, calcular $LC_p(G)$ para um grafo qualquer é um problema *NP-difícil*.

O problema da Procura Mista, quando há mais de um fugitivo no grafo, foi estudado por Richerby e Thilikos [34]. É interessante notar que, em estratégias de procura nas quais os fugitivos são invisíveis, o número desses fugitivos no grafo é irrelevante, pois o número de agentes para capturar qualquer número de fugitivos invisíveis é exatamente igual a largura em caminho própria do grafo mais um. Portanto, eles estudaram também as estratégias de procura nas quais os fugitivos são visíveis, ou seja, a cada passo da procura sabe-se exatamente a posição de cada fugitivo. A Procura Mista com vários fugitivos visíveis não é monotônica. A Figura 8 mostra um exemplo de um grafo no qual exigir monotonicidade aumenta o número de agentes necessários.

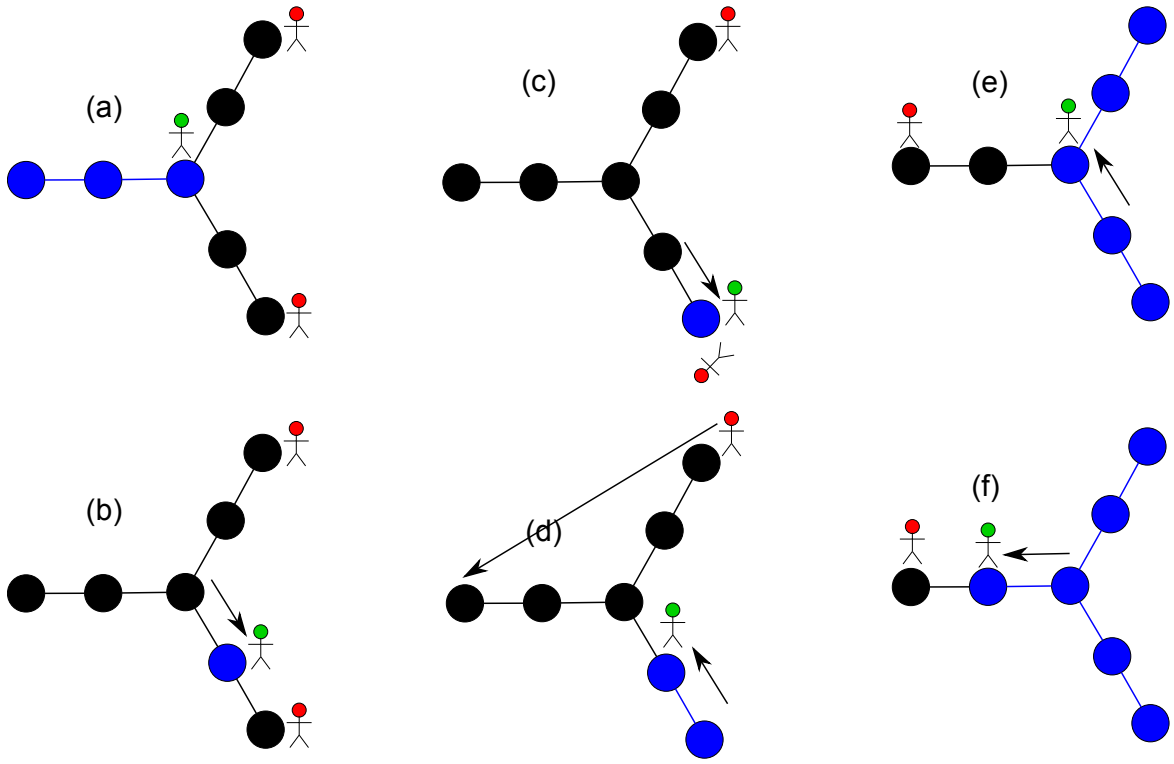


Figura 8: Exemplo de grafo no qual uma procura mista visível com dois fugitivos utiliza menos agentes que essa mesma procura com a restrição de monotonicidade. A cor vermelha representa os fugitivos, a cor verde o agente e a cor azul o subgrafo limpo.

Se T é uma árvore, então o número de agentes necessários para capturar r fugitivos utilizando estratégias de procura mista monotônicas para T, $\overline{mms}(T)$, é igual à $\min\{LC_p(T), \lfloor \log r \rfloor + 1\}$. É fácil ver que para capturar apenas um fugitivo nessas condições, um agente é suficiente, pois basta escolher um vértice qualquer da árvore e aplicar, sucessivamente, operações de "atravessar" para o vértice da componente conexa em que o fugitivo se encontra como na Figura 7. A Figura 9 mostra o porquê do valor $\lfloor \log r \rfloor + 1$, visto que os fugitivos podem se deslocar em direções diferentes cada vez que um agente se move. Para um grafo G qualquer, esse mesmo valor é limitado superiormente por $\min\{LC_p(G), LA_p(G) * (\lfloor \log r \rfloor + 1)\}$, pois até $\lfloor \log r \rfloor + 1$ sacolas da árvore de decomposição devem ser ocupadas por agentes. Da mesma forma que na Procura em Vértices, o número de agentes necessários para capturar um fugitivo visível e ativo com uma estratégia de procura mista é igual ao necessário para capturar um fugitivo preguiçoso e invisível.

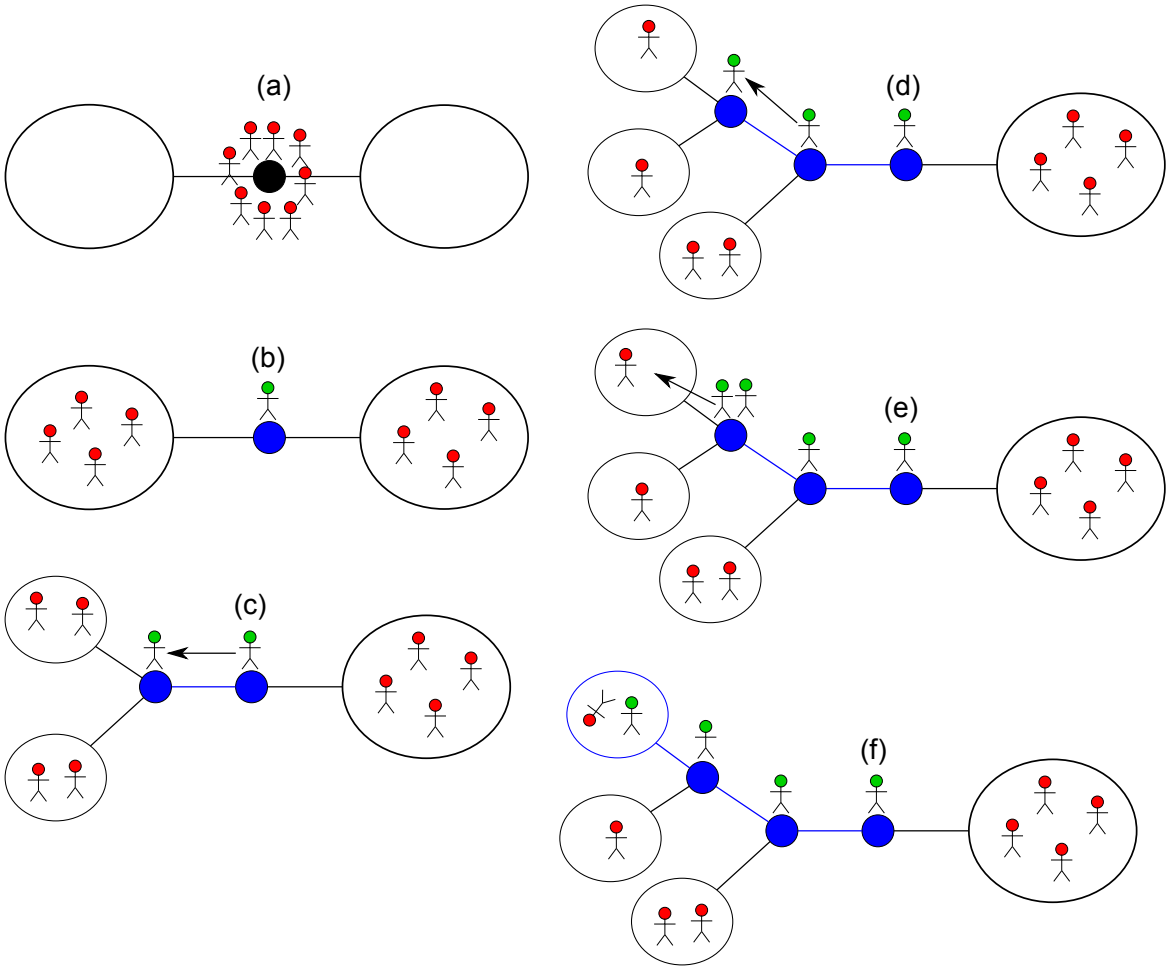


Figura 9: Exemplo de procura mista visível de 8 fugitivos em uma árvore. A cor vermelha representa os fugitivos, a cor azul o subgrafo limpo, a cor verde os agentes e as elipses representam componentes da árvore.

Dentre as classes de grafos nas quais é possível calcular o número de procura mista em tempo polinomial, podem ser citados os grafos de intervalo e os grafos split [35]. Em especial, para os grafos de intervalo esse tempo é linear. Para uma visão mais geral da complexidade da procura mista em diversas classes de grafos é recomendada a leitura da Tabela 2.

Tabela 2: Complexidade dos algoritmos existentes para resolver o problema de procura em grafo de um fugitivo invisível, considerando um grafo G com n vértices, m arestas e largura em caminho p.

Classe de Grafo	Procura em Vértices	Procura em Arestas	Procura Mista
Grafos "k-starlike" (k fixo)	$O(mn^k)$ [26]	$O(mn^k)$ [26]	X
Grafos "starlike"	NP-difícil [36]	X	X
Grafos de intervalo	Polinomial [37]	$O(n+m)$ [26]	Linear [35]
Árvores	Linear [38]	Linear [39]	Linear [21]
Grafos unicíclicos	$O(n \log n)$ [40]	Linear [32]	X
Grafos planares ($\Delta(G) \leq 3$)	NP-difícil [12]	NP-difícil [41]	X
Grafos bipartidos	NP-difícil [42]	X	X
Grafos co-bipartidos	NP-difícil [43]	X	X
Co-grafos	Linear [44]	X	X
Grafos de permutação	$O(pn)$ [45]	X	X
Grafos trapezoidais	$O(n^2)$ [46]	X	X
k-árvores (k fixo)	Linear [47]	X	X
Grafos split	Polinomial [37]	$O(mn^2)$ [26]	Polinomial [35]
Grafos cordais	X	NP-difícil [26]	X

4.5 Procura em Grafos Direcionados

Os problemas de Procura em Grafos são, normalmente, definidos para grafos não-direcionados, porém Boting Yang e Yi Cao consideraram os problemas de procura quando aplicados a grafos direcionados [48]. Em uma estratégia de procura em arestas direcionada, as mesmas regras para uma estratégia de procura em arestas são usadas, no entanto os agentes só podem "atravessar" uma aresta $u \to v$ no sentido de u para v e, da mesma forma, o fugitivo só pode alcançar os vértices seguindo caminhos orientados. Na versão mista direcionada, as arestas podem ser limpas tanto pela operação "atravessar", seguindo a orientação das arestas, quanto pela ocupação simultânea das extremidades de uma aresta. Os problemas da Procura Mista Direcionada e da Procura em Arestas Direcionada são *NP-completos* e monotônicos [48]. Boting Yang e Yi Cao também propuseram variações desses dois problemas nas quais os agentes podem "atravessar" as arestas independentemente do sentido da aresta, mas o fugitivo ainda é restrito a movimentar-se somente no sentido da aresta [49]. Eles mostraram que ambas são *NP-completas* e monotônicas.

5 *Limpeza de Vizinhança*

Um outro problema relacionado com a Procura em grafos é a *Limpeza de Vizinhança*. Ela pode ser vista como um problema de Procura em Vértices em grafos orientados simétricos, quando é possível limpar um vértice se toda a sua vizinhança já estiver limpa ou ocupada. Em outras palavras, vértices podem ser limpos sem a necessidade da ocupação por um agente, porém considerando apenas estratégias monotônicas. A Figura 10 mostra um exemplo de estratégia de limpeza em um ciclo.

Na Limpeza de Vizinhança, o interesse é calcular o menor número de agentes necessários para uma estratégia que limpe todos os vértices de um grafo orientado $G = (V, E)$. A *estratégia de limpeza* de uma Limpeza de Vizinhança é uma sequência $S = \langle (A_0, C_0), (A_1, C_1), \ldots, (A_s, C_s) \rangle$. É importante observar que, nesse problema, um agente não pode ser removido enquanto seu vértice estiver contaminado. Tem-se $C_0 = \{v \in V | N^+(v) \subseteq A_0\}$, $A_0 \subseteq V$, $C_s = V$ e, para todo $i \in [0, s-1]$, as seguintes operações são realizadas:

Mover: $A_i - C_i \subseteq A_{i+1}$, significando que os agentes só podem se movimentar após a limpeza dos vértices que eles ocupam;

Limpar: $C_{i+1} = C_i \cup \{v \in V | N^+(v) \subseteq A_{i+1} \cup C_i\}$, significando que os vértices que podem ser limpos são aqueles em que toda a sua vizinhança de saída já foi limpa ou está ocupada por um agente.

O número de agentes usados nessa estratégia de limpeza é dado por $\max_{i \in [0,s]}\{|A_i|\}$. O *número de Limpeza*, $l(G)$, de um grafo orientado G é o menor inteiro k tal que existe uma estratégia de limpeza para G. Um passo ou etapa de uma estratégia de limpeza é um par (A_i, C_i).

Uma das aplicações da Limpeza de Vizinhança é a reconfiguração de roteamento de conexões em uma rede Wavelength-Division Multiplexing (WDM) [50, 51]. Em uma rede WDM, às vezes, o roteamento das conexões não pode ser mantido. Isso acontece por diver-

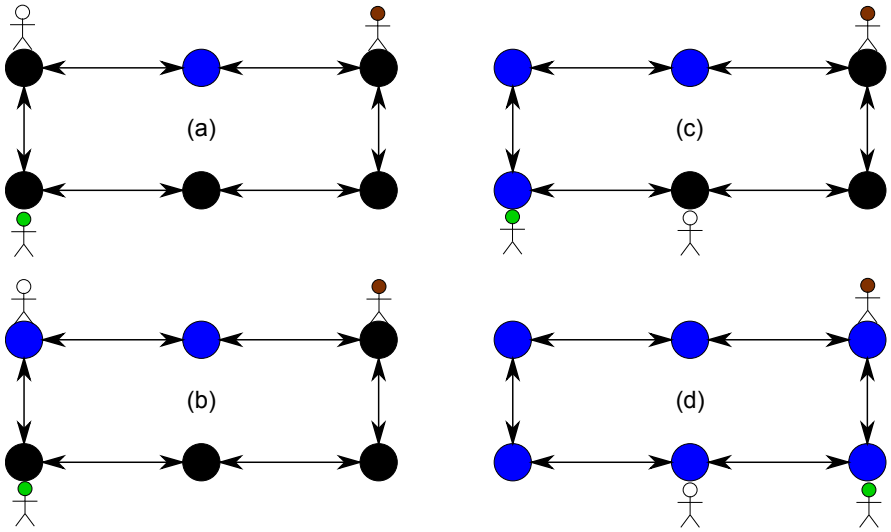


Figura 10: Estratégia de limpeza em um ciclo. A cor azul representa os conjuntos C_i em cada passo. Em (a), os agentes são postos no grafo e é possível limpar um vértice. Em (b), sem mover os agentes, é possível limpar um segundo vértice. Em (c), o agente "branco" é movido e mais um vértice é limpo. Em (d), o agente "verde" é movido e todos os vértices restantes podem ser limpos.

sos motivos, como, por exemplo, o aparecimento de uma nova conexão, a necessidade de desligamento de um nó da rede para manutenção, a falha de um nó ou de um ligamento da rede, para minimizar o custo de manutenção da rede ou, até mesmo, para melhorar a sua eficiência [6, 52, 53]. Para realizar esse re-roteamento, na maioria dos casos, é necessário interromper algumas conexões, pois uma ligação não pode ser compartilhada por mais de uma conexão. A Limpeza de Vizinhança pode ser usada para resolver a ordem de interrupções dessas conexões de forma minimizar o maior número de interrupções simultâneas. Para isso, seja $G = (V, E)$ um grafo orientado construído da seguinte forma: para cada conexão existente, adicione um vértice em $V(G)$ correspondente; se uma conexão u no seu roteamento final utiliza pelo menos um ligamento de uma conexão v no seu roteamento inicial, então adicione a aresta $u \to v$ a $E(G)$. O grafo direcionado G é chamado de *grafo de dependência* [6], pois se existe a aresta $u \to v$ então é necessário o re-roteamento ou a interrupção de v para que seja possível o re-roteamento de u, ou seja, u depende de v. Assumindo-se que o roteamento final das conexões é conhecido, resolver o problema de reconfiguração de roteamento de conexões de uma rede minimizando o maior número de interrupções simultâneas é equivalente a encontrar $l(G)$, sendo G o grafo de dependência relacionado. A Figura 11 mostra uma rede WDM, suas conexões e o grafo de dependência obtido. É importante notar que a existência de um ciclo no grafo implica em $l(G) \geq 1$. Assim, colocar somente um agente no vértice d é uma estratégia ótima de limpeza para

para o grafo de dependência da Figura 11.

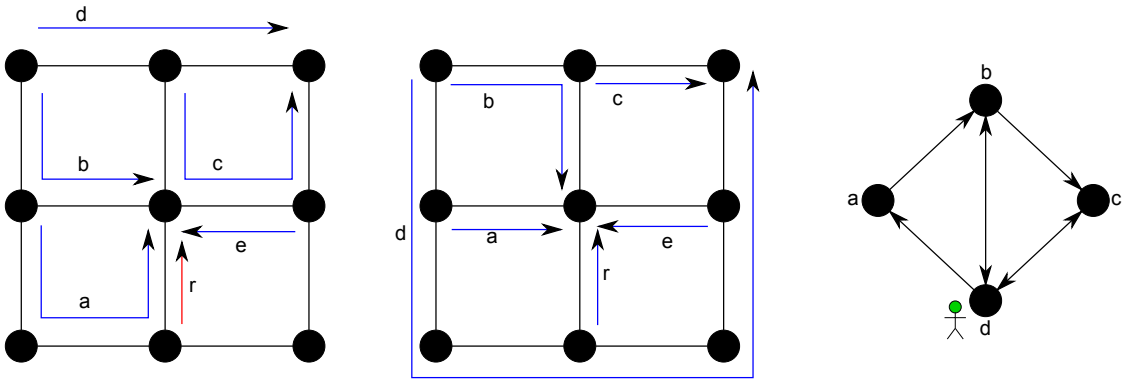


Figura 11: A rede e suas conexões à esquerda a conexão "r" não pode ser atendida. A rede e suas conexões no estado final ao centro. O grafo de dependência obtido à direita.

Inicialmente, esse problema foi estudado por N. Jose e A.K. Somani [6] que propuseram um algoritmo heurístico para minimizar o numero total de conexões interrompidas, que é equivalente a minimizar o número total de vértices ocupados por agentes. O problema de decisão associado a Limpeza de Vizinhança é NP-completo [7, 8], mesmo quando restrito a grafos orientados simétricos. Sabe-se que se G é um grafo orientado simétrico e G' seu grafo subjacente, então $LC(G') \leq l(G) \leq LC(G') + 1$ [7, 8]. É fácil mostrar que $l(G) \leq LC(G') + 1$, pois é possível traduzir uma uma decomposição em caminho de G', $P = \langle X_0, \ldots, X_s \rangle$, em uma estratégia de limpeza, $S = \langle (A_0, C_0), (A_1, C_1), \ldots, (A_s, C_s) \rangle$, para G fazendo $A_i = X_i$ para todo $0 \leq i \leq s$. Assim, a estratégia de limpeza obtida utiliza tantos agentes quanto o tamanho da maior sacola da decomposição. Os grafos completos são exemplos de grafos nos quais $LC(G') = l(G)$, pois qualquer decomposição em caminho possui pelo menos uma sacola com todos os vértices do grafo e é possível construir uma estratégia de limpeza a qual utiliza $LC(G')$ agentes, já que basta ocupar todos os vértices do grafo com exceção de um vértice qualquer. No primeiro passo esse vértice não ocupado pode ser limpo, pois todos os seus vizinhos estão ocupados, e no segundo passo o restante dos vértices podem ser limpos. Logo, o número de agentes necessários para que exista uma estratégia de limpeza e para que exista uma estratégia de procura em vértices rápida difere de no máximo uma unidade.

Uma observação importante é a relação entre o problema de encontrar a menor transversal de ciclos de um grafo orientado com o Limpeza de Vizinhança. Primeiramente, para qualquer DAG, G, $l(G)$ é zero. Como é possível limpar os sumidouros de G sem utilizar agentes, e eliminando esses vértices, o que resta é um subgrafo de G, que por sua vez também é um DAG. Aplicando-se esse raciocínio recursivamente, pode-se limpar

todos os vértices do grafo. Portanto, para um grafo qualquer, se seu menor transversal de ciclos é conhecido, então pode-se limpá-lo ao colocar um agente em cada vértice desse conjunto. De fato, o grafo induzido pelos vértices que restarem é um DAG, e, portanto, é possível limpá-lo sem a necessidade de agentes adicionais. Uma vez limpos, basta limpar os vértices ocupados pelos agentes. Desse modo, tem-se, naturalmente, um limite superior para o valor de $l(G)$. Esse limite, porém, pode ser arbitrariamente distante de $l(G)$, como mostra a Figura 12. A menor transversal de ciclos é tão grande quanto o número de vértices do ciclo interno da figura. No entanto, existem estratégias de limpeza que utilizam somente dois agentes, independentemente do tamanho desse ciclo. Infelizmente, o problema de decisão relacionado a encontrar a menor transversal de ciclos de um grafo é um problema NP-difícil [54].

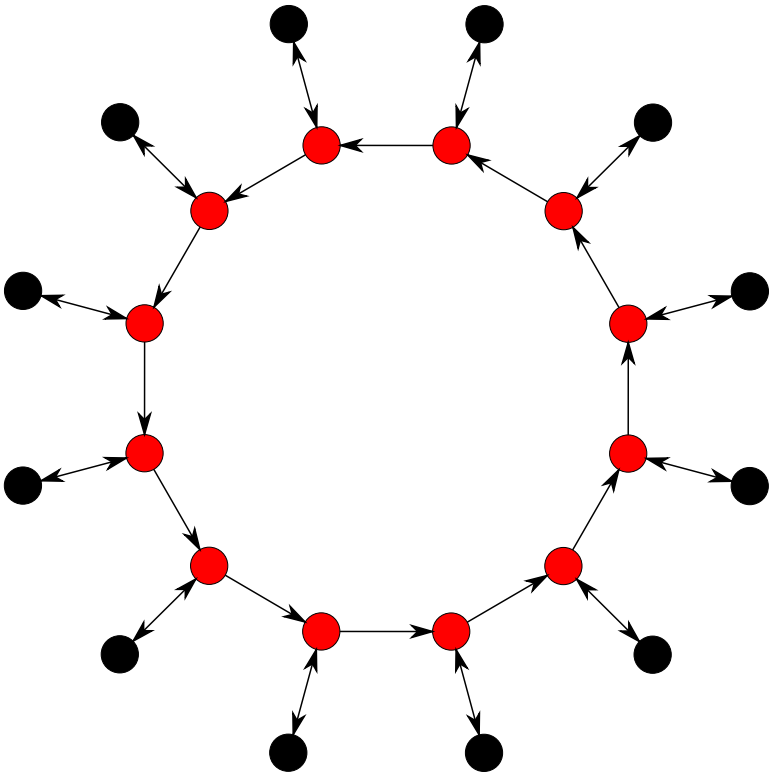


Figura 12: Todos os vértices em vermelho fazem parte de uma transversal de ciclos mínima.

6 Procuras Rápidas

Neste capítulo, serão apresentados os problemas de Procura em Grafos levando em consideração o número de passos de uma estratégia de procura, além de resultados na literatura relacionando o número de passos com o número de agentes de uma estratégia de procura. Por último, serão apresentados parâmetros para relacionar o número de agentes de uma estratégia de limpeza de vizinhança com o número de passos dessa estratégia.

6.1 Procura em Grafos Rápida

As definições para Procura em Vértice, Procura em Arestas e Procura Mista apresentadas no Capítulo 3 não levam em consideração a possibilidade de minimizar o número de passos dessas procuras, pois nesses casos o objetivo é apenas minimizar o número de agentes necessários para limpar o grafo.

Em um contexto no qual se deseja medir ou minimizar o número de passos de uma limpeza, é preciso fazer algumas alterações nessas definições para torná-las mais eficientes quanto ao número de passos utilizados. Uma estratégia de procura em arestas, por exemplo, possui pelo menos tantos passos quantas arestas possui o grafo em que ocorre a procura, pois para limpar uma aresta é necessária uma operação "atravessar" e essa operação ocupa um passo da procura limpando apenas uma aresta. Do ponto de vista da aplicação que motivou esses problemas, essa quantidade de passos que se leva para limpar o grafo não é realista, pois, no caso da busca nas cavernas, vários exploradores podem se movimentar ao mesmo tempo.

Por outro lado, em algumas aplicações, a rapidez da procura está associada a Limpeza de Vizinhança.

Outra motivação para levar em consideração o número de passos de uma limpeza são as aplicações em que o número de agentes disponíveis é superior ao necessário ou quando é importante que a busca acabe rapidamente. As próprias buscas por fugitivos são exemplos

de aplicações nas quais a rapidez é um fator imprescindível (homem-bomba) ou o número de agentes disponíveis pode ser superior ao necessário para efetuar a busca.

Dessa forma, há interesse, neste trabalho, de responder às seguintes perguntas:

A: Dado um número k e um grafo G, **qual o menor número de passos** necessários para limpar um grafo G com uma estratégia de procura que utilize k agentes, nos problemas da Procura em Grafos e da Limpeza de Vizinhança?

B: Dado um número s e um grafo G, **qual o menor número de agentes** necessários para limpar um grafo G em s passos, no máximo, para os problemas de Procura em Grafos e para a Limpeza de Vizinhança?

6.1.1 Procura em Arestas

Para tornar as estratégias de procura em arestas mais eficientes, deve ser considerada a utilização de operações "atravessar" simultâneas. A definição deste tipo de estratégia demanda algumas considerações. Por exemplo, as Figuras 13 e 14 mostram duas maneiras diferentes de interpretar a limpeza de um caminho utilizando operações simultâneas de "atravessar". O caminho está limpo ao final de um passo na Figura 13, porém isso não acontece na Figura 14, já que o fugitivo pode se "esconder" entre um agente e outro enquanto eles "atravessam" as arestas, produzindo uma recontaminação das arestas do caminho após a limpeza pelos agentes.

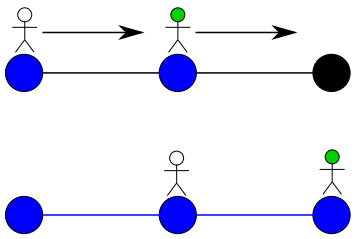


Figura 13: Estratégia de procura em arestas para limpar um caminho com apenas um passo. A cor azul representa a parte limpa do grafo e a cor preta a parte contaminada.

O problema de Procura em Arestas, quando as estratégias de procura estão restritas a um passo, utilizando a interpretação da Figura 14 para as operações simultâneas de "atravessar", foi estudado em [55–58]. É importante notar que mesmo com essa restrição ainda é NP-completo decidir se k agentes são suficientes para que exista uma estratégia de procura em arestas com apenas um passo para um grafo G qualquer. Observa-se que o número de arestas do grafo é um limite inferior para k, pois todas as arestas devem

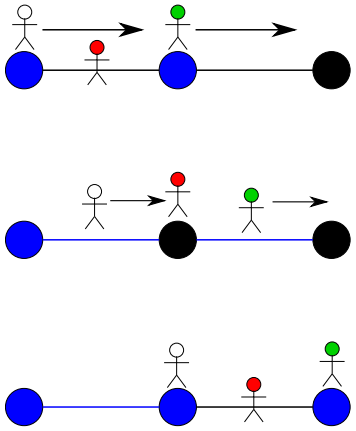


Figura 14: Estratégia de procura em arestas para limpar um caminho com apenas um passo utilizando uma interpretação diferente para operações atravessar simultâneas. A cor azul representa a parte limpa do grafo, a cor preta a parte contaminada e a vermelha representa o fugitivo. O fugitivo é capaz de "seguir" os agentes enquanto eles "atravessam" arestas.

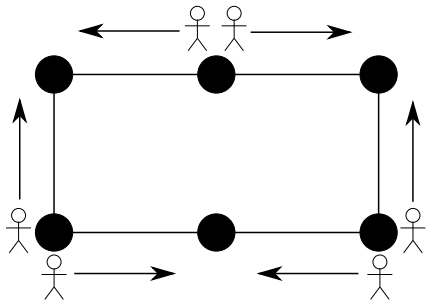


Figura 15: Estratégia de procura em arestas para limpar um ciclo com um número par de vértices com apenas um passo.

ser limpas em apenas um passo. A dificuldade desse problema está, principalmente, no fato de cada aresta precisar de um agente para ser limpa e alguns vértices precisarem de um agente estacionário para evitar a recontaminação. Usando a interpretação da Figura 14 para operações simultâneas de "atravessar", ilustra-se a dificuldade do problema da procura em arestas em um único passo na Figura 15 e na Figura 16. A Figura 15 mostra uma estratégia para limpar um ciclo com um número par de vértices e a Figura 16, uma estratégia para limpar um ciclo ímpar. Observa-se que um ciclo par pode ser limpo em um passo utilizando tantos agentes quantos forem os vértices do grafo, porém isso não acontece com ciclos ímpares, pois é necessário um agente extra para evitar a recontaminação.

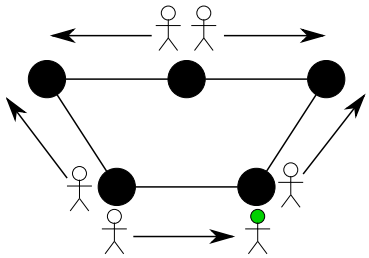


Figura 16: Estratégia de procura em arestas para limpar um ciclo com um número par de vértices com apenas um passo. A cor verde representa um agente que não se movimenta durante a estratégia de procura.

6.1.2 Procura em Vértices

No caso da Procura em Vértices, pequenas alterações devem ser feitas na definição, para tornár a procura mais eficiente quanto ao número de passos utilizados. É importante considerar a adição e a remoção de agentes de forma simultânea para que a decomposição em caminho associada à uma estratégia de procura tenha o mesmo número de sacolas que a estratégia tem de passos. A primeira alteração é a não exigência de que $A_i \subseteq A_{i+1}$ ou de que $A_{i+1} \subseteq A_i$. Dessa forma, agentes podem ser retirados e adicionados em um mesmo passo. A segunda se refere à forma de limpar o grafo, pois somente os agentes que não foram movimentados entre um passo e outro são capazes de bloquear o caminho do fugitivo. Mais formalmente, a definição de *estratégia de procura em vértices rápida* considerando operações de adição e remoção simultâneas é:

Definição 6.1. *Dado um grafo $G = (V, E)$, uma* estratégia de procura rápida*, ou simplesmente* procura rápida*, é uma sequência $S = \langle (A_0, C_0), (A_1, C_1), \ldots, (A_s, C_s) \rangle$. Cada (A_i, C_i) é chamado de* passo*, ou* etapa*, da procura. Os conjuntos A_i representam as posições dos policiais no passo i da procura e os conjuntos C_i os vértices livres do fugitivo no passo i da procura. Nessa sequência, tem-se que $C_0 = A_0$, $C_s = V(G)$ e para todo $i \in [1, s]$:*

$$C_i = C_{i-1} \backslash \{u \in C_{i-1} |\ \textit{existe } v \in \overline{C_{i-1}} \textit{ e caminho de } u \textit{ a } v \textit{ em } G\left[\overline{A_i \cap A_{i-1}}\right]\} \cup A_i.$$

A Definição 6.1 difere da Definição 3.1 também por C_i ser um conjunto de vértices na primeira e um subgrafo na segunda. No entanto, as duas definições são equivalentes, pois o número de procura em vértices de um grafo é o mesmo para as duas definições, já que é possível transformar uma estratégia de procura rápida em uma estratégia de procura em vértices com o mesmo número de agentes, como também é possível transformar uma

estratégia de procura em vértices em uma estratégia de procura rápida com o mesmo número de agentes.

É importante observar a relação do número de passos entre essas duas definições de estratégia de procura em vértices. A Figura 17 mostra um exemplo no qual esses números são diferentes. É trivial responder quantos agentes são necessários para que exista uma estratégia de Procura em Vértices com um passo apenas, diferentemente da Procura em Arestas, pois é necessário ocupar todos os vértices do grafo simultaneamente no primeiro passo. Outra observação importante é que, se existir uma decomposição em caminho de um grafo G com t sacolas, então existe uma procura com t passos para esse grafo, bastando fazer $A_i = X_i$ para cada sacola da decomposição.

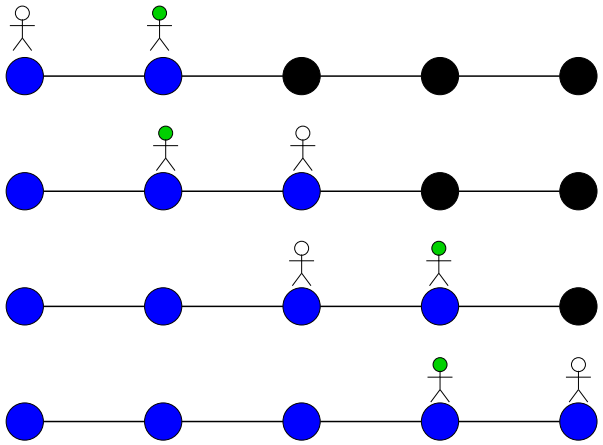


Figura 17: Exemplo de estratégia de procura em vértices rápida de um caminho. Nessa estratégia foram utilizados dois agentes e 4 passos.

Para que seja possível medir e relacionar o número de passos e o número de agentes necessários para limpar um grafo, alguns parâmetros são definidos a seguir.

O *tempo mínimo de procura* de um grafo G com k agentes, $fvs(G, k)$ (do inglês *fast vertex search*), é o menor valor t tal que existe uma estratégia de procura em vértices utilizando, no máximo, k agentes e t passos. Se $k < vs(G)$ não existe uma estratégia de procura rápida e, portanto, faz-se $fvs(G, k) = \infty$.

O parâmetro *número de Procura em Vértices com restrição de tempo*, $vs(G, t)$, é o menor número de agentes necessários para limpar o grafo com até t passos. Essa notação não se confunde com $vs(G)$, pois "t" define o número máximo de passos que uma estratégia de procura pode ter. O valor $vs(G, 0)$ não é definido pois não existe estratégia de procura rápida em zero passos. Observa-se que $vs(G, n) = vs(G)$, pois com $vs(G)$ agentes sempre é possível limpar pelo menos um vértice a cada passo.

Uma $(\leq p, \leq t)$-*estratégia de procura* é uma estratégia de procura rápida que utiliza no máximo p agentes e no máximo t passos.

Uma (p, t)-*estratégia de procura* é uma estratégia de procura rápida que utiliza exatamente p agentes e exatamente t passos.

Herrmann e Brandenburg mostraram, em 2006, que existem grafos em que o número de passos necessários para limpá-lo pode ser reduzido até a metade com a utilização de $vs(G) + 1$ agentes, e que existem grafos nos quais, mesmo utilizando $2vs(G) - 1$ agentes, ainda é necessário o mesmo número de passos de uma estratégia com $vs(G)$ agentes [59]. Sempre é possível limpar um grafo com um menor número de passos se $2vs(G)$ agentes forem utilizados, pois dada uma decomposição em caminho do grafo é possível limpar duas sacolas dessa decomposição a cada passo. Eles também mostraram que a Procura em Vértices, mesmo quando restrita a uma quantidade k de agentes e a uma quantidade t de passos, é monotônica e que é NP-completo decidir se existe uma estratégia de procura em vértices para um grafo G utilizando no máximo k agentes e t passos. Existem limites inferiores e superiores para $fvs(G, k)$ e $vs(G, t)$, como é mostrado no Teorema 6.1.

Teorema 6.1 (Herrmann, Brandenburg [59])**.** *Seja n o número de vértices de um grafo conexo G qualquer, $k \geq vs(G)$. As seguintes desigualdades são verdadeiras:*

$$\lceil \frac{n-k}{k-1} \rceil + 1 \leq fvs(G, k) \leq n + 1 - k$$

$$\lceil \frac{n-1}{t} \rceil + 1 \leq vs(G, t) \leq n$$

Prova: É fácil ver que $fvs(G, k) \leq n + 1 - k$, pois, no primeiro passo, k agentes podem ser colocados no grafo limpando k vértices e a cada passo seguinte no mínimo um vértice deve ser limpo, e, portanto, é impossível que uma estratégia de procura com k agentes precise de mais do que $n + 1 - k$ passos.

Da mesma forma, no primeiro passo k agentes podem ser colocados no grafo limpando k vértices e para cada passo subsequente no máximo $k-1$ agentes podem ser movidos, pois pelo menos um agente não pode se mover para evitar recontaminação. Logo, $\frac{n-k}{k-1} + 1 \leq fvs(G, k)$. Como $fvs(G, k)$ é um número inteiro pode-se usar $\lceil \frac{n-k}{k-1} \rceil + 1$ para obter uma melhor aproximação.

É fácil ver que $vs(G, t) \leq n$, pois n agentes são suficientes para limpar qualquer grafo em apenas um passo.

Considerando que pelo menos um agente deve permanecer estacionário a cada passo,

$1 + \lceil \frac{n-1}{t} \rceil$ agentes são suficientes para limpar G em t passos, pois, ao colocar o primeiro agente (escolhendo um agente qualquer arbitrariamente para ser "o primeiro", já que vários agentes são colocados simultaneamente no primeiro passo), sobram $n-1$ vértices para serem limpos em t passos. □

Como a Procura em Vértices, mesmo restrita a uma quantidade de passos t, é monotônica, então descobrir $fvs(G,k)$ é o mesmo que encontrar uma decomposição em caminho de largura no máximo $k-1$, pois cada sacola só pode ter no máximo k vértices, que minimize o número de sacolas. Dessa forma, fazendo o conjunto de vértices ocupados no passo i igual a sacola i da decomposição, tem-se uma estratégia de procura rápida que utiliza no máximo k agentes e é mínima no número de passos. De maneira semelhante, pode-se encontrar $vs(G,t)$ construindo uma decomposição em caminho com no máximo t sacolas que minimize a largura da decomposição.

6.2 Limpeza de Vizinhança

Na Limpeza de Vizinhança, não há necessidade de alterar a definição de estratégia de limpeza para torná-la mais eficiente quanto ao número de passos de uma estratégia, pois os conjuntos C_i são bem determinados e monotonicamente crescentes. Porém, é necessário definir parâmetros para medir e relacionar o número de passos e o número de agentes de uma estratégia de limpeza.

O *tempo mínimo de limpeza* de um grafo orientado G com k agentes, $fl(G,k)$, é o menor valor t tal que exista uma estratégia de limpeza utilizando, no máximo, k agentes e t passos. Da mesma forma que na Procura em Vértices Rápida, se $k < l(G)$ então $fl(G,k) = \infty$.

O *número de Limpeza com restrição de tempo*, $l(G,t)$, é o menor número de agentes necessários para limpar o grafo com até t passos. Da mesma forma que na Procura em Vértices Rápida, a notação $l(G,t)$ não se confunde com $l(G)$, pois "t" define o número máximo de passos que uma estratégia de procura pode ter e $l(G,n)$ é igual à $l(G)$.

Uma $(\leq p, \leq t)$-*estratégia de limpeza* é uma estratégia de limpeza que utiliza no máximo p agentes e no máximo t passos.

Uma (p,t)-*estratégia de limpeza* é uma estratégia de limpeza que utiliza exatamente p agentes e exatamente t passos.

Outra observação importante é que, na literatura, não existem resultados relativos ao

problema de encontrar $fl(G, k)$ ou $l(G, t)$, para um grafo G e inteiros k e t.

7 Limpeza de Vizinhança Rápida

Neste capítulo, são apresentados os resultados obtidos, neste trabalho, relativos ao problema de Limpeza de Vizinhança. Os resultados foram obtidos conjuntamente com David Couldert, Dorian Mazauric e Nicolas Nisse durante um estágio feito junto a Equipe Mascotte do INRIA Sophia Antipolis.

7.1 Complexidade

Sabe-se que decidir se existe uma $(\leq k, \leq t)$-estratégia de procura rápida em um grafo G é um problema NP-completo. E quanto ao problema de decidir se existe uma $(\leq k, \leq t)$-estratégia de limpeza em um grafo G? O teorema a seguir elucida essa questão:

Teorema 7.1. *Decidir se existe uma $(\leq k, \leq t)$-estratégia de limpeza em um grafo G é um problema NP-completo.*

Prova: O problema do empacotamento (Bin-Packing Problem) será utilizado na demonstração. Esse problema possui como entrada um conjunto de números inteiros $U = \{a_1, \ldots, a_n\}$, um número inteiro k, o qual define o tamanho máximo de cada pacote, e um inteiro t, que define qual o número de pacotes disponíveis. O problema consiste em descobrir se é possível particionar o conjunto U em até t conjuntos (pacotes) de forma que a soma dos inteiros em cada partição não exceda k. É importante ressaltar que esse problema é fortemente NP-completo, significando que ele é NP-completo mesmo quando restrito às instâncias nas quais os valores inteiros são limitados por um polinômio no tamanho da entrada.

Será feita uma redução do problema de empacotamento para o problema de decisão associado a uma $(\leq k, \leq t)$-estratégia de limpeza em um grafo G. Seja $I = (U, k, t)$ uma instância do problema de empacotamento. A construção do grafo G do problema de limpeza de vizinhança com restrição de tempo é feita a partir do conjunto de inteiros U, da seguinte forma: considerando um grafo orientado G vazio; adiciona-se a G um grafo

completo com a_i vértices, para cada $a_i \in U$; e, finalmente, para cada vértice v em $V(G)$ adicione o laço $v \to v$ em $E(G)$. A Figura 18 exemplifica essa construção. A criação desse grafo pode ser feita em tempo polinomial, pois supõe-se que a instância I é definida com valores inteiros limitados. A instância para o problema de limpeza com restrição de tempo é $I' = (G, k, t)$ na qual G é o grafo construído anteriormente, k é o número de agentes disponíveis e t é o número máximo de passos.

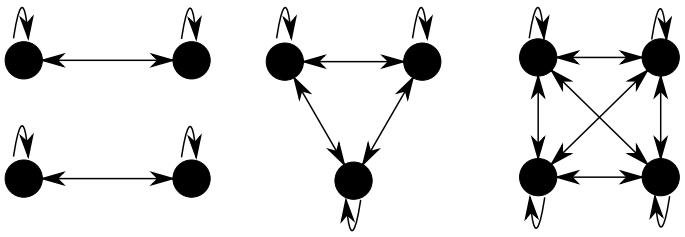


Figura 18: Grafo obtido a partir de $U = \{2, 2, 3, 4\}$.

Se, para a instância I do problema de empacotamento, a resposta é "sim", então há uma maneira de particionar U de forma que a soma dos inteiros de cada partição não exceda k e há no máximo t partições. Uma estratégia na qual, a cada passo, todos os vértices criados a partir dos inteiros de uma partição são ocupados é capaz de limpar uma partição por passo sem utilizar mais do que k agentes. Como existem no máximo t partições, essa estratégia também não utiliza mais do que t passos. Logo, existe uma $(\leq k, \leq t)$-estratégia de limpeza para G.

Supondo, por absurdo, que não existe uma forma de particionar os inteiros de U em até t partições sem uma das partições exceder o valor k, mas que há uma $(\leq k, \leq t)$-estratégia de limpeza, $S = \langle (A_0, C_0), \ldots, (A_{t-1}, C_{t-1}) \rangle$, para G. Seja X_i o conjunto dos vértices limpos no passo i da estratégia de limpeza, ou seja, $X_i = C_i \backslash C_{i-1}$ para todo $i > 0$ e $X_0 = C_0$. É importante notar que G é formado por cliques disjuntas nas quais cada vértice possui um laço, portanto a única maneira de limpar uma clique é ocupar todos os vértices dessa clique simultaneamente. Logo, $X_i \subseteq A_i$, $|X_i| \leq k$ e caso X_i contenha um vértice de uma clique, então X_i contem todos os vértices dessa clique. Observa-se que os subconjuntos X_i, que não são vazios, formam uma partição de $V(G)$, pois os vértices só podem ser limpos uma única vez. Seja X'_i o conjunto dos números inteiros os quais representam as cliques de $G[X_i]$, considerando apenas os índices i tais que $X_i \neq \emptyset$. $X'_0 \ldots X'_{t-1}$ é uma partição do conjunto U. Cada X'_i possui cardinalidade inferior ou igual a k e existem, no máximo, t conjuntos. Um absurdo, pois esse particionamento não deveria existir. □

Corolário 7.1. *Encontrar $fl(G,k)$ e encontrar $l(G,t)$ são problemas NP-difíceis.* □

Uma propriedade importante do problema de Limpeza de Vizinhança é o fato dele ser fechado para subgrafos, ou seja, se G' é um subgrafo de G então $l(G') \leq l(G)$. O Lema 7.1 mostra que essa propriedade é válida para ambos os parâmetros $l(G,t)$ e $fl(G,k)$.

Lema 7.1. *Dado um grafo direcionado G e um subgrafo, G', de G, tem-se $l(G',t) \leq l(G,t)$ e $fl(G',k) \leq fl(G,k)$.*

Prova: Se $fl(G,k) = t$ e $fl(G',k) > t$, então existe uma $(\leq k, t)$-estratégia de limpeza para G e não há uma $(\leq k, \leq t)$-estratégia de limpeza para G'. Seja $S_G = \langle (A_0, C_0) \dots (A_{t-1}, C_{t-1}) \rangle$ essa estratégia de limpeza para G e $S_{G'} = \langle (A'_0, C'_0) \dots (A'_{t'-1}, C'_{t'-1}) \rangle$ uma estratégia para G', obtida a partir de S_G, tal que $A'_i = A_i \cap V(G')$ e C'_i é obtido a partir de A'_i para todo $0 \leq v \leq t'-1$, pois, os conjuntos C'_i podem ser determinados a partir do grafo G' e dos conjuntos A'_i utilizando as regras "limpar" da definição de estratégia de limpeza.

Provar-se-á que $S_{G'}$ é uma estratégia de limpeza para G'. Para tanto, basta provar que $C'_i = C_i \cap V(G')$, para todo i. Essa prova será feita por indução no índice i. Para $i = 0$, observa-se que $C_0 \cap V(G') \subseteq C'_0$. Supondo, por hipótese de indução, que $C_j \cap V(G') \subseteq C'_j$ para $j < i$, chega-se a conclusão de que $C_i \cap V(G') \subseteq C'_i$ como será mostrado em seguida. Seja u um vértice de $C_i \cap V(G')$ tal que $u \notin C'_i$ e j o passo em que u ficou limpo em G, ou seja, não existe $j' < j$ tal que $u \in C_{j'}$. Se $j = i$, então $N^+_G(u) \subseteq C_{i-1} \cup A_i$, pela definição de estratégia de limpeza, e $N^+_{G'}(u) \subseteq C'_{i-1} \cup A'_i$, pela hipótese de indução, e portanto, $u \in C'_i$. No caso em que $j < i$, por indução, $N^+_G(u) \subseteq C_{j-1} \cup A_j$ e portanto $u \in C'_j$. Logo, se $t' = t$ então $C_{t-1} \cap V(G') = V(G') = C'_{t-1}$, e portanto $t' \leq t$. Um absurdo, pois $fl(G',k) > t$.

A prova de que o parâmetro $l(G,t)$ também possui essa propriedade é análoga à prova de $fl(G,k)$. □

7.2 Limpeza de Vizinhança com Agentes Fixos

Uma variação do problema da Limpeza de Vizinhança é quando se restringe a movimentação dos agentes, significando que $A_i \subseteq A_0$ para todo $i \in [0, t-1]$. Nessa variação, o importante é descobrir as posições dos agentes as quais minimizam o número de passos da limpeza. Após a introdução dos agentes, o procedimento de limpeza deve ocorrer automaticamente, ou seja, sem mudanças nas posições dos agentes. Um exemplo desse

automatismo é dado pela Figura 19. Observa-se que, nessa variação, o número de agentes disponível deve ser, necessariamente, pelo menos tão grande quanto o tamanho do menor transversal de ciclos.

Nesta seção, é apresentado um algoritmo para calcular uma estratégia de limpeza o menor número de passos quando o número de agentes é fixado e a sua a movimentação é restringida.

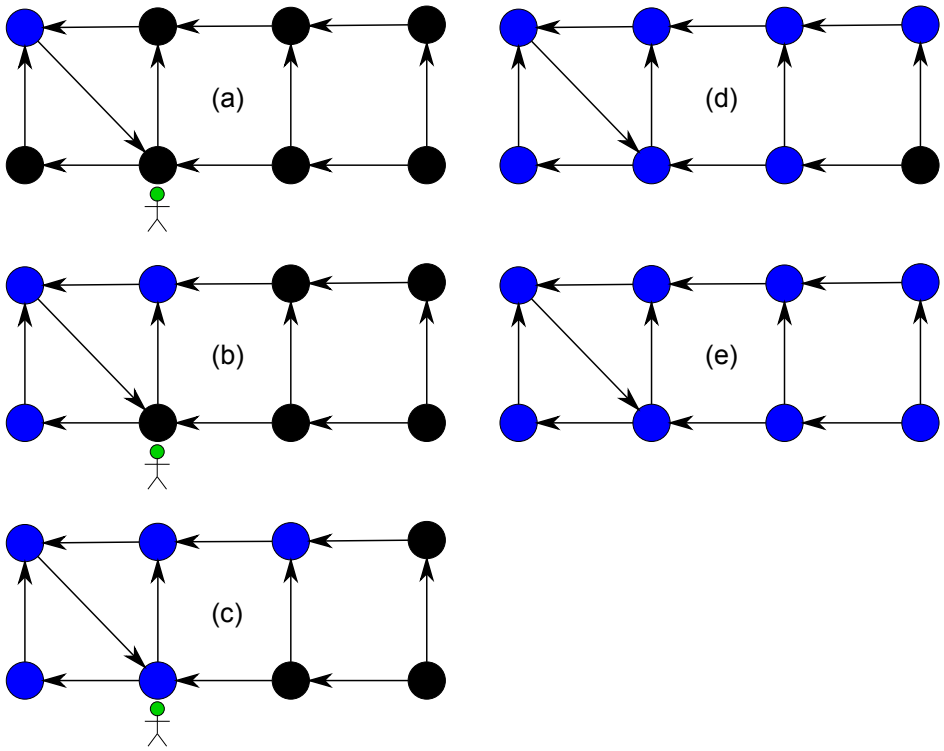


Figura 19: Após a colocação do agente no primeiro passo, (a), todos os vértices do grafo são limpos sem que seja necessária uma mudança de posição do agente. A cor azul representa os conjuntos C_i em cada passo.

No Algoritmo 1, A é o melhor conjunto de posições para a colocação dos agentes, x é uma variável auxiliar para guardar o tamanho do maior caminho após a ocupação dos vértices pelos agentes e X é o conjunto de arestas $u \to v$ tal que $v \in A$. O Algoritmo 1 testa todas as possíveis combinações para as posições dos agentes e calcula, para cada uma delas, quantos passos serão necessários para o término da limpeza. Após adicionar os agentes, se o grafo $D[E(D)\backslash X]$ contém um circuito, então é impossível realizar uma estratégia de limpeza sem movimentar os agentes, mas, caso contrário, o número de vértices do maior caminho do grafo $D[E(D)\backslash X]$ é exatamente o número de passos de uma estratégia de limpeza.

Teorema 7.2. *O Algoritmo 1 encontra uma estratégia que minimiza o número de passos*

Algoritmo 1: Digrafo com agentes fixos

Entrada: Um grafo orientado $D = (V, E)$ e uma constante k.
Saída: Encontra a melhor colocação para os k agentes.

```
1  x = ∞
2  A = ∅
3  para cada conjunto P ⊆ V(D) de tamanho k faça
4  |  X := {u → v ∈ E(D)|v ∈ P}
5  |  l := diâmetro de D[E(D)\X]
6  |  l := l + 1
7  |  se D[E(D)\X] possui um circuito então
8  |  |  l := ∞
9  |  fim
10 |  se x > l então
11 |  |  x := l
12 |  |  A := P
13 |  fim
14 fim
15 se x = ∞ então
16 |  retorna Não existe estratégia!
17 fim
18 retorna A
```

com k agentes para um digrafo $D = (V, E)$ considerando somente estratégias nas quais $A_i \subseteq A_0$ para todo $i \in [0, t-1]$.

Prova: Já que a reutilização dos agentes não é possível, se, após a colocação dos agentes, $D[E(D) - X]$ ainda possui um circuito, então o Algoritmo 1 descarta essa estratégia, pois faz $l = \infty$. Caso contrário, o número de passos que uma estratégia precisa para limpar o grafo D é exatamente uma unidade maior que o diâmetro do subgrafo obtido com a retirada das arestas as quais incidem nos vértices ocupados pelos agentes. O Algoritmo 1 compara todas as possíveis posições para os agentes, escolhendo aquela que minimiza o diâmetro de $D[E(D)\backslash X]$. Se é impossível limpar o grafo utilizando k agentes, então ao final da execução do Algoritmo 1 tem-se $x = \infty$ e, portanto, será retornado que não existe estratégia de limpeza. □

Lema 7.2. *A complexidade do Algoritmo 1 é limitada por $O\left(n^{k+2}\right)$*

Prova: O Algoritmo 1 necessita de $O(n^2)$ para cada iteração do laço **para**. É possível construir o conjunto X em $O(n)$. Para calcular o diâmetro do grafo resultante pode-se utilizar uma busca em profundidade, partindo dos sumidouros do grafo, percorrendo as arestas no sentido contrário. Tal procedimento pode ser feito em $O(n^2)$. Uma busca em profundidade partindo das fontes é suficiente para descobrir se existem circuitos no grafo

e, portanto, pode ser feita em $O(n^2)$. O número de iterações do laço é dado por $\begin{pmatrix} n \\ k \end{pmatrix}$. Como k é uma constante, tem-se que $\begin{pmatrix} n \\ k \end{pmatrix} = O\left(n^k\right)$. Logo, o tempo total do Algoritmo 1 é limitado pela expressão $O\left(n^k\right) * O\left(n^2\right)$ que é equivalente a $O\left(n^{k+2}\right)$. □

É uma questão em aberto se o problema de Limpeza de Vizinhança com Agentes Fixos pode ser resolvido em tempo polinomial quando o número de agentes não é fixo.

7.3 Cobertura e Limpeza de Vizinhança

Nesta seção, é feita uma comparação entre a Limpeza de Vizinhança e o problema de Cobertura por Vértices.

Um problema interessante é determinar o número de agentes mínimo quando o número de passos de uma estratégia é fixo. O caso $l(G, 1)$ é trivial. Todos os vértices que não são fontes devem ser ocupados por agentes para que seja possível limpar todo o grafo em apenas um passo. Já o caso $l(G, 2)$, para grafos simétricos, é limitado superiormente pela menor cobertura de vértices do grafo subjacente ao grafo G. Ocupando todos os vértices da cobertura com agentes, os vértices que não estão na cobertura podem ser limpos no primeiro passo, pois, no grafo subjacente, se um vértice v não está na cobertura então todos os seus vizinhos estão. Logo, no segundo passo todos os vértices, ainda contaminados, podem ser limpos. A Figura 20 mostra um exemplo desse tipo de estratégia. É importante ressaltar que existem estratégias de limpeza com dois passos as quais utilizam menos agentes do que o tamanho da menor cobertura de vértices do grafo, como é exemplificado na Figura 21.

7.4 Resultados para Classes de Grafos

Nessa seção, são apresentados os resultados obtidos para classes de grafos específicas.

7.4.1 Grafos direcionados sem circuitos (DAGs)

É importante lembrar que DAGs podem ser limpos sem a necessidade de agentes, porém uma limpeza de vizinhança de um DAG que não utilize agentes precisa de, necessariamente, tantos passos quantos vértices existirem no maior caminho direcionado

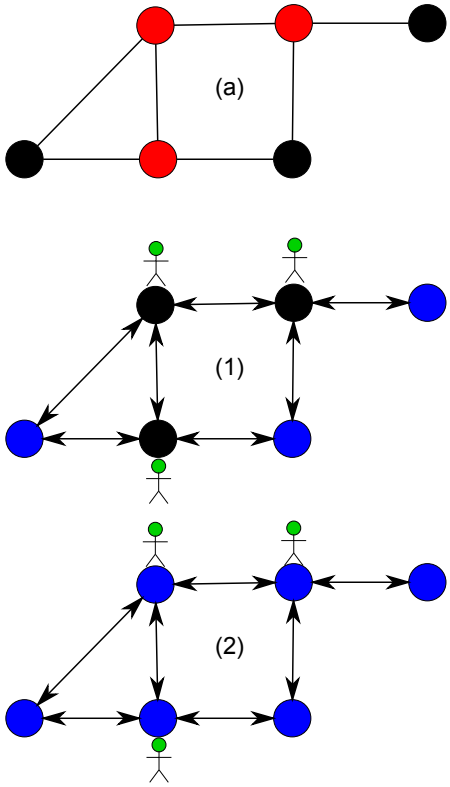


Figura 20: Em (a), os vértices pertencentes a menor cobertura do grafo subjacente estão com a cor vermelha. Em (1), são mostradas as posições dos agentes no primeiro passo e os vértices limpos em azul. Em (2), o restante dos vértices são limpos, pois todos os vértices os quais ainda não foram limpos estão ocupados por agentes.

desse grafo. A seguir, serão apresentados limites sobre o tempo de limpeza para grafos direcionados sem circuitos.

Proposição 7.1. *Dado um DAG $D = (V, E)$ com diâmetro $d - 1$, correspondendo ao maior caminho direcionado de D, então $fl(D, 0) = d$.*

Prova: No passo 0, uma estratégia de limpeza que não utiliza agentes pode limpar todos os sumidouros de D e, em um passo $i > 0$, todos os sumidouros do grafo induzido por $V \backslash C_{i-1}$ podem ser limpos. Portanto, no passo i, o diâmetro do grafo D considerando apenas vértices contaminados é dado pelo diâmetro do grafo $D[V \backslash C_i]$. Seja P um dos caminhos de D tal que P possui d vértices. Logo, ao final de i passos, tem-se i vértices de P limpos, pois a cada passo o último vértice de P que ainda está contaminado é limpo. Assim, a cada passo, o diâmetro do grafo induzido pelos vértices contaminados diminui de uma unidade e, portanto, ao final d passos, o diâmetro de $D[V \backslash C_{d-1}]$ é igual a zero, ou seja, todos os vértices de $D[V \backslash C_{d-1}]$ são sumidouros. Logo, ao final de d passos todos os vértices de D estão limpos. □

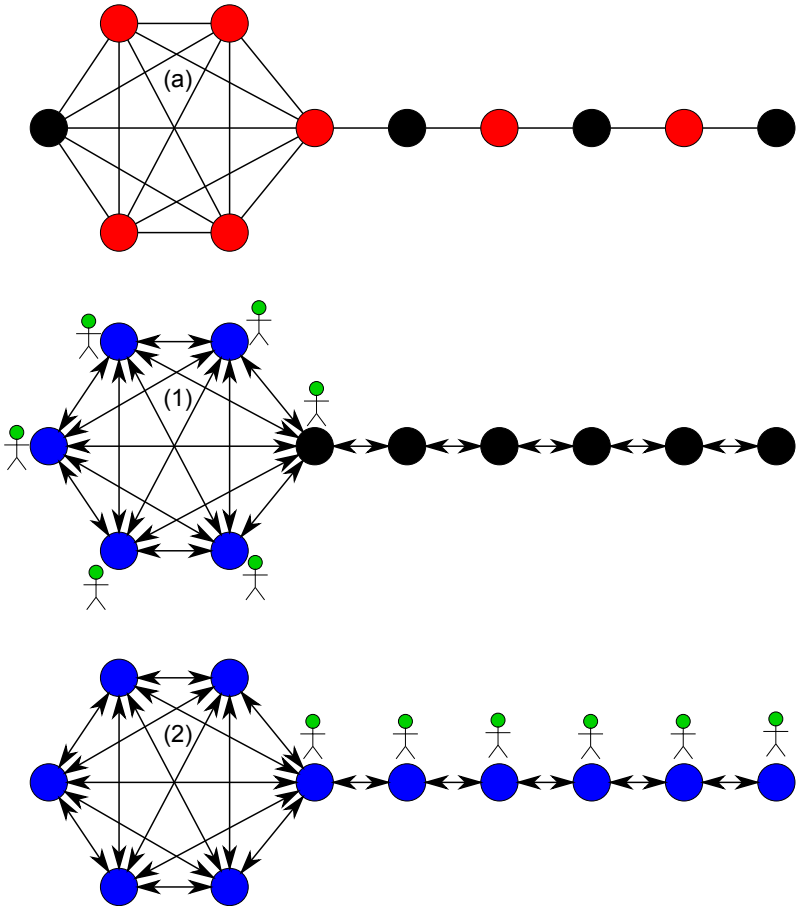


Figura 21: Em (a), os vértices que fazem parte de uma cobertura mínima do grafo estão com a cor vermelha. Em (1), é mostrado a posição dos agentes e os vértices limpos em azul. Em (2), todos os vértices do grafo estão limpos.

Lema 7.3. *Se $D = (V, E)$ é um DAG, então, para todo $k \geq 0$, $\lceil \frac{1}{(k+1)} fl(D, 0) \rceil \leq fl(D, k) \leq fl(D, 0)$.*

Prova: Seja $k \geq 0$. É fácil ver que $fl(D, k) \leq fl(D, 0)$. Seja $P = \langle v_0, \ldots, v_t \rangle$ o maior caminho direcionado de D e $D' = D[V(P)]$. Observa-se que $fl(D, 0) = fl(D', 0)$, pois, sem utilizar agentes, só é possível limpar um vértice de D' a cada passo. Ao limpar D' com k agentes, $k + 1$ vértices podem ser limpos a cada passo (com exceção, talvez, do último passo). Logo, $fl(D', k) = \lceil \frac{fl(D',0)}{k+1} \rceil$, e, pelo Lema 7.1, $fl(D', k) \leq fl(D, k)$, conclui-se que $fl(D, k) \geq \frac{fl(D,0)}{k+1}$. □

O Lema 7.3 mostra um limite inferior para quão rápida pode ser uma limpeza de vizinhança de um DAG, D, com relação ao maior caminho, d, dele. De fato, $fl(D, 0) = d$ pela Proposição 7.1. Logo, $\frac{d}{(k+1)} \leq fl(D, k)$.

No que segue, a definição de rótulo de um vértice será utilizada para indicar quantos

passos são necessários para que este vértice esteja limpo em uma estratégia de limpeza de vizinhança que não utiliza agentes.

Definição 7.1. *Seja $D = (V, E)$ um DAG. O rótulo, $r(u)$, de um vértice u é definido recursivamente como:*

$$r(u) = \begin{cases} 1 & , \text{ se } N^+(u) = 0 \\ \max_{v \in N^+(u)}\{r(v) + 1\} & , \text{ caso contrário} \end{cases}$$

Se o particionamento $P(V(D)) = \{X_1, \ldots, X_t\}$ é tal que cada X_i contém os vértices, $v \in V(G)$, com $r(v) = i$, então $P(V(D))$ é um particionamento regulado.

Observa-se que todo DAG possui um particionamento regulado do seu conjunto de vértices.

Definição 7.2. *Um* separador *de um DAG, $D = (V, E)$, com um particionamento regulado, $P(V(D)) = \{X_1, \ldots, X_t\}$ de D, é um vértice $u \in V(D)$, tal que se $u \in X_i$ então não existe aresta, $e = v \to w \in E(G)$, $w \neq u$, de forma que $v \in X_k$ e $w \in X_l$ com $l \leq i < k$ e $i < t$. A Figura 22 mostra um exemplo de um separador.*

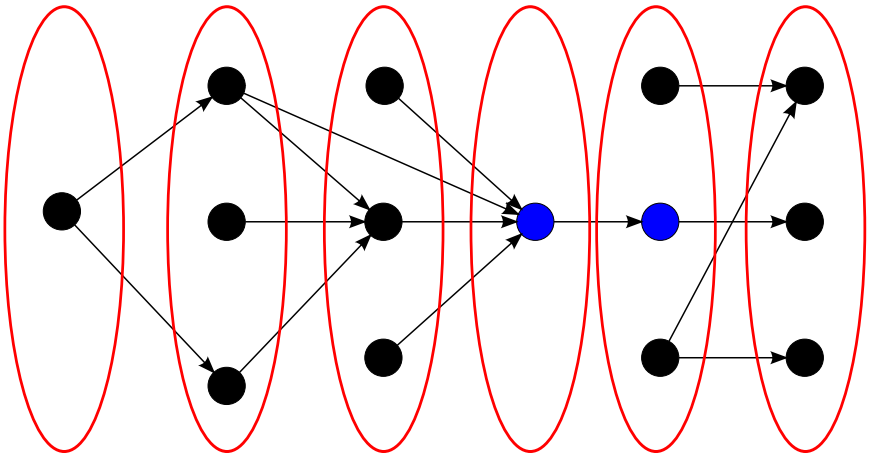


Figura 22: Exemplo de separadores para um DAG e um particionamento regulado. O particionamento está representado pelas elipses em vermelho e os separadores pela cor azul.

Definição 7.3. *$D[i, j]$, $i \leq j$, é o subgrafo de um DAG $D = (V, E)$ com particionamento regulado $P(V(D)) = \{X_1, \ldots, X_t\}$ induzido por $X_i \cup X_{i+1} \cup \ldots \cup X_j$.*

O Lema 7.4 a seguir mostra que é possível, para alguns DAGs, realizar uma estratégia de limpeza com a metade do número de passos de uma estratégia de limpeza com 0 agentes quando se passa a utilizar 1 agente.

Lema 7.4. *Seja $D = (V, E)$ um DAG, com pelo menos uma aresta, e $P(V) = \{X_1, ..., X_t\}$ um particionamento regulado de D. Se existe i, $1 \leq i \leq t/2$, com um separador $u \in X_i$ e $fl(D[2i+1, t], 1) = \frac{fl(D[2i+1,t],0)}{2}$, então $fl(D, 1) = fl(D, 0)/2$.*

Prova: Seja $D[1, 2i]$ o subgrafo induzido da Definição 7.3. Observa-se que $fl(D[1, 2i], 0) = 2i$, pois o diâmetro do grafo $D[1, 2i]$ é igual a $2i - 1$, e que é possível construir uma estratégia de limpeza de vizinhança para $D[1, 2i]$ com 1 agente utilizando i passos. No primeiro passo, o vértice separador $u \in X_i$ é ocupado pelo agente, assim, é possível limpar todos os vértices de X_1 e de X_{i+1}, nesse mesmo passo. No segundo passo, todos os vértices de X_2 e X_{i+2} podem ser limpos. Esse raciocínio pode ser mantido até o passo i no qual limpa-se os vértices de X_i e os vértices de X_{2i}, sendo possível mover o agente para outro vértice no passo $i + 1$, pois u está limpo no final do passo i. Seja S essa estratégia de limpeza para $D[1, 2i]$ e S' uma estratégia de limpeza com $\frac{fl(D[2i+1,t],0)}{2}$ passos para $D[2i + 1, t]$ que utiliza 1 agente. Concatenando as estratégias S e S' cria-se uma estratégia de limpeza para D que utiliza apenas um agente, pois, como foi argumentado anteriormente, o agente está livre ao final do passo i.

Notando que $fl(D, 0) = fl(D[1, 2i], 0) + fl(D[2i + 1, t], 0)$, pois D é um DAG e, portanto, o diâmetro de D é $t - 1$, e que, pela hipótese do lema, $fl(D[2i + 1, t], 1) = \frac{fl(D[2i+1,t],0)}{2}$, segue-se que $fl(D, 1) = fl(D[2i + 1, t], 1) + fl(D[1, 2i], 1) = fl(D, 0)/2$. □

Não é verdade que se $l(D, 1) = l(D, 0)/2$ então existe i, $1 \leq i \leq t/2$, de forma que existe um separador $u \in X_i$ e $l(D[2i + 1, t], 1) = \frac{l(D[2i+1,t],0)}{2}$. A Figura 7.4.1 mostra um contra exemplo para essa implicação.

7.4.2 Caminhos Direcionados Disjuntos

Nesta seção, são considerados grafos orientados que são a união de caminhos direcionados disjuntos. A Figura 24 mostra um exemplo dessa classe de grafo.

Pelo Lema 7.3 tem-se que a desigualdade $fl(G, k) \geq \frac{fl(D,0)}{k+1}$ é valida para um grafo direcionado G qualquer. Porém, se G é um caminho direcionado, então há um limite superior melhor, como mostra a Proposição 7.2.

Proposição 7.2. *Se $D = (V, E)$ é um caminho direcionado, então $fl(D, k) = \lceil \frac{fl(D,0)}{k+1} \rceil$.*

Prova: Seja S_k uma estratégia de limpeza, com k agentes, que coloca os agentes nos últimos vértices, contaminados e ainda não ocupados, do caminho direcionado de forma sucessiva. Observando que esses agentes estão livres para re-utilização no passo seguinte

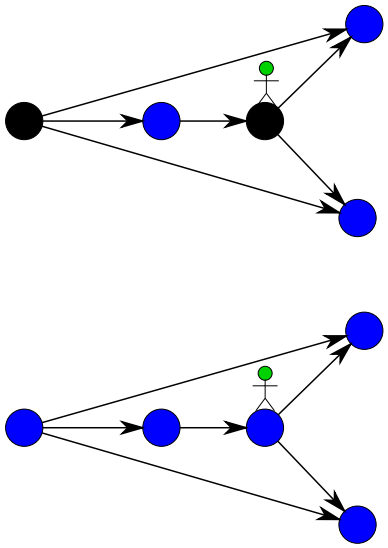


Figura 23: Um exemplo no qual, mesmo sem existir um separador no grafo, o número de passos necessários para limpar o grafo com um agente é igual a metade do necessário para limpá-lo sem a utilização de agentes.

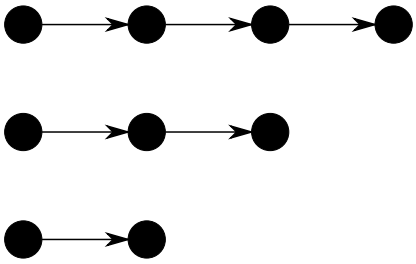


Figura 24: Exemplo de um grafo composto por caminhos disjuntos

e, para cada agente no grafo, é possível limpar um vértice extra a cada etapa, pois o grau de entrada de qualquer vértice é igual a um. Por outro lado, uma estratégia para D a qual não utiliza agentes limpa um vértice por passo. Portanto, é possível limpar D usando k agentes em $\lceil \frac{fl(D,0)}{k+1} \rceil$, pois a cada passo são limpos $k+1$ vértices (um vértice sumidouro e, para cada agente, um vértice extra).

Ao supor que existe uma estratégia S'_k que é capaz de limpar D em um menor número de passos do que S_k, conclui-se que S'_k é capaz de limpar um número maior do que $k+1$ vértices em algum passo. Um absurdo, pois o grafo é um caminho e o grau de entrada de qualquer vértice é no máximo 1. □

Definição 7.4. *Um grafo D_p é a união disjunta de p caminhos direcionados $L_1, \ldots, L_p$ com os respectivos número de vértices: $l_1, \ldots, l_p$, de forma que, para todo, $i \in \{1, \ldots, p-1\}$, $l_i \leq l_{i+1}$.*

O Lema 7.5 mostra qual o número máximo de vértices que podem ser limpos em um passo i de uma estratégia de limpeza. Para tanto, utiliza-se o fato de que um caminho direcionado com t vértices está, obrigatoriamente, limpo após t passos, e, consequentemente, após o passo $t \leq i$ evidentemente alguns caminhos já foram completamente limpos. Cada caminho possui um vértice sumidouro que é limpo a cada passo e cada agente pode limpar um vértice extra de um caminho. Logo, o número de vértices que pode ser limpo em um passo é dado pelo número de caminhos que ainda possuem vértices contaminados mais o número de agentes disponíveis para a limpeza.

Intuitivamente, o Lema 7.5 indica quantos vértices, no máximo, uma estratégia de limpeza é capaz de limpar a cada passo. Para isso o índice j que indica quantos caminhos orientados de D_p já estão limpos no passo i é utilizado. A ideia da prova se baseia no fato de sempre ser possível limpar um vértice de um caminho L_i de D_p caso L_i ainda não tenha sido completamente limpo, pois cada caminho é um DAG, então existem $p - j$ sumidouros em D_p no passo i.

Lema 7.5. *O número de vértices de um grafo D_p que podem ser limpos no passo i de uma $(\leq k, \leq t)$-estratégia de limpeza é dado pela seguinte função: seja j o maior número tal que $l_j < i \leq l_{j+1}$, caso $i \leq l_1$ então $j = 0$, e seja x o número de vértices do subgrafo induzido pelos vértices contaminados no passo i os quais não são sumidouros. Então, no máximo,* $\min\{k + p - j, x + p - j\}$ *vértices podem ser limpos no passo i.*

Prova: Observando que, no passo i, existem no máximo $p-j$ caminhos com algum vértice ainda contaminado independentemente da estratégia utilizada, então, como o grafo D_p é a união disjunta de caminhos e, por isso, pode-se limpar no máximo um vértice extra para cada agente no grafo, no passo i no máximo $p + k - j$ vértices podem ser limpos ao total (k, um para cada agente no grafo, $p - j$ de cada sumidouro). Por outro lado, se o número de vértices que não são sumidouros do subgrafo induzido pelos vértices contaminados no passo i é inferior ao número de agentes disponíveis, então no máximo $x + p - j$ vértices podem ser limpos nesse passo. Os agentes extra não são capazes de ajudar na limpeza. □

Usando a ideia do Lema 7.5, é possível criar um algoritmo que limpa, a cada passo, o maior número de vértices possível em um D_p. O Algoritmo 2 tenta minimizar o tamanho do maior caminho, considerando vértices contaminados e não ocupados.

Teorema 7.3. *Dado um D_p e um inteiro k, o Algoritmo 2 limpa* $\min\{k+p-j, x+p-j\}$ *vértices, a cada passo, i, com $l_j \leq i < l_{j+1}$.*

Prova: Primeiramente será demonstrado que os caminhos $L_{j+1}, \ldots, L_p$ não estão comple-

Algoritmo 2: Caminhos Disjuntos

Entrada: Um grafo direcionado D_p e um número de agentes k.
Saída: Calcula $fl(D_p, k)$ para D_p com k agentes.

```
1  t := 0                                                    // número de passos
2  A := ∅                                                    // vértices ocupados por agentes
3  C := ∅                                                    // vértices limpos
4  ∀i ∈ {1, ..., p}, x_i := l_i     // variáveis para guardar o número de vértices
   contaminados no caminho i
5  enquanto C ⊂ V faça
6      para cada agente faça
7          Seja D_i tal que i é o índice de max_{i∈[1,p]}{x_i}, escolher arbitrariamente se
           existir mais de uma possibilidade para i.
8          Encontrar v ∈ V[D_i] − (C ∪ A) e N⁻(v) ≠ ∅ e v é sumidouro de
           D_i − (A ∪ C)
9          Colocar um agente em v, A := A + {v}
10         x_i := x_i − 1
11     fim
12     para cada vértice v ∈ V − C faça
13         se N⁺(v) − (A ∪ C) = ∅ então
14             C := C + {v}
15             A := A − {v}
16             se v ∈ D_i então
17                 x_i := x_i − 1
18             fim
19         fim
20     fim
21     t := t + 1
22 fim
23 retorna t
```

tamente limpos no passo i.

Supondo que um dos caminhos L_y, $j+1 \leq y \leq p$ está completamente limpo em algum passo $i' < i$. Isso implica que, no passo i', o Algoritmo 2 ocupou pelo menos um vértice do caminho L_y. Seja $L_{y'}$ um caminho ainda não limpo no passo i. Tem-se que $|L_{y'} - (A_{i'} \cup C_{i'})| > |L_y - (A_{i'} \cup C_{i'})|$. Mas o algoritmo deveria ter ocupado um vértice de $L_{y'}$, já que ele escolhe o caminho de maior tamanho.

A cada passo o Algoritmo 2 utiliza todos os agentes disponíveis e existem $p-j$ caminhos com vértices ainda contaminados no passo i. Conclui-se, então, que esse algoritmo limpa $\min\{k-j+p, x+p-j\}$ vértices a cada passo. □

Teorema 7.4. *A complexidade do Algoritmo 2 para calcular $fl(D_p, k)$ é $O(n^3)$.*

Prova: O laço da instrução 5 é executado, no máximo, n vezes. O laço da instrução 6 é executado k vezes. Uma execução da instrução 7 pode ser realizada em $O(p)$ e como ela é executada até $n * k$ vezes, chega-se a conclusão que a linha 7 do Algoritmo 2 leva $O(n * k * p)$ ao total. Uma execução da instrução 8 pode ser realizada em $O(n)$ e, portanto, leva-se $O(n * k * n)$ para todas as execuções da instruçao 8. As instruções 9 e 10 podem ser realizadas em até $O(n * k)$. O laço da instrução 12 pode ser executado até n vezes. As instruções $13, 14, 15, 16, 17$ podem ser realizadas em um total de $O(n * n)$ passos. Finalmente, a instrução 21 leva $O(n)$ passos para ser realizada. Logo, o tempo de execução do Algoritmo 2 é dado por $O(n*k*p)+O(n*k*n)+O(n*k)+O(n*n)+O(n)$ e, portanto, ele leva até $O(n^3)$, pois $k \leq n$ e $p \leq n$. □

Pelos Teorema 7.3 e Teorema 7.4 conclui-se que calcular $fl(D_p, k)$ pode ser resolvido em tempo polinomial.

7.4.3 Subdivisão de Estrela Direcionada

Definição 7.5. *Uma* estrela direcionada*, $D = (V, E)$, é um grafo direcionado construído a partir de uma estrela $G = (V, E)$ na qual, para cada aresta $(u, v) \in V(G)$, tem-se $u \to v \in E(D)$ ou $v \to u \in E(D)$, mas não ambos.*

A subdivisão de uma aresta, $e = (u, v)$, é a substituição de e por e_1, e_2 e um vértice uv de forma que $e_1 = (u, uv)$ e $e_2 = (uv, v)$. No caso de uma aresta direcionada $e = u \to v$ a substituição é feita de forma que $e_1 = u \to uv$ e $e_2 = uv \to v$.

Definição 7.6. *Uma* subdivisão de estrela direcionada *é construída a partir de uma*

estrela direcionada, subdividindo as arestas da estrela de forma independente um número arbitrário e finito de vezes.

As figuras 25 e 26 mostram exemplos desses grafos.

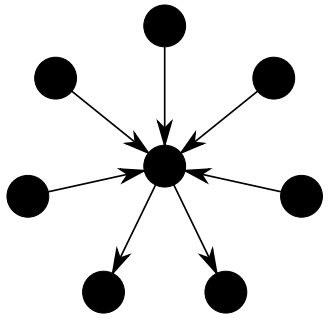


Figura 25: Exemplo de estrela direcionada

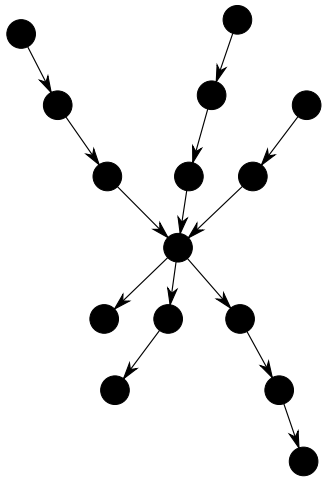


Figura 26: Exemplo de subdivisão de estrela direcionada

Sejam $I = \{I_1, \ldots, I_k\}$ o conjunto de caminhos que começam nas fontes da subdivisão e acabam nos vértices imediatamente anteriores ao centro da subdivisão de estrela e $O = \{O_1, \ldots, O_{k'}\}$ o conjunto de caminhos que começam imediatamente após o centro da estrela e terminam nos sumidouros da subdivisão de estrela.

Definição 7.7. *Seja*

$$I^i = \bigcup_{j=1}^{k} \{v \in I_j | v \text{ está à no mínimo distância } i \text{ do centro}\}$$

Definição 7.8. *Seja*

$$O^i = \bigcup_{j=1}^{k'} \{v \in O_j | v \text{ está à no mínimo distância } i \text{ do centro}\}$$

Por convenção, se $i \leq 1$ então $O^i = O^1$ e $I^i = I^1$. De forma similar, se $i > k$ então $I^i = \emptyset$ e se $i > k'$ então $O^i = \emptyset$.

Conjectura 7.1. *Se G é uma subdivisão de estrela direcionada, com pelo menos um caminho direcionado para o centro da estrela e um caminho direcionado que parte do centro da estrela, e $x = \max\{k, k'\}$ então $\min_{i=0}^{x}\{fl(G[I^{k'-i}], 1)+fl(G[O^{k'-i}], 1)+k'-i\} \leq fl(G, 1) \leq \min_{i=0}^{x}\{fl(G[I^{k'-i}], 1) + fl(G[O^{k'-i}], 1) + k' - i + 1\}$.*

A Conjectura 7.1 se baseia no fato de que com apenas um agente é possível limpar parte dos vértices de O. Em seguida, é possível mover o agente para o centro da estrela. Após a liberação do agente, pois ele só pode se mover quando todos os vértices restantes em O forem limpos, é possível limpar o restante dos vértices de I. Já que, enquanto o agente ocupa o vértice central, alguns vértices de I são limpos.

7.4.4 Árvores Direcionadas

Definição 7.9. *Uma* árvore direcionada *D é um DAG no qual existe no máximo um caminho direcionado entre qualquer par de vértices e o grafo subjacente de D é uma árvore.*

Definição 7.10. *Uma árvore direcionada T, enraizada em um vértice r, na qual todos os vértices $v \neq r$ têm $d^-(v) = 1$, é chamada de* árvore top-down.

Definição 7.11. *Uma árvore direcionada T, enraizada em um vértice r, na qual todos os vértices $v \neq r$ têm $d^+(v) = 1$, é chamada de* árvore bottom-up.

Uma árvore direcionada é completa se o grafo subjacente dessa árvore é uma árvore completa.

Definição 7.12. *Se T é uma árvore top-down na qual cada vértice $v \in V(T)$ tem $d^+(v) = k$ ou $d^+(v) = 0$, então diz-se que T é uma árvore k-ária top-down.*

Denota-se por T_k^h uma árvore k-ária top-down completa de altura h. Um exemplo de árvore k-ária top-down completa pode ser visto na Figura 27.

Lema 7.6. *Se T é uma árvore top-down com altura h e T_{k+1}^h é um subgrafo de T, então $fl(T, k) \geq h$.*

Prova: Assume-se que T_{k+1}^{h-i} é o subgrafo de T_{k+1}^h considerando apenas os vértices contaminados no passo i de uma (k, t)-estratégia de limpeza qualquer. Observando que todos

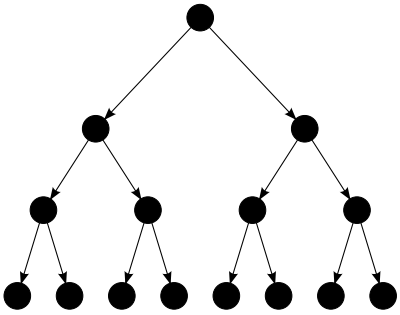


Figura 27: Exemplo de uma T_2^3.

os vértices internos de T_{k+1}^h têm grau de saída pelo menos $k+1$, não é possível limpar qualquer nó interno utilizando apenas k agentes. Supondo, por indução, que até o passo $i-1$ nenhum vértice interno pode ser limpo com a ajuda de k agentes, chega-se a conclusão de que T_{k+1}^{h-i}, o subgrafo induzido pelos vértices ainda contaminados, é uma árvore $k+1$-aria top-down completa com altura $h-i$. Como o grau de saída de qualquer vértice interno de T_{k+1}^{h-i} é pelo menos $k+1$, não é possível limpar qualquer um deles no passo i. Logo, para limpar T_{k+1}^h com até k agentes, são necessário h passos, pois a cada passo somente as folhas da árvore induzida pelos vértices ainda contaminados podem ser limpas. □

Para o lema a seguir, seja T_i o subgrafo de T considerando apenas os vértices contaminados no passo i.

Conjectura 7.2. *Seja T uma árvore top-down. Existe uma $(1,t)$-estratégia de limpeza, S_1, para T mínima no número de passos t tal que, para cada passo i, o agente ocupa apenas um desses dois tipos de vértices (v):*

1. *$d^+(v) = 0$, $d^+(u) = 1$ e $u \to v \in E(T_i)$; ou*
2. *$d^+(v) > 1$, $d^+(u) = 1$ e $u \to v \in E(T_i)$.*

Os parâmetros $d^+(v)$ e $d^+(u)$ são os graus de saída dos vértices v e u considerando apenas os vértices de T_i.

7.5 Modelos Lineares Inteiros

Nesta seção, dois modelos lineares para o problema de Limpeza de Vizinhança são apresentados. Não foram encontrados na literatura modelos para esse problema.

Em ambos modelos apresentados, as variáveis binárias x_{ij} representam a ocupação do vértice i no passo j por um agente, valor 1 indicando que o vértice i está ocupado por um agente no passo j. As variáveis binárias v_{ij} representam a limpeza do vértice i no passo j, o valor 1 indicando que o vértice está limpo no passo j.

As equações 7.3 e 7.12 indicam que ao final da limpeza todos os vértices devem estar limpos. As equações 7.4 e 7.13 indicam que no início da limpeza todos os vértices estão contaminados. Um vértice só pode ser limpo se cada vizinho está limpo ou ocupado por um agente como mostram as equações 7.5 e 7.14. Um agente só pode deixar o vértice que ele ocupa quando esse vértice é limpo (7.6 e 7.15). As equações 7.7 e 7.16 indicam que um vértice limpo no passo j se mantem limpo no próximo passo.

No modelo da Figura 28, a variável X representa o número máximo de agentes utilizados simultaneamente em um passo.

No modelo da Figura 29, as variáveis y_j indicam se no passo j algum vértice foi processado. A variável Y representa o número de passos utilizados. As equações 7.17 e 7.18 são usadas para calcular Y. As equações 7.11 restringem o número de agentes utilizados para no máximo k.

Figura 28: Minimizar número de agentes em t passos.

$$\min X \tag{7.1}$$

subject to:

$$\sum_{i \in V} x_{ij} \leq X \qquad \forall j \in [0..t] \tag{7.2}$$

$$v_{it} = 1 \qquad \forall i \in V \tag{7.3}$$

$$v_{i0} = 0 \qquad \forall i \in V \tag{7.4}$$

$$v_{ij} \leq v_{u(j-1)} + x_{uj} \qquad \forall i \to u \in E, j \in [1..t] \tag{7.5}$$

$$x_{ij} \geq x_{i(j-1)} - v_{i(j-1)} \qquad \forall i \in V, j \in [1..t] \tag{7.6}$$

$$v_{ij} \geq v_{i(j-1)} \qquad \forall i \in V, j \in [1..t] \tag{7.7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in V, j \in [1..t] \tag{7.8}$$

$$v_{ij} \in \{0, 1\} \qquad \forall i \in V, j \in [1..t] \tag{7.9}$$

Figura 29: Minimizar número de passos com k agentes.

$$\min Y \tag{7.10}$$

subject to:

$$\sum_{i \in V} x_{ij} \leq k \qquad \forall j \in [0..n] \tag{7.11}$$

$$v_{in} = 1 \qquad \forall i \in V \tag{7.12}$$

$$v_{i0} = 0 \qquad \forall i \in V \tag{7.13}$$

$$v_{ij} \leq v_{u(j-1)} + x_{uj} \qquad \forall i \to u \in E, j \in [1..n] \tag{7.14}$$

$$x_{ij} \geq x_{i(j-1)} - v_{i(j-1)} \qquad \forall i \in V, j \in [1..n] \tag{7.15}$$

$$v_{ij} \geq v_{i(j-1)} \qquad \forall i \in V, j \in [1..n] \tag{7.16}$$

$$\sum_{i \in V} (v_{ij} - v_{i(j-1)})/n \leq y_j \qquad \forall j \in [1..n] \tag{7.17}$$

$$j * y_j \leq Y \qquad \forall j \in [0..n] \tag{7.18}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in V, j \in [0..n] \tag{7.19}$$

$$v_{ij} \in \{0, 1\} \qquad \forall i \in V, j \in [0..n] \tag{7.20}$$

8 Procura em Vértices Rápida

Neste capítulo são apresentados os resultados obtidos, neste trabalho, relativos ao problema de Procura em Vértices Rápida.

8.1 Resultados Gerais

A propriedade do fechamento para subgrafos dos parâmetros $vs(G,t)$ e $fvs(G,k)$ também é verificada para o problema da Procura em Vértices Rápida como mostra o Lema 8.1.

Lema 8.1. *Dado um grafo G e um subgrafo, G', de G, tem-se $vs(G',t) \leq vs(G,t)$ e $fvs(G',k) \leq fvs(G,k)$.*

Prova: Similar à demonstração do Lema 7.1. □

Uma ideia que surge naturalmente, quando é interessante minimizar o número de passos de uma estratégia de procura, é tentar minimizar o número médio de agentes que não se movimentam entre um passo e outro da procura, ou seja, tentar minimizar $\sum_{i\in[0,t-1]} |A_i \cap A_{i+1}|/t$ na qual t é o número de passos dessa procura, pois um agente estacionário não contribui para a limpeza do grafo. Uma simplificação é simplesmente minimizar a soma: $\sum_{i\in[0,t-1]} |A_i \cap A_{i+1}|$. Como a Procura em Vértices é monotônica, mesmo quando há restrição no número de passos, é possível associar uma decomposição em caminho P à uma estratégia de procura rápida monotônica S da mesma forma que é feito para a versão clássica da procura em vértices.

Seja $P = \langle X_0, \ldots, X_{t-1}\rangle$ uma decomposição em caminho de um grafo G com $LC_G(P) = k-1$ mínima no número de sacolas. $P' = \langle X'_0, \ldots, X'_{t'-1}\rangle$ uma decomposição em caminho com $LC_G(P) = k-1$ que minimiza a soma $\sum_{i=1}^{t'-1} |X_{i-1} \cap X_i|$. Seguindo o raciocínio anterior, é natural concluir que $t' = t$, entretanto a Figura 30 mostra um contra-exemplo para essa afirmação.

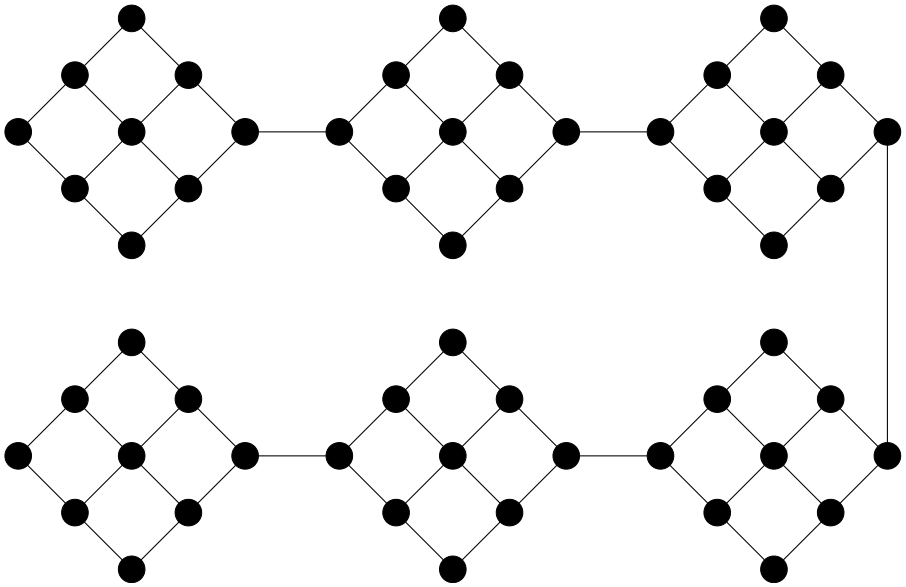


Figura 30: Neste grafo é possível realizar uma procura com 9 agentes em 6 passos, mas uma decomposição em caminho com largura 8 que minimize a soma $\sum_{i=1}^{t'-1} |X_{i-1} \cap X_i|$ precisa de 11 passos.

8.2 Caminhos

Teorema 8.1. *Se G é um caminho com p vértices, P_p, então $fvs(G,k) = \lceil \frac{p-k}{k-1} + 1 \rceil$.*

Prova: Notando que $\lceil \frac{p-k}{k-1} + 1 \rceil$ é um limite inferior para $fvs(G,k)$, conclui-se que para provar esse teorema basta construir uma estratégia de procura rápida com esse número de passos.

Seja $P = \langle v_1, v_2, \ldots, v_p \rangle$ os vértices do caminho P_p ordenados a partir de um vértice de grau 1, ou seja, $(v_i, v_{i+1}) \in E(P)$. Seja S a seguinte estratégia de procura utilizando k agentes. No passo 1, ocupam-se os vértices v_1 à v_k. No passo $i > 1$, removem-se os agentes dos vértices que tem toda a sua vizinhança limpa, deixando um agente para evitar re-contaminação, e, em seguida, ocupam-se todos os vértices seguintes ao vértice ocupado, respeitando o limite do número de agentes. De uma maneira mais formal, definindo s como $\lceil \frac{p-k}{k-1} \rceil$, tem-se $A_0 = \{v_1, \ldots, v_k\}$ e $A_i = \{v_{i*k-(i-1)}, \ldots, v_{(i+1)*k-i}\}$, para todo $0 < i < s$. Finalmente, $A_s = \{v_{s*k-s-1}, \ldots, v_p\}$.

Como $|A_i| \leq k$ para todo $0 \leq i \leq s$, conclui-se que S é uma estratégia de procura rápida capaz de limpar o grafo P em $s+1$ passos, portanto, $fvs(G,k) = \lceil \frac{p-k}{k-1} + 1 \rceil$. □

Teorema 8.2. *Se G é um caminho com p vértices, P_p, então $vs(G,t) = \lceil \frac{p-1}{t} \rceil + 1$.*

Prova: Seja $P = \langle v_1, v_2, \ldots, v_p \rangle$ os vértices do caminho P_p ordenados como no Teo-

rema 8.1. Seja S a seguinte estratégia de procura utilizando $\lceil \frac{p-1}{t} \rceil + 1$ agentes. No passo 1, ocupam-se os vértices v_1 à $v_{\lceil \frac{p-1}{t} \rceil + 1}$. No passo $i > 1$, removem-se os agentes dos vértices que tem toda a sua vizinhança limpa, deixando um agente para evitar re-contaminação, e, em seguida, ocupam-se todos os vértices seguintes ao vértice ocupado, respeitando o limite do número de agentes. De uma maneira mais formal, tem-se $A_0 = \{v_1, \ldots, v_{\lceil \frac{p-1}{t} \rceil + 1}\}$ e $A_i = \{v_{i*\lceil \frac{p-1}{t} \rceil + 1}, \ldots, v_{(i+1)*\lceil \frac{p-1}{t} \rceil + 1}\}$, para todo $0 < i < t$. Finalmente, $A_{t-1} = \{v_{(t-1)*\lceil \frac{p-1}{t} \rceil + 1}, \ldots, v_p\}$.

S é uma estratégia de procura que limpa todos os vértices do grafo com $\lceil \frac{p-1}{t} \rceil + 1$ agentes em t passos. Logo $vs(G,t) \leq \lceil \frac{p-1}{t} \rceil + 1$ e, portanto, $vs(G,t) = \lceil \frac{p-1}{t} \rceil + 1$. □

8.3 Modelos Lineares Inteiros

Nesta seção, dois modelos lineares para o problema de Procura em Vértices Rápida são apresentados. Não foram encontrados outros modelos, na literatura, para esse problema, porém há um artigo, ainda em revisão, que apresenta modelos para uma variação da Procura em Vértices Rápida. Nessa variação existem possivelmente vários fugitivos, a cada passo um fugitivo só pode se movimentar para a vizinhança do vértice que ele ocupa e os agentes podem "atravessar" arestas, porém não podem ser removidos do grafo.

Em ambos modelos exibidos as variáveis binárias x_{ij} representam a ocupação do vértice i no passo j por um agente, o valor 1 indicando que o vértice está ocupado por um agente nesse passo. As variáveis binárias v_{ij} representam a limpeza do vértice i no passo j, o valor 1 indicando, nesse caso, que o vértice está limpo no passo j.

As equações 8.3 e 8.14 garantem que ao final da limpeza todos os vértices devem estar limpos. As equações 8.4 e 8.15 indicam que no início da limpeza todos os vértices estão contaminados. As equações 8.5 e 8.16 garantem que um vértice ocupado por um agente está limpo. Somente são consideradas estratégias monotônicas para a procura (8.6 e 8.17). As equações 8.7 e 8.18 indicam que um vértice pode se manter limpo se ele estava limpo no passo anterior ou se está ocupado por um agente. As equações 8.8 e 8.19 garantem que um vértice limpo só não é recontaminado se ele estiver ocupado. Por outro lado, as equações 8.9 e 8.20 indicam que um agente só pode deixar o vértice que ele se encontra se seus vizinhos estão limpos.

No modelo da Figura 31, a variável X representa o número máximo de agentes utilizados simultaneamente em um passo.

No modelo da Figura 32, as variáveis y_j indicam se no passo j algum vértice foi processado. A variável Y representa o número de passos utilizados. As equações 8.21 e 8.22 são usadas para calcular Y. As equações 8.13 restringem o número de agentes utilizados para no máximo k.

Figura 31: Minimizar número de agentes em t passos.

$$\min X \tag{8.1}$$

subject to:

$$\sum_{i \in V} x_{ij} \leq X \qquad \forall j \in [0..t] \tag{8.2}$$

$$v_{it} = 1 \qquad \forall i \in V \tag{8.3}$$

$$v_{i0} = 0 \qquad \forall i \in V \tag{8.4}$$

$$x_{ij} \leq v_{ij} \qquad \forall i \in V, j \in [0..t] \tag{8.5}$$

$$v_{i(j-1)} \leq v_{ij} \qquad \forall i \in V, j \in [1..t] \tag{8.6}$$

$$v_{ij} \leq v_{i(j-1)} + x_{ij} \qquad \forall i \in V, j \in [1..t] \tag{8.7}$$

$$v_{ij} \leq v_{uj} + x_{ij} \qquad \forall (i,u) \in E, j \in [0..t] \tag{8.8}$$

$$x_{i(j-1)} - x_{ij} \leq v_{u(j-1)} \qquad \forall (i,u) \in E, j \in [1..t] \tag{8.9}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in V, j \in [0..t] \tag{8.10}$$

$$v_{ij} \in \{0,1\} \qquad \forall i \in V, j \in [0..t] \tag{8.11}$$

Figura 32: Minimizar número de passos com k agentes.

$$\min Y \tag{8.12}$$

subject to:

$$\sum_{i \in V} x_{ij} \leq k \qquad \forall j \in [0..n] \tag{8.13}$$

$$v_{in} = 1 \qquad \forall i \in V \tag{8.14}$$

$$v_{i0} = 0 \qquad \forall i \in V \tag{8.15}$$

$$x_{ij} \leq v_{ij} \qquad \forall i \in V, j \in [0..n] \tag{8.16}$$

$$v_{i(j-1)} \leq v_{ij} \qquad \forall i \in V, j \in [1..n] \tag{8.17}$$

$$v_{ij} \leq v_{i(j-1)} + x_{ij} \qquad \forall i \in V, j \in [1..n] \tag{8.18}$$

$$v_{ij} \leq v_{uj} + x_{ij} \qquad \forall (i,u) \in E, j \in [0..n] \tag{8.19}$$

$$x_{i(j-1)} - x_{ij} \leq v_{u(j-1)} \qquad \forall (i,u) \in E, j \in [1..n] \tag{8.20}$$

$$j * y_j \leq y \qquad \forall j \in [0..n] \tag{8.21}$$

$$\sum_{i \in V} (v_{ij} - v_{i(j-1)})/n \leq y_j \qquad \forall j \in [1..n] \tag{8.22}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in V, j \in [0..n] \tag{8.23}$$

$$v_{ij} \in \{0,1\} \qquad \forall i \in V, j \in [0..n] \tag{8.24}$$

9 Conclusão

O problema da Limpeza de Vizinhança Rápida foi estudado por meio de diferentes abordagens. Foi demostrado que é *NP-difícil* encontrar tanto $fl(G,k)$ quanto $l(G,t)$, para um grafo G e inteiros k e t. Em uma tentativa de simplificar o problema, uma versão em que os agentes não podem se mover, Limpeza de Vizinhança com Agentes Fixos, foi estudada e, para essa versão mais restrita, um algoritmo de complexidade $O(n^{k+2})$ foi desenvolvido para calcular $fl(G,k)$. Obviamente, tal algoritmo só e útil para valores pequenos de k. Ainda é uma questão em aberto se a Limpeza de Vizinhança com Agentes Fixos é um problema *polinomial*. Em uma outra tentativa de simplificar a Limpeza de Vizinhança Rápida, foram consideradas estratégias de limpeza com apenas 1 ou 2 passos. Dessa forma, um paralelo com o problema da cobertura de vértices pode ser traçado.

Foi estudado também a Limpeza de Vizinhança Rápida restrita a DAGs. Para este caso, um limite inferior para $fl(G,k)$ foi determinado. Foram definidos os separadores de DAGs e uma propriedade quanto ao valor de $fl(G,1)$ com relação a esses separadores. A classe dos grafos direcionados formados por caminhos disjuntos também foi definida e um algoritmo de complexidade $O(n^3)$ para calcular $fl(G,k)$, quando G pertence a essa classe, foi desenvolvido. Foi feita uma conjectura sobre o formato de uma estratégia de limpeza para uma subdivisão de estrela direcionada, como também outra conjectura para árvores direcionadas top-down. Finalmente, um modelo matemático de programação linear inteira para calcular $fl(G,k)$ e um modelo matemático de programação linear inteira para calcular $l(G,t)$, para grafos direcionados quaisquer, foram construídos e implementados, porem falta testar o desempenho de tais modelos.

No problema da Procura em Vértices Rápida, foi exibida uma propriedade relacionando os valores $fvs(G,k)$ e $vs(G,t)$ de um grafo G com esses mesmos valores aplicados a um subgrafo de G. Também foram criados e implementados modelos matemáticos de programação linear inteira para calcular os valores $fvs(G,k)$ e $vs(G,t)$, para grafos quaisquer. Deixa-se para um trabalho futuro a análise do desempenho de tais modelos.

Lista de Figuras

1 Exemplo de grafo de permutação. As cores representam a bijeção entre os vértices do grafo e os segmentos de reta. . . . . p. 16

2 Exemplo de grafo trapezoidal. As cores representam a bijeção entre os vértices do grafo e os trapézios. . . . . p. 16

3 Decomposição em árvore de um grafo. Em vermelho estão representadas as sacolas da decomposição e em azul a árvore dessa decomposição. . . . p. 18

4 Decomposição em caminho de um grafo. Em vermelho estão representadas as sacolas da decomposição e em azul o caminho associado. . . . . p. 18

5 Uma estratégia para limpar um caminho utilizando 2 policiais. Os dois policiais são diferenciados pelas cores branca e verde. Os conjuntos C_i estão representados pela cor azul. . . . . p. 22

6 Uma estratégia para limpar um caminho utilizando apenas um policial. Os conjuntos C_i estão representados em azul. . . . . p. 25

7 Uma estratégia de procura em arestas de um fugitivo visível em uma árvore utilizando apenas um policial. O fugitivo é representado pelo vértice em vermelho. As elipses representam as componentes conexas do grafo induzido pelos vértices não ocupados pelos agentes. Na primeira etapa, todas as arestas estão contaminadas. Na segunda etapa, coloca-se um agente na raiz da árvore. Nas etapas subsequentes, o policial atravessa a aresta ainda contaminada incidente ao vértice que ele ocupa. p. 27

8 Exemplo de grafo no qual uma procura mista visível com dois fugitivos utiliza menos agentes que essa mesma procura com a restrição de monotonicidade. A cor vermelha representa os fugitivos, a cor verde o agente e a cor azul o subgrafo limpo. . . . . p. 33

9 Exemplo de procura mista visível de 8 fugitivos em uma árvore. A cor vermelha representa os fugitivos, a cor azul o subgrafo limpo, a cor verde os agentes e as elipses representam componentes da árvore. . . . . p. 34

10 Estratégia de limpeza em um ciclo. A cor azul representa os conjuntos C_i em cada passo. Em (a), os agentes são postos no grafo e é possível limpar um vértice. Em (b), sem mover os agentes, é possível limpar um segundo vértice. Em (c), o agente "branco" é movido e mais um vértice é limpo. Em (d), o agente "verde" é movido e todos os vértices restantes podem ser limpos. . . . . p. 37

11 A rede e suas conexões à esquerda a conexão "r" não pode ser atendida. A rede e suas conexões no estado final ao centro. O grafo de dependência obtido à direita. . . . . p. 38

12 Todos os vértices em vermelho fazem parte de uma transversal de ciclos mínima. . . . . p. 39

13 Estratégia de procura em arestas para limpar um caminho com apenas um passo. A cor azul representa a parte limpa do grafo e a cor preta a parte contaminada. . . . . p. 41

14 Estratégia de procura em arestas para limpar um caminho com apenas um passo utilizando uma interpretação diferente para operações atravessar simultâneas. A cor azul representa a parte limpa do grafo, a cor preta a parte contaminada e a vermelha representa o fugitivo. O fugitivo é capaz de "seguir" os agentes enquanto eles "atravessam" arestas. . . . . p. 42

15 Estratégia de procura em arestas para limpar um ciclo com um número par de vértices com apenas um passo. . . . . p. 42

16 Estratégia de procura em arestas para limpar um ciclo com um número par de vértices com apenas um passo. A cor verde representa um agente que não se movimenta durante a estratégia de procura. . . . . p. 43

17 Exemplo de estratégia de procura em vértices rápida de um caminho. Nessa estratégia foram utilizados dois agentes e 4 passos. . . . . p. 44

18 Grafo obtido a partir de $U = \{2, 2, 3, 4\}$. . . . . p. 49

19 Após a colocação do agente no primeiro passo, (a), todos os vértices do grafo são limpos sem que seja necessária uma mudança de posição do agente. A cor azul representa os conjuntos C_i em cada passo. . . . . p. 51

20 Em (a), os vértices pertencentes a menor cobertura do grafo subjacente estão com a cor vermelha. Em (1), são mostradas as posições dos agentes no primeiro passo e os vértices limpos em azul. Em (2), o restante dos vértices são limpos, pois todos os vértices os quais ainda não foram limpos estão ocupados por agentes. . . . . p. 54

21 Em (a), os vértices que fazem parte de uma cobertura mínima do grafo estão com a cor vermelha. Em (1), é mostrado a posição dos agentes e os vértices limpos em azul. Em (2), todos os vértices do grafo estão limpos. p. 55

22 Exemplo de separadores para um DAG e um particionamento regulado. O particionamento está representado pelas elipses em vermelho e os separadores pela cor azul. . . . . p. 56

23 Um exemplo no qual, mesmo sem existir um separador no grafo, o número de passos necessários para limpar o grafo com um agente é igual a metade do necessário para limpá-lo sem a utilização de agentes. . . . . p. 58

24 Exemplo de um grafo composto por caminhos disjuntos . . . . . p. 58

25 Exemplo de estrela direcionada . . . . . p. 62

26 Exemplo de subdivisão de estrela direcionada . . . . . p. 62

27 Exemplo de uma T_2^3. . . . . p. 64

28 Minimizar número de agentes em t passos. . . . . p. 65

29 Minimizar número de passos com k agentes. . . . . p. 66

30 Neste grafo é possível realizar uma procura com 9 agentes em 6 passos, mas uma decomposição em caminho com largura 8 que minimize a soma $\sum_{i=1}^{t'-1} |X_{i-1} \cap X_i|$ precisa de 11 passos. . . . . p. 68

31 Minimizar número de agentes em t passos. . . . . p. 70

32 Minimizar número de passos com k agentes. . . . . p. 71

Lista de Tabelas

1 Parâmetros . . . . . p. 28

2 Complexidade dos algoritmos existentes para resolver o problema de procura em grafo de um fugitivo invisível, considerando um grafo G com n vértices, m arestas e largura em caminho p. . . . . p. 35

Índice Remissivo

(p,t)-estratégia de limpeza, 46
(p,t)-estratégia de procura, 45
$(\leq p, \leq t)$-estratégia de limpeza, 46
$(\leq p, \leq t)$-estratégia de procura, 45
k-árvore, 17
k-conexo, 13
árvore, 14
árvore bottom-up, 63
árvore direcionada, 63
árvore enraizada, 14
árvore top-down, 63

adjacente, 11
ancestral, 14
arbusto, 19
aresta, 11
aresta múltipla, 11
ativo, 27

borda, 13
buraco, 13

caminho, 13
ciclo, 13
cintura, 13
circuito, 13
clique, 12
co-grafo, 15
cobertura de vértices, 14
coloração, 14
coloração própria, 14
complemento, 12, 14
componente, 13
conectado, 13
conexa, 26
conexo, 13
conjunto estável, 12
conjunto independente, 12
contaminado, 22
corda, 13
corte de arestas, 13
corte de vértices, 13

DAG, 14
decomposição em árvore, 17
decomposição em árvore própria, 17
decomposição em caminho, 18
decomposição em caminho direcionada, 19
decomposição em caminhos própria, 18
descendente, 14
desconexo, 13
diâmetro, 13
digrafo, 11
distância, 13
dual fraco, 15

edge search, 25
estratégia de limpeza, 36
estratégia de procura, 21
estratégia de procura em arestas, 24
estratégia de procura mista, 26
estratégia de procura rápida, 43
estrela, 15
estrela direcionada, 61
etapa, 21, 43
extremidade, 11

face, 15
face externa, 15
filho, 14
floresta, 14
folha, 14
fonte, 12

grafo, 11
grafo r-partido, 14
grafo "k-starlike", 17
grafo "split", 15
grafo "starlike", 16

grafo bipartido, 14
grafo completo, 12
grafo cordal, 16
grafo de dependência, 37
grafo de interseção, 15
grafo de intervalo, 15
grafo de permutação, 16
grafo direcionado, 11
grafo isomorfo, 14
grafo orientado, 11
grafo simples, 11
grafo subjacente, 12
grafo trapezoidal, 16
grafo unicíclico, 15
grafo vazio, 12
grau, 11
grau de entrada, 12
grau de saída, 12

incidir, 11
interna, 27
interno, 14
inverso, 12
invisível, 27
isomorfismo, 14

laço, 11
largura de banda, 19
largura de ordenação linear, 20
largura em árvore, 17
largura em árvore própria de um grafo, 17
largura em banda topológica, 19
largura em caminho, 18
largura em caminho direcionada, 19
largura em caminho própria de um grafo, 19
largura linear, 20
Limpeza de Vizinhança, 36
limpo, 22

mixed search, 26
monotônica, 26

nós, 14
número cromático, 14
número de Limpeza, 36
número de Limpeza com restrição de tempo, 46
número de Procura em Arestas, 25
número de Procura em Vértice com restrição de tempo, 44
número de Procura em Vértices, 23
número de Procura Mista, 26

ordenação linear, 19
ordenação linear de arestas, 20

pai, 14
partição de um conjunto, 14
particionamento regulado, 56
passo, 21, 43
periplanar, 15
planar, 15
preguiçoso, 27
procura, 21
Procura em Grafos, 21
Procura em Vértices, 21
Procura Mista, 26
procura rápida, 43

raiz, 14
recontaminado, 22

sacola, 17, 18
separação de vértices, 19
separador, 56
simétrico, 12
subdivisão de estrela direcionada, 61
subgrafo, 12
subgrafo gerador, 12
subgrafo induzido, 12
sumidouro, 12
supergrafo, 12

tempo mínimo de limpeza, 46
tempo mínimo de procura, 44
tocar, 13
transversal de ciclos, 20
trilha, 12

vértice, 11
vertex search, 23
visível, 27
vizinhança, 11
vizinhança de entrada, 12
vizinhança de saída, 12
vizinho, 11

X-componente, 13

Referências

[1] FOMIN, F. V.; THILIKOS, D. M. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, v. 399, n. 3, p. 236–245, 2008. ISSN 0304-3975.

[2] PARSONS, T. D. Pursuit-evasion in a graph. In: *Theory and applications of graphs*. Berlin: Springer, 1978, (Lecture Notes in Mathematics, v. 642). p. 426–441.

[3] BREISCH, R. An intuitive approach to speleotopology. *Southwestern Region of the National Speleological Society*, v. 4, p. 72–78, 1967.

[4] GOLOVACH, P. Equivalence of two formulations of a search problem on a graph (russian). *Vestnik Leningrad*, Univ. Mat. Mekh. Astronom, p. 10–14, 1989.

[5] MEGIDDO, N. et al. The complexity of searching a graph. *J. ACM*, ACM, New York, NY, USA, v. 35, n. 1, p. 18–44, 1988. ISSN 0004-5411.

[6] JOSE, N.; SOMANI, A. Connection rerouting/network reconfiguration. *Design of Reliable Communication Networks*, p. 23–30, out. 2003.

[7] COUDERT, D. et al. Rerouting requests in wdm networks. In: *AlgoTel'05*. [S.l.: s.n.], 2005. p. 17–20.

[8] COUDERT, D.; SERENI, J.-S. *Characterization of graphs and digraphs with small process number*. [S.l.], set. 2007. Disponível em: <http://hal.inria.fr/inria-00171083/fr/>.

[9] BONDY, J. A.; MURTY, U. S. R. *Graph Theory with Applications*. [S.l.]: North-Holland, 1976.

[10] KIROUSIS, M.; PAPADIMITRIOU, C. H. Interval graphs and searching. *Discrete mathematics*, Elsevier, Amsterdam, PAYS-BAS (1971), INIST-CNRS, Cote INIST : 15322, v. 55, n. 2, p. 181–184, 1985. ISSN 0012-365X.

[11] BIENSTOCK, D.; SEYMOUR, P. Monotonicity in graph searching. *J. Algorithms*, Academic Press, Inc., Duluth, MN, USA, v. 12, n. 2, p. 239–245, 1991. ISSN 0196-6774.

[12] KIROUSIS, M.; PAPADIMITRIOU, C. H. Searching and pebbling. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, v. 47, n. 2, p. 205–218, 1986. ISSN 0304-3975.

[13] ELLIS, J. A.; SUDBOROUGH, I. H.; TURNER, J. S. The vertex separation and search number of a graph. *Inf. Comput.*, Academic Press, Inc., Duluth, MN, USA, v. 113, n. 1, p. 50–79, 1994. ISSN 0890-5401.

[14] MAZOIT, F.; NISSE, N. Monotonicity of non-deterministic graph searching. In: . Essex, UK: Elsevier Science Publishers Ltd., 2008. v. 399, n. 3, p. 169–178. ISSN 0304-3975.

[15] BODLAENDER, H. L.; THILIKOS, D. M. Computing small search numbers in linear time. In: . Univ. : Utrecht, 1998. Disponível em: <http://webdoc.sub.gwdg.de/ebook/ah/2000/techrep/ CS-1998/1998-05.pdf>.

[16] THILIKOS, D. M. Algorithms and obstructions for linear-width and related search parameters. *Discrete Appl. Math.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 105, n. 1-3, p. 239–271, 2000. ISSN 0166-218X.

[17] FELLOWS, M. R.; LANGSTON, M. A. On search decision and the efficiency of polynomial-time algorithms. In: *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1989. p. 501–512. ISBN 0-89791-307-8.

[18] LAPAUGH, A. S. Recontamination does not help to search a graph. *J. ACM*, ACM, New York, NY, USA, v. 40, n. 2, p. 224–245, 1993. ISSN 0004-5411.

[19] PIERRE, F.; NICOLAS, N. Monotony properties of connected visible graph searching. In: . INIST-CNRS, Cote INIST : 16343, 35400015360598.0210: pringer, Berlin, ALLEMAGNE (1973) (Revue), 2006. v. 4271, n. 32, p. 229–240. ISSN 0302-9743. Disponível em: <http://www.informatik.uni-trier.de/ ley/db/conf/wg/wg2006.html>.

[20] YANG, B.; DANNY, D.; BRIAN, A. Sweeping graphs with large clique number. In: *International symposium on algorithms and computation*. [S.l.: s.n.], 2004. p. 908–920.

[21] TAKAHASHI, A.; UENO, S.; KAJITANI, Y. Mixed searching and proper-pathwidth. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, v. 137, n. 2, p. 253–268, 1995. ISSN 0304-3975.

[22] MAKEDON, F. S.; PAPADIMITRIOU, C. H.; SUDBOROUGH, I. H. Topological bandwidth. *SIAM Journal on Algebraic and Discrete Methods*, SIAM, v. 6, n. 3, p. 418–444, 1985. Disponível em: <http://link.aip.org/link/?SML/6/418/1>.

[23] SEYMOUR, P.; THOMAS, R. Graph searching, and a min-max theorem for treewidth. *Journal of Combinatorial Theory (Series B)*, v. 58, p. 239–257, 1993.

[24] DENDRIS, N. D.; KIROUSIS, L. M.; THILIKOS, D. M. Fugitive-search games on graphs and related parameters. In: *WG '94: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*. London, UK: Springer-Verlag, 1995. p. 331–342. ISBN 3-540-59071-4.

[25] BODLAENDER, H. L. A linear time algorithm for finding tree-decompositions of small treewidth. In: *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1993. p. 226–234. ISBN 0-89791-591-7.

[26] PENG, S.-L. et al. Graph searching on chordal graphs. In: *ISAAC '96: Proceedings of the 7th International Symposium on Algorithms and Computation*. London, UK: Springer-Verlag, 1996. p. 156–165. ISBN 3-540-62048-6.

[27] BARRIÈRE, L. et al. Connected and internal graph searching. In: *In 29th Workshop on Graph Theoretic Concepts in Computer Science (WG), Springer-Verlag, LNCS 2880.* [S.l.: s.n.], 2003. p. 34–45.

[28] LALI, B. et al. Searching is not jumping. In: . NIST-CNRS, Cote INIST : 16343, 35400011781326.0040: Springer, Berlin, ALLEMAGNE (2003), 2003. v. 2880, p. 34–45. ISSN 0302-9743.

[29] NISSE, N. Connected graph searching in chordal graphs. *Discrete Applied Maths.*, 2008. In press, published online on sept 2, 2008.

[30] FOMIN, F. V.; THILIKOS, D. M.; TODINCA, I. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, v. 22, p. 213–216, 2005. Disponível em: <http://dblp.uni-trier.de/db/journals/endm/endm22.html FominTT05>.

[31] FRAIGNIAUD, P.; NISSE, N. Connected treewidth and connected graph searching. In: *Proceeding of the 7th Latin American Symposium on Theoretical Informatics (LATIN).* [s.n.], 2006. p. 479–490. Disponível em: <http://www.informatik.uni-trier.de/ ley/db/conf/latin/latin2006.html>.

[32] YANG, R. Z. B.; CAO, Y. Searching cycle-disjoint graphs. In: . [S.l.]: Springer Berlin / Heidelberg, 2007. v. 4616, p. 32–43. ISSN 0302-9743 (Print), 1611-3349 (online).

[33] ALSPACH, B. et al. Lower bounds on edge searching. In: . [S.l.]: Springer Berlin / Heidelberg, 2007. v. 4614, p. 516–527. ISSN 0302-9743.

[34] RICHERBY, D.; THILIKOS, D. M. Graph searching in a crime wave. *SIAM Journal on Discrete Mathematics*, SIAM, v. 23, n. 1, p. 349–368, 2009. Disponível em: <http://link.aip.org/link/?SJD/23/349/1>.

[35] FOMIN, F. V.; HEGGERNES, P.; MIHAI, R. Mixed search number and linear-width of interval and split graphs. 2008. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary10.1.1.74.3794>.

[36] GUSTEDT, J. On the pathwidth of chordal graphs. *Discrete Appl. Math.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 45, n. 3, p. 233–248, 1993. ISSN 0166-218X.

[37] KLOKS, T.; KRATOCHVÍL, J.; MüLLER, H. Computing the branchwidth of interval graphs. *Discrete Appl. Math.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 145, n. 2, p. 266–275, 2005. ISSN 0166-218X.

[38] SCHEFFLER, P. A linear algorithm for the pathwidth of trees. In: BODENDIEK, R. H. R. (Ed.). *Topics in Combinatorics and Graph Theory.* [S.l.]: Physica-Verlag Heidelberg, 1990. p. 613–620.

[39] PENG, S.-L. et al. Edge and node searching problems on trees. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, v. 240, n. 2, p. 429–446, 2000. ISSN 0304-3975.

[40] ELLIS, J.; MARKOV, M. Computing the vertex separation of unicyclic graphs. *Inf. Comput.*, Academic Press, Inc., Duluth, MN, USA, v. 192, n. 2, p. 123–161, 2004. ISSN 0890-5401.

[41] MONIEN, B.; SUDBOROUGH, I. H. Min cut is np-complete for edge weighted trees. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, v. 58, n. 1-3, p. 209–229, 1988. ISSN 0304-3975.

[42] KLOKS, T. *Treewidth*. Tese (Doutorado) — Utrecht University, 1993.

[43] ARNBORG, S.; CORNEIL, D. G.; PROSKUROWSKI, A. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 8, n. 2, p. 277–284, 1987. ISSN 0196-5212.

[44] BODLAENDER, H. L.; MöHRING, R. H. The pathwidth and treewidth of cographs. *SIAM J. Discret. Math.*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 6, n. 2, p. 181–188, 1993. ISSN 0895-4801.

[45] BODLAENDER, H. L.; KLOKS, T.; KRATSCH, D. Treewidth and pathwidth of permutation graphs. *SIAM J. Discrete Math.*, v. 8, n. 4, p. 606–616, 1995.

[46] BODLAENDER, H. L. et al. *Treewidth and Minimum Fill-in on D-Trapezoid Graphs*. 1998.

[47] BODLAENDER, H. L.; KLOKS, T. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, Academic Press, Inc., Duluth, MN, USA, v. 21, n. 2, p. 358–402, 1996. ISSN 0196-6774.

[48] YANG, B.; CAO, Y. Directed searching digraphs: Monotonicity and complexity. In: . [S.l.]: Springer Berlin / Heidelberg, 2007. v. 4484, p. 136–147. ISSN 0302-9743.

[49] YANG, B.; CAO, Y. Digraph strong searching: Monotonicity and complexity. In: . [S.l.]: Springer Berlin / Heidelberg, 2009. v. 4508, p. 37–46. ISSN 0302-9743.

[50] MUKHERJEE, B. *Optical WDM Networks*. [S.l.]: Springer, 2006. ISBN 978-0-387-29055-3.

[51] DUTTA, R.; KAMAL, A.; ROUSKAS, G. (Ed.). *Traffic Grooming for Optical Networks: Foundations, Techniques and Frontiers*. [S.l.]: Springer, 2008. (Optical Networks). ISBN 6978-03877, 10-0387745173.

[52] SAENGUDOMLERT, P.; MODIANO, E.; GALLAGER, R. On-line routing and wavelength assignment for dynamic traffic in WDM ring and torus networks. *IEEE/ACM Trans. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 2, p. 330–340, 2006. ISSN 1063-6692.

[53] LU, K.; XIAO, G.; CHLAMTAC, I. Analysis of blocking probability for distributed lightpath establishment in WDM optical networks. *IEEE/ACM Trans. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 13, n. 1, p. 187–197, fev. 2005.

[54] GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN 0716710455.

[55] CHANG, R. S. Single step graph search problem. *Information processing letters*, Elsevier Science, Amsterdam, PAYS-BAS (1971) (Revue), INIST-CNRS, Cote INIST : 15156, 35400001084731.0090, v. 40, n. 2, p. 107–111, 1991. ISSN 0020-0190.

[56] HSIAO, J. Y.; TANG, C. Y.; CHANG, R. S. Solving the single step graph searching problem by solving the maximum two-independent set problem. *Information processing letters*, Elsevier Science, Amsterdam, PAYS-BAS (1971) (Revue), INIST-CNRS, Cote INIST : 15156, 35400002300342.0090, v. 40, n. 5, p. 283–287, 1991. ISSN 0020-0190.

[57] HSIAO, J. Y.; TANG, C. Y.; CHANG, R. S. The summation and bottleneck minimization for single-step searching on weighted graphs. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 74, n. 1-2, p. 1–28, 1993. ISSN 0020-0255.

[58] HSIAO, J. Y. et al. Single step searching in weighted block graphs. *Inf. Sci. Inf. Comput. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 81, n. 1-2, p. 1–29, 1994. ISSN 0020-0255.

[59] BRANDENBURG, F. J.; HERRMANN, S. Graph searching and search time. In: . INIST-CNRS, Cote INIST : 16343, 35400015360614.0150: Springer Berlin / Heidelberg, 2006. v. 3831, p. 197–206. ISSN 0302-9743.