ALGORITHMIC COMPLEXITY AND EXTREMALITY CHARACTERIZATIONS FOR EDGE SEARCHING AND ITS VARIATIONS

ÖZNUR YAŞAR


CENTRE FOR NFLD. STUDIES
MEMORIAL UNIVERSITY
OF NEWFOUNDLAND

ALGORITHMIC COMPLEXITY AND EXTREMALITY CHARACTERIZATIONS FOR EDGE SEARCHING AND ITS VARIATIONS

by

© Öznur Yaşar

A Thesis Submitted to the School of
Graduate Studies in partial fulfillment of
the requirement for the degree of
Doctor of Philosophy

Department of Mathematics and Statistics
Memorial University of Newfoundland

May 2008

St. John's Newfoundland

Abstract

Edge searching is a combinatorial game played on graphs. The aim is to construct a search strategy to catch an intruder hidden in the graph independent of his actions. If the intruder has a diffused form then searching corresponds to cleaning the graph. A related problem consists of minimizing the number of searchers used in this search. Various versions of edge searching have been introduced in the past depending on how searchers and the intruder can move. In this dissertation we define Weighted Search and Fast Search as two new variants and answer some complexity and extremality problems.

Weighted Search corresponds to cleaning a contaminated graph where edges may have different capacities. The main result we have is that Weighted Search is an NP-complete problem. We also give comparison results such as bounds on the weighted search number in terms of related graph parameters including pathwidth. We characterize those graphs which two searchers can clean.

Fast Search is an internal monotone search where no edge is traversed more than once in a non-weighted graph. We present a linear time algorithm to compute a fast search strategy for a given tree. We investigate the fast search strategies for bipartite graphs.

The construction of k-searchable graphs, those graphs which k searchers can clean, has been of major interest. Graphs that are $1, 2$ or 3-searchable have been completely characterized previously, whereas characterizing 4-searchable graphs was left as an

open problem. We solve this problem partially and give insights for future work.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor Dr. D. Dyer for providing his valuable time and expert opinions. Our various meetings and reading courses have given me invaluable mathematical insights. His encouragement and constant guidance were my main support.

I wish to express my warm and sincere thanks to my supervisors Dr. M. Kondratieva and Dr. D. Pike for theoretical and professional advice and experience sharing. I also would like to acknowledge all my supervisors for their instrumental guidance and financial support throughout my doctoral degree.

I would like to thank my examination committee; Dr. R. J. Nowakowski, Dr. N. Clarke and Dr. N. Shalaby for their comments on the results presented here and suggestions about future directions.

I would like to thank Dr. B. Yang for our numerous discussions and for sharing his insightful comments. I am grateful to Dr. D. Morgan and N. McKay for fruitful discussions.

Many thanks to Dr. P. Booth and Dr. S. Kocabıyık as members of my supervisory committee. I am also greatly indebted to many teachers in the past; Dr. T. Erlebach for getting me interested in combinatorial mathematics, Dr. N. Shalaby for introducing me to design theory. I also thank the School of Graduate Studies and the Department of Mathematics and Statistics for financial support and its members for moral support.

I also want to thank İdil, Jason, Bursin, Michelle, Oktay, Onar, Şule, Ayhan, Ali Kerem, Bora, Ann, İştar, Burcu, Lili, Vicki and Xiande for their friendship during my years in St. John's. And my friends outside of St. John's; Zelha, Evren, Yeliz, Mert, Willi, Jan, Matúš and Peter. Many thanks to my family and my partner Çağrı for their continued support.

List of Figures

Contents

Chapter 1

Introduction

Edge searching is a pursuit evasion game played on graphs. In the past two decades there has been fundamental work devoted to this topic by scientists in many diverse fields, but mainly in discrete mathematics, operations research and computer science. We consider the problem of constructing a search plan in order to find a person lost in a system of caves which can be represented as a graph. Equivalently, we aim to clean a network of tunnels filled with noxious gas using as few cleaners as possible. In this chapter we define the problem mathematically and mention some of the related results and applications. We introduce Weighted Search and Fast Search as two new models in which we address different aspects of the problem. These two models are motivated by theory and application. In the first section we give the basic definitions and notations that we shall use throughout the text.

1.1 Terminology

A *graph* is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges of G. The edges are unordered pairs of elements of V. We will normally denote an edge $e = \{u, v\}$ as $e = uv$. Two vertices are *adjacent* if there is an edge connecting

them. If $e = uv$, then we say that u and v are the *end vertices* of the edge e and e is said to be *incident to* each of them. Thus an edge $e = uv$ contains both its end vertices u and v.

For a graph $G = (V, E)$ the *order* of G is $|V|$ and the *size* of G is $|E|$. Here the vertical lines around the set denote the cardinality of the set.

A *multigraph* is a graph which may have parallel edges, i.e. $\exists e_1, e_2 \in E$ where $e_1 \neq e_2$ but both have the same end vertices. A *reflexive graph* is a graph which may have loops, i.e. $\exists e \in E$ where $e = uv$ and $u = v$.

The number of edges that contain a vertex v is called the *degree* of v, denoted by $\deg(v)$. Note that a loop at a vertex u will contribute two to the degree of u.

A *path of length n*, denoted as $\mathcal{P}_n$, is a graph with vertex set $V = \{v_0, v_1, \ldots, v_n\}$ and edges $e_i = v_i v_{i+1}$ for every $i = 0, \ldots, n-1$, hence $\mathcal{P}_n = e_0 e_1 \ldots e_{n-1}$. Here the internal vertices are $v_1, \ldots, v_{n-1}$ and the internal edges are $e_1, e_2, \ldots, e_{n-2}$. The number of edges of a path is its *length*.

A *suspended path* in a graph G is a path of length at least 2 such that all internal vertices of the path have degree 2.

A graph is *connected* if there is a path between any two of its vertices.

For given graphs $G = (V, E), G' = (V', E')$, if $V' \subseteq V, E' \subseteq E$ and for every $e = uv \in E', u, v \in V'$, then G' is called a *subgraph* of G. Furthermore G' is called the *subgraph induced by* V' if G' contains all edges of G that join two vertices in V'. If $v \in V'$, then $\deg_{G'}(v)$ denotes the degree of v in G'.

For $n \geq 2$, the graph $C_n := \mathcal{P}_{n-1} \cup v_{n-1} v_0$ is called a *cycle* of length n. The *girth* of a graph G is the minimum length of a cycle that is a subgraph of G.

An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of that cycle.

A graph is *acyclic* if it does not contain any cycles. A connected acyclic graph is called a *tree*. If a vertex of a tree has degree one, then it is called a *leaf*. An edge

$e = uv$ is called a *pendant edge* if $\deg(v) = 1$.

The set of vertices adjacent to a vertex v is called the *neighborhood* of v and is denoted by $N(v)$.

A graph G' is said to be a *subdivision* of a graph G if G' is obtained from G by subdividing its edges, that is by replacing the edges by paths of length two (which is equivalent to inserting a vertex of degree two on every edge).

For a connected graph G, we define the *distance* $d(u, v)$ between two vertices u and v as the length of any shortest $u - v$ path.

A *circuit* is an alternating list of vertices and edges $v_0, e_1, v_1, \ldots, e_k, v_k$ such that

(1) for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i,

(2) for $1 \leq i < j \leq k$, $e_i \neq e_j$, and

(3) $v_0 = v_k$.

An *Eulerian circuit* in a graph is a circuit containing all the edges.

The *complete graph* on n vertices, denoted K_n, is the graph where $d(u, v) = 1$ for every $u, v \in V$. A *clique* in a graph G is a set of pairwise adjacent vertices.

The *complete bipartite graph* on $m+n$ vertices, denoted as $K_{m,n}$, is the graph with vertex set $V = V_1 \cup V_2$ where $V_1 = \{v_1, v_2, \ldots, v_m\}$ and $V_2 = \{u_1, u_2, \ldots, u_n\}$ are called the *bipartition sets*, and the edge set $E = \{e_{ij} = v_i u_j | i = 1, \ldots, m, j = 1, \ldots, n\}$.

The *(Cartesian) product* of two graphs G and H, denoted $G \square H$ (or $G \times H$) has vertex set $V(G) \times V(H)$, and (v_i, w_j) is adjacent to (v_h, w_k) if either

(1) v_i is adjacent to v_h in G and $w_j = w_k$, or,

(2) w_j is adjacent to w_k in H and $v_i = v_h$.

Hence, $G \square H$ is obtained by taking n copies of H and joining corresponding vertices in different copies whenever there is an edge in G. In particular $G = \mathcal{P}_m \square \mathcal{P}_n$ is called an $m \times n$ *grid*.

A graph $G = (V, E, w)$ is called a *weighted graph* if each edge e of G is assigned a nonnegative number $w(e)$ called the *weight* of e. In this thesis, we assume that the

weights assigned are positive integers.

Given two weighted graphs $G = (V, E, w)$ and $G' = (V', E', w')$, if G and G' have the same underlying graphs, i.e. $V = V'$ and $E = E'$, then G' is said to be *lighter than* G when $w'(e) \leq w(e)\ \forall e \in E$.

We say that $G' = (V', E', w')$ is a *subgraph* of $G = (V, E, w)$ when $V' \subseteq V, E' \subseteq E$ and for every $e = uv \in E', u, v \in V'$ and $w'(e) = w(e)\ \forall e \in E'$. For given weighted graphs $G = (V, E, w)$ and $G' = (V', E', w')$, if $V' \subseteq V, E' \subseteq E$ and $w'(e) \leq w(e)\ \forall e \in E'$ then G' is a *lighter subgraph* of G.

In order to define a minor of a given graph we need two operations: *edge deletion*, which corresponds to deleting an edge e, and *edge contraction*, which corresponds to deleting an edge $e = uv$ and identifying the vertices u and v. The second operation corresponds to replacing an edge $e = uv$ with a new vertex v' which is adjacent to all of the former neighbors of u and v.

For given graphs $G = (V, E)$ and $G' = (V', E')$, G' is called a *minor* of G if G' can be obtained from G by a series of edge deletions or contractions. Similarly, given weighted graphs $G = (V, E, w)$ and $G' = (V', E', w')$, we say that G' is a *lighter minor* of G, if G' is a minor of G, considering the corresponding underlying graphs, and $w'(e) \leq w(e)\ \forall e \in E'$.

A vertex is called a *cut vertex* if its removal makes the graph disconnected. A graph is *biconnected* (or *two-connected*) if it has no cut vertices. A *biconnected component* of a graph is a maximal biconnected subgraph.

An edge and a cycle are examples of biconnected components. Observe that every edge belongs to exactly one biconnected component. A vertex may belong to more than one biconnected component, in which case, it is a cut-vertex.

A *path addition* [52] to G is the addition of a path of length at least $n \geq 1$, between two vertices of G introducing $n - 1$ new vertices; the added path is called an *ear*. An *ear decomposition* is a partition of E into sets $H_0, H_1, H_2, \ldots, H_k$ such that H_0 is a

cycle, and H_i is a path addition to the graph formed by $H_0, H_1, \ldots, H_{i-1}$.

Theorem 1 [53] A graph is biconnected if and only if it has an ear decomposition.

The set of biconnected components of a graph G forms a graph, called the *block graph* which has as its vertices the biconnected components and cut vertices of G, and there is an edge between two vertices if one of them is a cut vertex and the other is a biconnected component containing that vertex.

Theorem 2 [28] The block graph of a connected graph is a tree.

A *valid coloring* of a graph $G = (V, E)$ is a labeling $f : E \rightarrow F$ where $F = \{1, 2, \ldots, k\}$. The labels are called the *colors* and $|F|$ is the number of colors used by the coloring f.

A graph is said to be *planar* if it can be drawn in the plane so that its edges intersect only at their end vertices. A drawing of a planar graph G is called a *planar embedding* of G.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are said to be *isomorphic* if there is a bijective mapping f from the vertex set V to the vertex set V' such that $e = uv \in E$ if and only if $e' = f(u)f(v) \in E', \forall e \in E$. The mapping f is called an *isomorphism*. We denote the fact that G and G' are isomorphic by $G \simeq G'$.

The ceiling of a number n is the smallest integer greater than or equal to n, denoted $\lceil n \rceil$.

Let $(\mathcal{G}, +)$ be a finite group with identity element 0. Let $\mathcal{S} \subseteq (\mathcal{G} \backslash \{0\})$ such that $\mathcal{S} = -\mathcal{S}$, that is $a \in \mathcal{S}$ if and only if $-a \in \mathcal{S}$. Recall that $-a$ denotes the inverse of a in $(\mathcal{G}, +)$.

The *Cayley graph* [3] on a group $\mathcal{G}$ with *connection set (or generating set)* $\mathcal{S}$, denoted as $\mathrm{Cay}(\mathcal{G}, \mathcal{S})$, is the graph that is constructed as follows:

(1) Each element of $\mathcal{G}$ corresponds to a vertex v_i, and,

(2) There exists an edge joining v_i and v_j if and only if $v_i = v_j + a$ where $a \in \mathcal{S}$.

A special class of Cayley graphs is those on cyclic groups. A *circulant graph*, denoted as $\text{circ}(n, \mathcal{S})$, is the Cayley graph $\text{Cay}(\mathbb{Z}_n, \mathcal{S})$ where $\mathbb{Z}_n$ is the abelian group of integers modulo n.

1.2 Edge Search

Assume that we want to secure a system of tunnels from a hidden intruder who is trying to avoid us and has unbounded speed. We model this system as a finite connected graph $G = (V, E)$ where junctions correspond to vertices and tunnels correspond to edges. We will launch a group of searchers into the system in order to catch the intruder.

We assume that every edge of G is contaminated initially and our aim is to clean the whole graph by a sequence of steps. At each step we are allowed to do one of the moves defined below.

Definition 1 The following actions that build up an edge search are called *the moves*;

(1) Place a searcher at a vertex,

(2) Remove a searcher from one vertex and place it on another vertex (a "jump"),

(3) Slide a searcher from a vertex along an edge to an adjacent vertex.

Note that placing multiple searchers on any vertex is allowed. We do not pose any restriction on the number of searchers used.

For a given graph, it is a natural question to ask what is the smallest value of k with which we can clean the graph. Next we define this term formally.

Definition 2 If a searcher slides along an edge $e = uv$ from u to v, then the edge e is *cleaned* if either (i) another searcher is stationed at u, or (ii) all other edges incident to u are already clean. An *edge search strategy* is a combination of the moves so that the state of all edges being simultaneously clean is achieved, in which case we say

that the graph is *cleaned.* The least number of searchers needed to clean the graph is the *edge search number* of the graph and is denoted $s(G)$.

All models of searching in this work are variations of edge searching, so we will normally omit the term "edge".

The problem becomes cleaning the graph using the fewest searchers. In this respect, we are interested in the optimal search strategies, those that use only $s(G)$ searchers. If an optimal search strategy is not known, we approximate the search number by relating it to other graph parameters, such as minimum degree, or other search numbers, such as weighted search number.

Notice that even once an edge is cleaned, it may not necessarily be true that it will remain clean until the end of the search strategy. In other words, an edge can be cleaned at some step and at a later step it can get contaminated again.

Definition 3 If a searcher is stationed at a vertex v, then we say that v is *guarded.* If a path does not contain any guarded vertex, then it is called an *unguarded path.* If there is an unguarded path that contains one endpoint of a contaminated edge and one endpoint of a cleaned edge e, then e gets *recontaminated.*

Hence, a clean edge remains clean as long as every path from it to a contaminated edge is blocked by at least one searcher.

The edge search problem has many variants based on, for instance, how searchers move or how the edges are cleaned. We will next introduce the three main variants that are of interest to us.

If we are not allowed to remove a searcher from the graph, then we have an *internal search strategy,* in which case each move is either to place a searcher at a vertex, or, to slide a searcher from a vertex along an edge, to an adjacent vertex. This is equivalent to the case where the 2nd move in Definition 1 is not allowed.

If we insist that once an edge becomes clean it must be kept clean until the end

of the searching strategy, then such a strategy will be called *monotonic*. Hence for every step of a monotone strategy the set of cleaned edges is a subset of the set of cleaned edges at the next step. In other words, each edge should be cleaned once.

If, on the other hand, the set of clean edges induces a connected subgraph of G after each step of the strategy, then the strategy will be a *connected* one.

The minimum number of searchers needed for an internal, monotone or a connected strategy are denoted as $is(G), ms(G)$ and $cs(G)$ respectively. A strategy may combine any of these. Hence, for example, $mis(G)$ will correspond to monotone internal search number which is defined analogously.

The following equation that summarizes the relationship between the search numbers is given in [8]. For an extended version of this paper see [9]. We should mention that through personal communication with the authors of [8] we are informed that it is possible that the inequality $mis(G) \leq cs(G)$ fails to be true for some graphs although known to hold for many classes of graphs.

Theorem 3 [8] For a connected graph $G = (V, E)$, we have

$$s(G) = is(G) = ms(G) \leq mis(G) \leq cs(G) = ics(G) \leq mcs(G) = mics(G). \quad (1.1)$$

There are graphs for which the inequalities are strict [4, 8]. The smallest graph found so far for which $is(G) < mis(G)$ is given in [4, 56].

There are many variants of edge searching that consider one or more of the constraints defined above. In this work we introduce two new models: Weighted Search and Fast Search. We will also partially answer some open problems regarding edge searching.

1.3 Previous Work and Applications

The edge searching problem is an extensively studied graph theoretical problem. Its origins date back to the late 1960s in the works of Breisch [14]. It was first faced by a group of spelunkers who were trying to find a person lost in a system of caves. They were interested in the minimum number of people they needed in the searching team. Parsons [42, 43] was the first one to formalize it as a mathematical problem in 1976. He defined it as a nondiscrete problem where the searchers and the intruder are allowed to move according to continuous functions. In 1982, Petrov [45] defined searching independently. Golovach [24] proved the equivalence of this continuous problem to the discrete one we are considering.

One of the major problems of edge search is to characterize the graphs G such that $s(G) \leq k$ for a fixed positive interger k. A graph G is said to be *k-searchable* if $s(G) \leq k$. It has been shown in [37] that finding whether a graph G is *k-searchable*, i.e. solving the EDGE SEARCHING problem for G, is NP-complete.

Let k be a fixed positive integer. We say that a graph H is a *forbidden minor* for k-searchable graphs if $k < s(H)$ and if any minor of H has search number at most k. For fixed k, the set of forbidden minors for k-searchable graphs is called the *obstruction set*.

The theory on graph minors built by Robertson and Seymour [47, 48] implies that the obstruction set is finite for minor closed families. Furthermore, it is known that edge searching is closed under the taking of minors [42]; that is, if G contains H as a minor, then $s(H) \leq s(G)$. Therefore the obstruction set for k-searchable graphs is finite whenever k is fixed.

On the other hand no general method is known for constructing an obstruction set. Further, the size of such a set is not known either except for some initial cases. The obstruction sets for $k = 2$ and $k = 3$ are given in [37]. However, a construction of the obstruction set is not known for any fixed $k \geq 4$. We partially answer this open

problem in Chapter 4. For results on obstruction sets of graphs with small search numbers refer to [50].

Node search and digraph search are two major variants of edge search. In node search, which is introduced by Kirousis and Papadimitriou [32, 33], we are only allowed to place searchers on vertices and remove searchers from vertices. In this model an edge is cleaned when there are searchers on both of its end points. By reduction from edge search it is shown that node search is NP-complete [11]. It can be seen that the edge search number and node search number cannot differ by more than one [32], namely, if $ns(G)$ denotes the node search number of a graph, then

$$ns(G) - 1 \leq s(G) \leq ns(G) + 1. \tag{1.2}$$

Digraph search is defined as the search problem defined on directed graphs. It is mainly motivated by a graph parameter called directed treewidth [29]. Furthermore, there are variety of search models on directed graphs which may have different rules depending on how the intruder or the searchers traverse the directed edges [6, 41]. The NP-completeness of a directed search model is given by Yang and Cao [54].

Mixed search is defined as a combination of edge search and node search [11]. In this model an edge becomes clean either when both its end points are guarded by searchers or when a searcher slides along it properly. Using mixed search it has been shown [11] that forcing a search to be monotonic does not change the search number, hence we can always assume that edge search strategies are monotonic. A different proof of monotonicity is given by LaPaugh [34]. Similarly, monotonicity does not require more searchers in node search and in mixed search [11].

The relationships between the various search strategies mentioned in Section 1.2 is examined in [8]. For a recent survey on graph searching and its variants, see [23]. For a book on search games, see [1].

We consider guaranteed search strategies; that is, those that capture the intruder regardless of its moves. For a probabilistic approach where randomized algorithms

are utilized, see [30].

The cops and robber model, defined in [40] and [51], is a search model with complete information, i.e., the intruder and the searcher know each others' location. Initially the searchers are located on vertices and then the intruder chooses a vertex. First a subset of searchers move followed by the move of the intruder. After this they alternate moves. A move is to slide along an edge or along a loop. For a graph G the minimum number of cops that guarantee a winning strategy for the cops is the cop number of G. Deciding whether a given graph has cop number at most k, for a given integer k, can be done in polynomial time [10]. In [26] it is shown that infinite chordal graphs do not neccessarily possess a strategy that guarentees a win for the cop(s). An algorithmic characterisation of finite cop-win graphs is given in [27].

The complexity of edge searching and its variations invoked interest in solving these problems on special classes of graphs. Node search and edge search algorithms are given for some subclasses of chordal graphs in [44].

Due to its closeness with the layout problems, the problem is related to widely utilized graph parameters such as pathwidth [19, 31], cutwidth [36], bandwidth [22, 46], linearwidth [12], treewidth [15, 49] and topological bandwidth [35]. It has strong connections with the cutwidth of a graph which arises in VLSI circuit design [16] and with the gate matrix layout problem [39]. For instance, the search number of a graph G equals its cutwidth when G has maximum degree 3 [36]. The pathwidth is node search number minus one [32].

The problem and its variants are related to many applications such as network security [7]. In this application, we consider the capture of a possibly hostile intruder in a given network by software agents. The intruder is arbitrarily fast and has access to information about the position of the agents. They all move along the network links. The problem is to construct the agents' strategy to capture the intruder in an efficient way, which corresponds to minimizing the number of agents used. A similar

application is addressed in robotics for its applications in search and rescue [25]. A typical example for this case is collision avoidance and air traffic control.

Edge search is not only interesting theoretically but also has applications in combinatorial problems [39] such as pebble games that are played on directed acyclic graphs. In a pebble game, initially there are no pebbles on the graph. At every move either a pebble is placed on a vertex with no pebble, or a pebble is deleted from a pebbled vertex. The game ends when all vertices of the graph are pebbled and no pebble is left on the graph. A translation between search problems and pebble games is given in [32]. They show that the minimum number of pebbles used in a monotonic pebble game is equal to the node search number of a graph.

1.4 Weighted Search

Assume that we use a graph to represent a system of gates (which correspond to vertices) and pipes (which correspond to edges) where pipes may have different priorities (depending on size or location). Let us consider these pipes to be full of poison gas. Then we can think of edge searching as cleaning the system of poison gas. If one gate is left open and if gas leakage can occur through that gate then gas will contaminate every pipe that it can reach; that is, all connected pipes with open gates. When a pipe becomes recontaminated, it will do so to its capacity; that is, even if a recontaminated pipe had been partially (or entirely) cleaned, it must now be fully cleaned again.

Motivated by the gas leakage scenario we define weighted search on weighted graphs. Consider a team of searchers (or sweepers) and a finite connected graph G with positive integer weights which represent the maximum amount of contamination of edges. Again we assume that the graph is contaminated initially and our aim is to decontaminate or clean the whole graph by a sequence of steps. At each step we are

allowed to do one of the following *moves*: placing a searcher at a vertex, removing a searcher from a vertex or sliding a searcher along an edge. The moves are the same as those for edge search, whereas the rules for cleaning are slightly different.

Note that, when all edge weights are equal to one, then the weighted edge searching problem becomes the edge searching problem.

Definition 4 If a searcher slides along an edge $e = uv$ from u to v, then *the current positive weight of the edge e is decreased by one* if (i) another searcher is stationed at u, or (ii) all other edges incident to u have weight 0 and the current weight of e is 1, or (iii) u is a leaf.

When a searcher moves from a leaf u to an adjacent vertex, it is not possible to contaminate the graph through u due to the nature of the system and hence we do not need to place a searcher at u. When an edge has weight $w(e)$, it means that a searcher has to slide along e at least $w(e)$ times and decrease the weight at each move.

Assume that the weight of an edge e is decreased after some steps. Then we say that e is *clean* if its weight is reduced to zero and *partially clean* otherwise. We note that this does not guarantee that the weight will remain reduced, because of a possible recontamination. If there is an unguarded path that contains one end point of a partially clean or a contaminated edge and one end point of e, then e gets *recontaminated. If e gets recontaminated, its weight goes back to* $w(e)$*, its original value.* If in the system there occurs a gap in which an intruder (which may have a diffused form as in the gas leakage scenario) can enter a pipe (i.e. contaminate an edge), then we can no longer consider that pipe as clean (and not even partially clean). Recontamination occurs instantly and there is no order of recontamination.

A *weighted edge search strategy* is a combination of the moves defined above that reduces all edge weights to zero. We say that the graph is *cleaned* when the state of all edge weights being zero simultaneously is achieved. The least number of searchers

required in the weighted edge search strategy is the *weighted search number* which is denoted by $ws(G)$.

Weighted searching is a reasonable extension of the searching problem, as in many "real-world" situations, an edge in a graph may represent a pipe or a corridor. Traditional edge searching is not robust enough to deal with situations where particular edges may be more important or may require more effort (be it cost or time) to be cleaned. To return to Breisch's original problem [14], a tunnel in a cave may be quite constricted, allowing only a single searcher through, or broad, requiring several passes to search effectively.

In another search model the edges and vertices have dissimilar weights [7] for internal connected search. The rules of this model is different from the weighted search defined in this thesis.

A weighted graph is said to be *k-searchable* if $ws(G) \leq k$. The decision problem for the weighted case can be stated as below.

WEIGHTED SEARCHING:

Instance A weighted graph $G = (V, E, w)$ and a positive integer k.

Question Is G k-searchable?

By transformation from the MINIMUM CUT INTO EQUAL SIZED SUBSETS problem which is known to be NP-complete we see that WEIGHTED SEARCHING is NP-hard. We will show in Section 2.4 that this decision problem is in fact NP-complete.

1.5 Fast Search

In the majority of networks the tasks relating to cleaning the tunnels are very costly or time consuming. Therefore a good strategy to search a network would require it

to be monotone; that is one would not have to return to edges already examined.

On the other hand, the cost of a searcher used in a search strategy may be low in some search scenarios. Hence we may use more searchers to reduce the time spent for searching. However, at some point we will have loaded the graph with enough searchers that no more searchers would decrease the time needed to clean the graph. Here we are interested in such a case.

Therefore, one definition of an efficient way to clean a graph would be to do so in the minimum number of steps using the least number of searchers. In particular, each edge would be traversed exactly once. First we place a given set of k searchers on a subset of V. Here, we again allow multiple searchers to be placed on a vertex. Then the moves we are allowed to do are of type (3) of Definition 1, i.e., sliding.

We define this new version of edge searching to be *fast searching.* A *fast search strategy* for a graph $G = (V, E)$ is a sequence of $|E|$ moves that clean G. The *fast search number* of G is the least number of searchers for which a *fast search strategy* exists, and is denoted $s_f(G)$. Consequently, this must be an internal monotone search where no edge is traversed more than once.

We accordingly introduce the following decision problem:

FAST SEARCHING:

Instance A graph $G = (V, E)$ and a positive integer k.

Question Is $s_f(G) \leq k$?

We give a linear time algorithm for the FAST SEARCHING problem when it is restricted to trees.

The minimum length of time needed to search a graph using a given number of searchers is a complicated problem and there has not been much work devoted to this topic. In [5], the authors define *one-tick search* number of a graph as the minimum number of searchers needed to capture the intruder in their first move according to

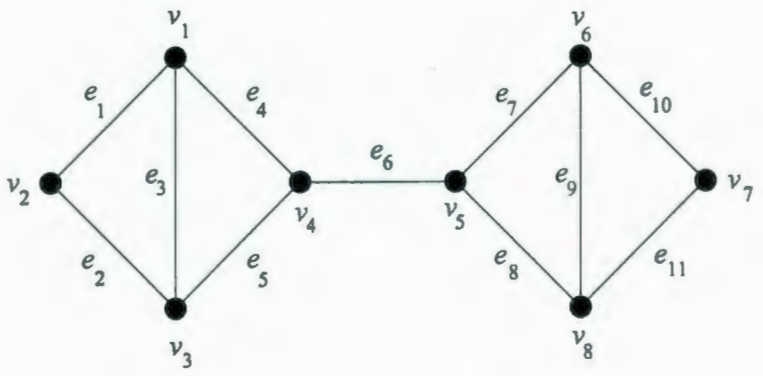


Figure 1.1: A first example

the cops and robber model. They show that one-tick node search number is equal to the domination number for any graph.

Consider a function that gives the cost of searching a graph. Among other parameters, it would depend on the number of searchers and the total time spent for searching the graph. We give a formal definition and some analysis of this function in Chapter 3.

1.6 A first example

Consider the graph $G = (V, E)$ in Figure 1.1. To demonstrate the search models we have introduced so far, we next give an edge search and a fast search for G. We also give a weighted search strategy for a particular weight distribution assigned to the edges in G.

Observe that given a finite reflexive multigraph G, for all of the search models we have introduced, there exists a corresponding search strategy for G that uses a finite number of searchers. This also holds for weighted graphs.

Hereafter σ_i denotes the ith searcher used in the search strategy.

An Edge Search Strategy

Place three searchers, σ_1, σ_2 and σ_3, on v_1. Slide σ_1 along e_1 and then along e_2. This cleans e_1 and e_2. Slide σ_2 along e_3 and clean e_3. Slide σ_2 along e_5 and clean e_5. Next slide σ_3 along e_4. Now σ_2 and σ_3 are both on v_4. Slide σ_3 along e_6 and clean e_6. Remove σ_2 from v_4 and place it on v_5. First slide σ_3 along e_7 and then slide σ_2 along e_8. Remove σ_1 from v_3 and place it on v_8. There are two searchers on v_8 at this step. Let σ_1 slide along e_{11}, e_{10} and then along e_9 in this order. This cleans all edges by 3 searchers. In fact, since G has a forbidden minor for $k = 2$, we know that $3 \leq s(G)$ [37]. Thus $s(G) = 3$.

A Weighted Search Strategy

Let G' be the weighted graph with the underlying graph G in Figure 1.1 and $w(e) = 3, \forall e \in E$. Note that the following weighted search strategy using 4 searchers can also be applied to a weighted graph with arbitrary edge weights all of which are not less than 3.

Place two searchers, σ_1 and σ_2, on v_1. Place σ_3 on v_2 and σ_4 on v_3. Let σ_1 slide along e_1 three times and clean it. Repeat the same for e_2 and e_3. Since all edges incident to v_2 are clean at this step, we can remove σ_3 from v_2 and place it on v_4. Next let σ_1 clean e_4 and then e_5 by keeping other searchers on their places. Notice that after this is done σ_1 is on v_3. Remove σ_1 from v_3 and place it on v_4. Remove σ_2 from v_1 and place it on v_5. Also remove σ_4 from v_3 and place it on v_8. Clean e_6 by σ_1. Remove σ_3 from v_4 and place it on v_6. Next clean e_7, e_8 and e_9 by sliding σ_1 along them as many times as needed. Remove σ_2 from v_5 and place it on v_7. Remove σ_1 from v_5 and place it on v_6. Finally clean e_{10} and e_{11} by σ_1.

This gives us a weighted search that uses 4 searchers. In fact, it is a simple exercise to show that we need at least four searchers to clean a first vertex. Therefore $ws(G) = 4$.

A Fast Search Strategy

Place three searchers, σ_1, σ_2 and σ_3, on v_1. Place σ_4 on v_5 and σ_5 on v_6. Clean v_1 by sliding σ_1 along e_1, then σ_2 along e_3 and σ_3 along e_4. Next clean e_2, e_5 and e_6 by σ_3. Then clean e_7 and e_9 by σ_4. Finally clean e_8, e_{11} and e_{10} by σ_3. Thus we have a fast search strategy that cleans the graph using 5 searchers.

Observe that since an edge can be traversed only once in a fast search strategy, only one of the three searchers can cross through e_6 and thus only one searcher can be transformed from the left part to the right. Also, we need at least three searchers to clean the left side or the right side. This implies that $5 \leq s_f(G)$. Hence $s_f(G) = 5$.

Remark 1 In this text unless it is explicitly stated otherwise we consider simple connected graphs. Our results can naturally be extended to disconnected graphs. For a disconnected graph G, we let $s(G) = \max s(G')$ where G' is a connected component of G. We define $ws(G)$ for disconnected graphs similarly. On the other hand $s_f(G) = \sum s_f(G')$ where G' is a connected component of G. This is because of fast search being internal.

Remark 2 Let S denote any search strategy for G. Assume that σ denotes one of the searchers used in S. We say that v is the *start vertex* for σ, if σ is initially placed on v according to the strategy S. We say that u is the *end vertex* for σ, if σ stops at u (and never moves again).

Chapter 2

Weighted Search

In traditional edge searching the aim is to clean all of the edges in a graph employing the least number of searchers. It is assumed that each edge of the graph is cleaned in the same way and initially each edge has equal contamination that can be considered a weight of one. In this chapter we modify the problem and consider it on graphs with arbitrary positive integer weights assigned to their edges. We give bounds on the weighted search number in terms of related graph parameters including pathwidth. We characterize the graphs for which two searchers are sufficient to clean all edges. We show that for every weighted graph the minimum number of searchers needed for a not-necessarily-monotonic weighted search strategy is enough for a monotonic weighted search strategy, where each edge is cleaned only once. This result proves the NP-completeness of the problem.

2.1 Preliminaries

Let $G = (V, E, w)$ be a weighted graph. Let $w_0(e) := w(e)$ denote the initial weight or contamination of the edge $e \in E$. We denote the contamination of e at step i of a weighted search as $w_i(e)$. Initially all edges are assumed to be contaminated,

therefore $w(e) \geq 1, \forall e \in E$.

If $w_i(e) = 0$, then the weight of edge e is zero at step i and we say that e is *clean* at step i. Note that even if the weight of an edge is zero at some step the edge may be recontaminated at a later point. A vertex u will be said to be *clean* if all edges incident to u are clean.

An *exposed vertex* is a vertex that has at least two edges incident with it, one of which is either clean or partially clean and the other is not clean. For a weighted search S on G, the number of exposed vertices after the ith step is denoted as $wex_S(G, i)$. Let t be the number of steps used in S. The maximum number of exposed vertices is denoted as $mwex_S(G) = \max_{1 \leq i \leq t}\{wex_S(G, i)\}$. Observe that

$$mwex_S(G) \leq ws(G). \tag{2.1}$$

Note that for an unweighted graph G the weighted search number, $ws(G)$, is computed by taking all edge weights equal to one. Similarly, given a weighted graph, $s(G)$ corresponds to the search number of the underlying unweighted graph. Observe that for any weighted graph G we have:

$$s(G) \leq ws(G). \tag{2.2}$$

We can consider internal, monotone or connected weighted search strategies for a weighted graph. The corresponding search numbers are denoted as $iws(G), mws(G)$ and $cws(G)$ respectively. Let us give the following result that states that an internal search does not require more searchers than a not-necessarily internal search for a weighted graph.

Theorem 4 If $G = (V, E, w)$ is a weighted graph, then

$$ws(G) = iws(G).$$

PROOF. Assume that a searcher jumps from u to v according to a weighted search strategy for G. Let P be a path that connects u and v. Such a path exists since G is connected. Hence in the internal search, the searcher can go from u to v along P by a series of moves (slides). Hence any weighted search strategy can be converted to a weighted internal search. ■

An analog of Equation 1.2 that relates node search number and weighted search number is given in the next theorem.

Theorem 5 If G is a weighted graph, then

$$ns(G) - 1 \leq ws(G) \leq ns(G) + 1.$$

PROOF. The Equations 1.2 and 2.2 imply that $ns(G) - 1 \leq s(G) \leq ws(G)$. On the other hand, assume that we are given a monotone node search for the underlying unweighted graph G. Recall that monotonicity does not increase the node search number [32]. Let e be cleaned at step i. Thus there must be a searcher on both end points of e at step i. We construct the weighted search by using an extra searcher, σ, to slide along the edge $w(e)$ times at steps $i_1, i_2, \ldots, i_{w_0(e)}$. This will reduce the weight of e to zero. Thus in weighted search in addition to the steps that make up the node search for the underlying unweighted graph, for every $e \in E$ we will have the steps $i_1, i_2, \ldots, i_{w_0(e)}$ for some i. At the next step of the weighted search, we remove σ and place it to one of the end points of the next edge to be cleaned according to the node search. We apply this to all edges of the weighted graph. Hence $ws(G) \leq ns(G) + 1$. ■

Next we give examples of weighted graphs and their weighted search numbers. Notice that a graph may have three different weighted search numbers depending on the weight distribution.

Example 1 Path of length n: The search number is $s(\mathcal{P}_n) = 1$ whereas

$$ws(\mathcal{P}_n) = \begin{cases} 1, & \text{if } w(e) = 1 \ \forall e \in E \text{ or when } n = 1 \text{ and } w(e) \text{ is arbitrary;} \\ 2, & \text{if } n \geq 2, \exists e \in E \text{ such that } w(e) \geq 2 \text{ and } w(e_i) \leq 2 \text{ where} \\ & i = 1, \ldots, (n-2), \text{ and when } w(e_0) \text{ or } w(e_{n-1}) \text{ are arbitrary;} \\ 3, & \text{otherwise.} \end{cases}$$

Example 2 Loop l: We know that $s(l) = 2$. For any edge weight we see that $ws(l) = 2$.

Example 3 Cycle of length n: For every $\mathcal{C}_n$, observe that $s(\mathcal{C}_n) = 2$. Also note that

$$ws(\mathcal{C}_n) = \begin{cases} 2, & \text{if } w(e) = 1 \quad \forall e \in E; \\ 4, & \text{if } \exists e_1, e_2, e_3 \in E, \text{ each of weight at least 3;} \\ 3, & \text{otherwise.} \end{cases}$$

Example 4 Edge searching a weighted graph is not the same as edge searching an unweighted multigraph where each edge e of weight $w(e)$ is replaced with $w(e)$ parallel edges. One example is the path of length two where both edges have weight 3. Then the corresponding unweighted multigraph, with 3 vertices and 6 edges has search number 3 whereas the weighted graph has search number 2.

2.2 Bounds on Weighted Search Number

In this section we give results that relate the weighted search number with other parameters. First we will consider the complete graphs. For $n \geq 4$ we know that $s(K_n) = n$, [42].

Lemma 6 For $n \geq 4$, we have

$$ws(K_n) = \begin{cases} n+1, & \text{when all edges have weight at least 3,} \\ n, & \text{otherwise.} \end{cases}$$

PROOF. Observe that since $s(K_n) = n$, we have $n \leq ws(K_n)$.

First let $uv = e \in K_n$ such that $w(e) \leq 2$. Place a searcher on every vertex except for u. This accounts for $n-1$ searchers. Place the nth searcher, σ_n, on v. Clean all edges incident to v other than e by σ_n. Remove σ_n and place it on v. Now the only contaminated edge incident to v is e and there are two searchers located on v: σ_1 and σ_n. At this step, say i, let σ_n slide along e from v to u. Since $w(e) \leq 2$, $w_i(e) \leq 1$. Hence we can clean e by sliding σ_1 along e. This cleans v. Now there is a searcher located on every vertex except for v and u contains two searchers. Hence we can clean all the remaining contaminated edges by keeping a searcher on each vertex, except for v, and by sliding σ_n along these edges. Thus, in this case, $ws(K_n) = n$.

Assume that all edges have weight at least 3. Placing a searcher on each vertex and cleaning the edges by the $(n+1)$th searcher gives us a weighted search strategy that uses $n+1$ searchers. Hence $n \leq ws(K_n) \leq n+1$. We show that n searchers are not enough to clean weighted K_n. Notice that to clean a first vertex we need a searcher to guard v, a searcher for each neighbor and another searcher to slide along the edges. This uses $n+1$ searchers. Hence, a first vertex can never be cleaned by n searchers. ■

It is known [42] that if H is a minor of G, then

$$s(H) \leq s(G).$$

However, this result does not hold for monotone search [17]. The following theorem implies that weighted search is also minor closed.

Theorem 7 If $H = (V', E', w')$ is a lighter minor of $G = (V, E, w)$, where G and H are weighted reflexive multigraphs, then

$$ws(H) \leq ws(G).$$

PROOF. Assume that G is cleaned according to a strategy S. Let $f : V \to V'$ be the function that is associated with the edge contractions and deletions which

transform G to H. Assume that S uses m searchers. Using S and f, we construct a weighted search S' for H so that S' uses m searchers.

We order the vertices of G on which the searchers are placed during S as $v_1, v_2, \ldots, v_m$, where v_i's are not necessarily distinct. When searching H, we place the searchers on vertices $f(v_i), \forall i = 1, \ldots, m$, at the same step as they appeared in the strategy S. If $f(v_i) = f(v_j)$, for $i \neq j$, we place both searchers on the same vertex. It is clear that whenever we clean an edge $e \in G$ and e is not deleted from H, we can also clean $e \in H$ by moving the searchers according to S. In detail, when a searcher σ_1 moves from u to v, we will move the searcher on $f(u)$ to $f(v)$. When $f(u) = f(v)$, we do not do anything. If $f(u)$ and $f(v)$ are not adjacent, the searcher on $f(u)$ moves along a path to $f(v)$. If σ_1 can move from u to v in G without occurrence of any recontamination on $e = uv$, there will also be no recontamination when σ_1 leaves $f(u)$. The validity of this operation is due to the placement of the searchers on H. Note that there may be edges that are cleaned in H before they were cleaned in G according to S, nevertheless this does not falsify the weighted search since we do not necessarily construct S' as an internal or a monotone search. Therefore, these modifications will give us a weighted search strategy for H without requiring more searchers. ■

2.2.1 Weighted Search Number and Search Number

We start with two bounds comparing the search number of the graph with the weighted search number. They are shown by modifying the search strategy for the underlying unweighted graph.

Theorem 8 For a weighted reflexive multigraph G,

$$ws(G) \leq s(G) + 2.$$

PROOF. Assume that S is a search strategy that uses $s(G)$ searchers to clean G. We will give a search strategy S' that cleans the weighted G using $s(G)+2$ searchers. To construct S' we start with S and modify it. Since the underlying graph is cleaned by $s(G)$ searchers we can assume that there is a certain time when an edge $e = uv$ is cleaned for the last time. Hence according to S a searcher, σ_0, must traverse the edge e either from u to v or from v to u. Without loss of generality, we can assume that it is cleaned from u to v. When cleaning the weighted graph we place another searcher, σ_1, on u and hold it on u. Then σ_0 slides along e, as it would according to S. The weight of the edge e will be reduced to zero by the second extra searcher, σ_2, which slides along e back and forth. Since e was arbitrary we clean the weighted graph in this way. Note that no recontamination will occur, since $s(G)$ searchers were assumed to be sufficient to clean G. ■

In Examples 1 and 3, we saw that for certain distributions of weights equality holds in Theorem 8. In fact, these graphs constitute an infinite family of such graphs.

Together with Equation 2.2, Theorem 8 implies that $s(G) \leq ws(G) \leq s(G) + 2$ for any reflexive multigraph G and any weight distribution associated with its edges. The next theorem improves this bound for certain weight distributions.

Theorem 9 Let $G = (V, E, w)$ be a weighted reflexive multigraph. If $w(e) \leq 2\ \forall e \in E$, then

$$ws(G) \leq s(G) + 1.$$

PROOF. As in the proof of Theorem 8, if an edge $e = uv$ is cleaned from u to v according to a monotone search strategy, S, we will place an extra searcher on u in S' and reduce the weight by 1 according to S. Then the extra searcher will clean the edge from u to v in S'. ■

Let us denote the minimum vertex degree of a graph G by $\delta(G)$. It has been shown in [17] that $s(G) \geq \delta(G) + 1$ for a connected graph G whose minimum degree

is at least 3. Below is a stronger result for weighted search number.

Theorem 10 Let $G = (V, E, w)$ be a weighted graph. If $w(e) \geq 3 \ \forall e \in E$ and $\delta(G) \geq 3$, then

$$ws(G) \geq \delta(G) + 2.$$

PROOF. We know that $ws(G) \geq s(G) \geq \delta(G)+1$. Consider a search strategy S on G and let the first vertex cleaned be u. As a first case assume that u is of minimum degree. We claim that S uses at least $\delta(G) + 2$ searchers. If the graph induced by $N(u)$ forms a clique, then we know from Lemma 6 that we need at least $\delta(G) + 2$ searchers to clean u, and we are done. Hence assume that the graph induced by $N(u)$ does not form a clique. Let the last cleaned edge that is incident to u be $e = uv$. Then u together with all the remaining $\delta(G) - 1$ vertices adjacent to u must each contain a searcher and there must be one more searcher. Hence all $\delta(G)+1$ searchers are used. Notice that all vertices have minimum degree at least 3, hence none of the $\delta(G) - 1$ searchers located on the $\delta(G) - 1$ adjacent vertices can be moved. This is due to the fact that to be able to remove a searcher from $v \in N(u)$, all neighbors of v other than u must be in $N(u)$ and we need $|N(u)| + 2 = \delta(G) + 2$ searchers. Hence the validity of our claim is shown. Since all the edges have weight at least 3, the searcher on u cannot be moved either. Therefore, u cannot be cleaned, since $e = uv$ cannot be cleaned by a single free searcher and a searcher on u, because all vertices have degree at least 3.

Otherwise, if u is not of minimum degree, then there are at least $\delta(G)+1$ vertices adjacent to u. Since the weights of the edges are at least 3, when u is cleaned u together with all its neighbors must contain a searcher, and there must be one more searcher to clean the edges incident to u. This makes in total at least $\delta(G) + 3$ searchers. Hence the theorem is proved. ■

2.2.2 Weighted Search Number and Pathwidth

Pathwidth is a widely utilized graph parameter in pursuit evasion games including edge searching.

Definition 5 A *path decomposition* of a reflexive graph $G = (V, E)$ is a sequence $X_1, X_2, \ldots, X_r$ of subsets of V such that the following conditions hold:

1. $\bigcup_{i=1}^{r} X_i = V$,
2. $\forall e \in E, \exists i \in \{1, 2, \ldots, r\}$ such that X_i contains every end vertex of e,
3. If $1 \leq i \leq j \leq k \leq r$, then $X_i \cap X_k \subseteq X_j$.

In the definition, the first condition means that all vertices of the graph should be covered by the decomposition. The second condition implies that all edges are covered. The last condition can be regarded as a connectivity condition as once a vertex is included in a set, it should remain in all successive sets if it is to reappear.

Definition 6 The *width* of a path decomposition $X_1, X_2, \ldots, X_r$ for a graph G is $\max |X_i| - 1$ where $i = 1, \ldots, r$. The *pathwidth* of a graph G, denoted by $pw(G)$, is the minimum $h \geq 0$ such that G has a path decomposition of width h.

Example 5 Let $G = \mathcal{P}_n$ with $V = \{v_0, v_1, \ldots, v_n\}$. Then $X_1 = \{v_0, v_1\}, X_2 = \{v_1, v_2\}, \ldots, X_n = \{v_{n-1}, v_n\}$ is a path decomposition and $|X_i| = 2, \forall i = 1, 2, \ldots, n$. Condition two of the definition of a path decomposition implies that $pw(\mathcal{P}_n) \geq 1$. Thus $pw(\mathcal{P}_n) = 1$. Recall that $s(\mathcal{P}_n) = 1$. If H is a weighted path with all edge weights equal to one, then $pw(H) = ws(H) = 1$.

Example 6 For a cycle $\mathcal{C}_n$ with $V = \{v_0, v_1, \ldots, v_{n-1}\}$, observe that $X_1 = \{v_0, v_1, v_2\}$, $X_2 = \{v_0, v_2, v_3\}, \ldots, X_{n-1} = \{v_0, v_{n-2}, v_{n-1}\}$ gives us a path decomposition. Hence $pw(\mathcal{C}_n) \leq 2$. Assume that there exists a path decomposition of width one. Hence all

sets are of size at most two. Also, each vertex must be contained in two consecutive sets. But this cannot hold for each v_i where $i = 0, 1, \ldots, n$ without violating the third condition. Thus $1 < pw(\mathcal{C}_n)$. Therefore $pw(\mathcal{C}_n) = 2$.

Hence we observe that for $\mathcal{C}_n$ all of the three terms are equal, namely, $pw(\mathcal{C}_n) = s(\mathcal{C}_n) = ws(\mathcal{C}_n) = 2$.

Given a graph G, finding the pathwidth of G is an NP-hard problem in general [2]. On the other hand, if we are given a fixed k, then deciding whether $pw(G) \leq k$ can be solved linearly [13].

Theorem 11 [19] For any graph G, $pw(G) \leq s(G) \leq pw(G) + 2$.

We need another definition before we give the main result of this part.

Definition 7 A *vertex separator* of G is a set of vertices, the removal of which makes the graph disconnected. A *layout* of a graph $G = (V, E)$, where $|V| = n$, is a one to one mapping L from V to $\{1, 2, \ldots, n\}$. A *partial layout* of G is a one to one mapping L' from a subset V' of V to $\{1, 2, \ldots, n'\}$ where $n' = |V'|$. Given a partial layout L', we define $V_{L'}(i) := \{v \in V : \exists u \in V$ such that $uv \in E$ and $L'(v) \leq i$ and either $L'(u) > i$ or $L'(u)$ is undefined$\}$. For a given partial layout L' where $|domain(L')| = n'$, the vertex separation of G with respect to L' is defined as $vs_{L'}(G) := \max\{|V_{L'}(i)| : 1 \leq i \leq n'\}$. The *vertex separation* of G is $vs(G) = \min\{vs_L(G) : L$ is a layout of $G\}$.

Theorem 12 [31] For any graph G, $vs(G) = pw(G)$.

In Theorem 13 we will prove that the same bounds in Theorem 11 also hold for weighted edge searching. In the algorithm, for a partial layout L' where $domain(L') = V'$ and $1 \leq i \leq |V'|$, we define the partial layout L_i as the one that assumes the same values for the vertices in $\{L'^{-1}(1), L'^{-1}(2), \ldots, L'^{-1}(i)\}$ and undefined elsewhere. An edge $e = uv$ is *dangling* in L' when $u \in V'$ and $v \notin V'$. A vertex u is *active* in a partial layout L' if $u \in V'$ and u is incident to a dangling edge.

Theorem 13 For any weighted reflexive multigraph G,

$$pw(G) \leq s(G) \leq ws(G) \leq pw(G) + 2.$$

PROOF. The lower bound is trivial due to Theorem 11 and Equation 2.2.

To show the upper bound, we give an algorithm that is derived from Lemma 2.2 in [19]. The algorithm will take as input a weighted graph G, a layout L of G and it will result in all edges of G being simultaneously clean. It will use at most $vs_L(G)+2$ searchers. The result follows due to Theorem 12.

Algorithm WS(G, L)

```
for i := 1 to |V|
do
   let v := L^{-1}(i);
   place a searcher σ_1 at v;
   for u ∈ V such that L(u) < i and e = uv ∈ E
   do
      place a searcher σ_2 at u;
      clean e by sliding σ_2 along e back and forth w(e) times;
      remove σ_2;
   end
   for every loop e = vv ∈ E
   do
     place a searcher σ_2 at v;
     slide σ_2 along e back and forth w(e) times;
     remove σ_2;
   end
   remove searchers from the vertices that are not active in L_i;
end
```

First observe that at the end of ith iteration the set of active vertices has size no more than $|vs_{L_i}(G)|$. Hence the number of searchers that remain on the graph at the end of every iteration is no more than the search number of the graph.

Notice that before the beginning of the ith iteration of the outer do loop, the subgraph induced by the domain of L_{i-1} is cleaned. Furthermore, at each vertex in the domain of L_{i-1} there is exactly one searcher and there are no other searchers on G. This is why no recontamination occurs during the second do loop. By induction we see that G is cleaned as a result of Algorithm $\texttt{WS}(G, L)$. Note also that at each iteration of the algorithm there are at most $vs_L(G) + 2$ searchers on G since at each iteration the algorithm calls for at most two searchers other than the ones on at most $vs_L(G)$ vertices. Hence for an optimal layout, Algorithm $\texttt{WS}(G, L)$ will use at most $vs(G) + 2$ searchers. ■

Example 7 Consider K_n with vertex set $V = \{v_1, v_2, \ldots, v_n\}$. Naturally, $X_1 = V$ is a path decomposition with width $n - 1$. Also, by Theorem 11 $s(G) \leq pw(G) + 2$, thus $n - 2 \leq pw(G) \leq n - 1$.

Assume that $X_1, X_2, \ldots, X_r$ is a path decomposition for K_n such that $|X_i| \leq n-1$. It is easy to see that subsets of size at most k for $k < n - 1$ can not form a path decomposition for K_n. Thus assume that $|X_i| = n - 1$ for every $i \in \{1, 2, \ldots, r\}$. Let $v_1 \notin X_1$ and $v_2 \notin X_2$. Since v_1 and v_2 are adjacent vertices, there exists $i \in \{3, 4, \ldots, r\}$ such that $v_1, v_2 \in X_i$. Hence by definition $v_2 \in X_2$, a contradiction. Therefore $pw(K_n) = n - 1$.

Let G_1 be the weighted K_n with all edge weights equal to two. We have seen that $ws(G_1) = n$. Thus $ws(G_1) = pw(G_1) + 1$.

Furthermore, if G_2 is the weighted K_n with all edge weights equal to three, then $ws(G_2) = pw(G_2) + 2$.

2.3 Restricted Weighted Search

We now consider graphs that can be cleaned by small numbers of searchers. In this section we will consider weighted reflexive multigraphs.

2.3.1 Reduction in Weighted Graphs

First, let us give the conditions for a graph to have weighted edge search number equal to one.

Theorem 14 For a weighted graph G, $ws(G) = 1$ if and only if G is either a path with n edges where all edges have weight one, or G is a single edge of an arbitrary weight.

For the proof note that for $ws(G)$ to be 1, G cannot have a vertex v such that $\deg(v) > 2$, in which case we need at least two searchers to clean v. The same argument shows that G cannot have more than one edge of weight greater than one.

We will introduce the notion of a containment relation between two weighted graphs for which we will define a set of rules.

Definition 8 We say that G *reduces to* G' if G' is obtained from G by applying a series of the following rules, called *reduction rules*;

(1) Any suspended path with edge weights 1 is reduced to a single edge of weight 1.

(2) In a suspended path the consecutive internal edges that have weight 2 are reduced to a single edge of weight 2.

If G and H reduce to the same graph, then we say that G and H have the same reduction.

The first rule in Definition 8 implies that, in the reduced graph, there are no degree 2 vertices whose incident edges both have weight 1. For instance, a path $\mathcal{P}_n$

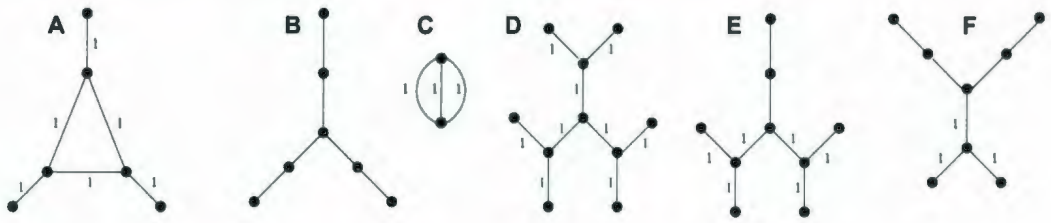


Figure 2.1: Forbidden configurations **A, B, C, D, E** and **F.**

where all edges have weight 1, will reduce to a single edge of weight 1. A cycle C_n where all edges have weight 1, will reduce to a loop.

It is a simple exercise to see that any search strategy for a graph G can be transformed into a search strategy that uses the same number of searchers for the reduced G and vice versa. Hence, we have the following result.

Lemma 15 If G and H have the same reduction, then $ws(G) = ws(H)$.

Definition 9 Given two weighted graphs G and H, we say that G *contains* F if there exists a weighted graph H such that

(1) G and H have the same reduction, and

(2) F is a lighter minor of H.

Recall that weighted searching is minor closed due to Theorem 7. This result and Lemma 15 imply that whenever G contains F, then $ws(F) \leq ws(G)$. We make use of this observation in the proof of Theorem 16.

2.3.2 2-Searchable Graphs

Here we are going to characterize graphs for which $ws(G) \leq 2$. Recall that for any fixed k, the obstruction set is finite since weighted search is minor closed.

Theorem 16 For any reduced graph G, the following are equivalent:

1. $ws(G) \leq 2$

2. G either does not contain any of the configurations **A, B, C, D, E, F** given in Figure 2.1 or the following conditions hold simultaneously:

   (a) G does not contain any edge e that is not a loop or a pendant edge and $w(e) > 2$,

   (b) G does not contain any 2-cycle having an edge of weight greater than 1,

   (c) G does not contain the graphs **D, E** and **F** where any two pendant edges with a common end are replaced with a loop of weight 1,

   (d) Every vertex of degree two in graphs **B, E** and **F** has an edge incident to it with weight 1 and the other edge with weight 2.

3. G consists of a path with vertex set $\{v_1, v_2, \ldots, v_n\}$ together with the following conditions:

   (a) The only edges between v_is are the ones between each consecutive pair and they can either be a single edge of weight at most 2 or a pair of edges of weight 1.

   (b) There may be pendant edges or loops of arbitrary weight attached to each v_i.

PROOF. We will prove the equivalence by showing that *(1)⇒(2)⇒(3)⇒(1)*

(1)⇒(2). None of the graphs that satisfy condition (2) have weighted search number less than 3. The result follows since weighted edge searching is minor closed due to Theorem 7. Let us show for instance that $ws(\mathbf{A}) > 2$. We show that two searchers do not suffice to clean **A**.

If the first vertex cleaned has degree one, then there are two cases to consider. When the second cleaned vertex has degree 3, the two searchers are stuck at

the remaining two vertices of degree 3. Otherwise, if the second cleaned vertex has degree one, both searchers are stuck at the other end points of the pendant edges.

If the first vertex cleaned has degree 3, then we need at least 3 searchers.

(2)⇒(3). G does not contain **C**, hence there are no chords in G. By Theorem 1 edges or cycles are the only possible biconnected components of G. Because of condition (2b), edges of the cycles can only have weight 1. On the other hand, G does not contain **A**, hence at most two vertices of a cycle can have degree at least 3. This implies that the only biconnected components of G are paired edges with weight 1, loops and edges with arbitrary weight. Then by Theorem 2, G must be a tree together with paired edges of weight 1 and loops of arbitrary weight. Due to condition (2a), the internal edges of the tree can have weight at most 2. The result follows if we show that when all of the vertices of degree one are removed, the resulting graph is a path, with possible loops or paired edges that have weights as described in (3). Assume that it is not true. Then, there must be a vertex of G that has three different neighbors none of which is a leaf. When none of these neighbors have degree less than three, G would contain **D** together with condition (2c). Similarly, when all of these neighbors have degree two, G would contain **B** together with condition (2d). When one of these neighbors has degree two, G would contain **E** together with condition (2c) or (2d). Finally, when two of these neighbors both have degree two, G would contain **F** together with condition (2c) or (2d). Since all of these configurations are forbidden, we arrive at a contradiction. Therefore, G has the form given in *(3)*.

*(3)⇒(1).*The first vertex v_1 can be cleaned by putting both searchers on v_1, then by keeping one of the searchers on v_1 and cleaning the incident loops or leaves by

the other searcher. Then both searchers can either move along the edge that connects v_1 to v_2 or each can move along one of the paired edges of weight 1. The same procedure can be applied to v_2 and in this way one can clean the whole graph.

■

2.4 Monotonicity of Weighted Search

In this section, we will show that if there exists a weighted search strategy for a weighted graph G using at most k searchers, then there exists a monotonic weighted search strategy for G using at most k searchers. The *crusade method* is a widely used proof method to show monotonicity in edge searching or its variants. Here the terminology is similar to that used in [11].

2.4.1 Pairs of Crusades

Notice that when sliding a searcher along an edge $e = uv$ from u to v, if no recontamination is possible from u, then either e becomes clean or the current weight of e decreases from k to $k-1$, where $k > 1$, in which case we say that *partial cleaning* is done.

At step i, let the set of cleaned edges correspond to A_i, the set of partially cleaned edges correspond to P_i and let Z_i be the set of vertices where at least one searcher is located. In the edge search there may be more than one searcher located at a vertex, hence we consider Z_i to be a multiset. Set difference, namely, $Z_i \backslash \{u\}$ corresponds to removing one copy of u from the multiset Z_i.

A weighted search strategy S that uses n steps for a weighted graph $G = (V, E, w)$ can be recorded as a sequence of a triples of sets

$$S = (A_i, P_i, Z_i)_{i=0}^{n}$$

such that $A_i \subseteq E, P_i \subseteq E\backslash A_i, Z_i \subseteq V$ for $0 \leq i \leq n$ and $A_0 = P_0 = P_n = \emptyset, A_n = E$. If $v \in V$ is incident with at least one edge in $A_i \cup P_i$ and at least one edge in $E\backslash A_i$, then $v \in Z_i$.

The following are the only possible situations we may encounter during a weighted edge search:

1. Placing new searchers: $A_i = A_{i-1}, Z_i \supseteq Z_{i-1}, P_i = P_{i-1}$.
2. Recontamination:
   - by removing searchers: $A_i \subseteq A_{i-1}, Z_i \subseteq Z_{i-1}, P_i \subseteq P_{i-1}$, or,
   - by sliding a searcher along an edge: $A_i \subseteq A_{i-1}, Z_i = (Z_{i-1}\backslash\{u\}) \cup \{v\}$, $P_i \subseteq P_{i-1}$
3. Partial Cleaning: For $uv = e \in E$ such that $w(e) \geq 2$, $A_i = A_{i-1}$, $Z_i = (Z_{i-1}\backslash\{u\}) \cup \{v\}$ and $P_i = \begin{cases} P_{i-1}, & 2 \leq w_{i-1}(e) < w(e) \text{ or;} \\ P_{i-1} \cup \{e\}, & w_{i-1}(e) = w(e). \end{cases}$
4. Cleaning: For $uv = e \in E$, $A_i = A_{i-1} \cup \{e\}$, $Z_i = (Z_{i-1}\backslash\{u\}) \cup \{v\}$ and $P_i = \begin{cases} P_{i-1}\backslash\{e\}, & 2 \leq w(e) \text{ or;} \\ P_{i-1}, & w(e) = 1. \end{cases}$

We break up the steps of the strategy so that at most one action is done at each move. In this way, in each move we force that at most one edge gets cleaned, partially cleaned or contaminated.

We define a connectivity function δ properties of which will be used in the next sections. For a given edge set E and a vertex set V, for $A \subseteq E, P \subseteq E\backslash A$, $\delta(A, P)$ denotes the set of vertices in V that have at least two edges, e_1 and e_2, incident to it such that $e_1 \in A \cup P$ and $e_2 \in E\backslash A$.

In edge searching, if A is the set of clean edges and P is the set of partially clean edges at some instant, then $\delta(A, P)$ would correspond to the set of exposed vertices. Recall Equation 2.1 which states that every exposed vertex must contain a searcher.

In this section we will need the following lemma.

Lemma 17 [Submodularity] For given pairs of subsets of E, $(A,P),(B,R)$ where $P \subseteq E\backslash A$ and $R \subseteq E\backslash B$, the following holds:

$$|\delta((A\cap B),(P\cap R))| + |\delta((A\cup B),(P\cup R))| \leq |\delta((A,P)| + |\delta(B,R)|. \tag{2.3}$$

PROOF. We consider three cases. Assume that e_1 and e_2 are any two edges incident with u.

CASE 1. $u \in \delta((A\cap B),(P\cap R))$ and $u \in \delta((A\cup B),(P\cup R))$: Since $u \in \delta((A\cap B),(P\cap R))$, $\exists e_1 \in (A\cap B)\cup(P\cap R)$ and hence $e_1 \in A\cup P$ and $e_1 \in B\cup R$. Further, since $u \in \delta((A\cup B),(P\cup R))$, $\exists e_2 \in E\backslash(A\cup B)$ and hence $e_2 \in (E\backslash A)\cap(E\backslash B)$. These observations imply that $u \in \delta(A,P)$ and $u \in \delta(B,R)$.

CASE 2. $u \in \delta((A\cap B),(P\cap R))$ and $u \notin \delta((A\cup B),(P\cup R))$: First, since $(A\cap B)\cup(P\cap R) \subseteq A\cup P$ and $(A\cap B)\cup(P\cap R) \subseteq B\cup R$, $u \in \delta((A\cap B),(P\cap R))$ implies that $\exists e_1 \in (A\cup P)$ and $e_1 \in (B\cup R)$. Furthermore, $u \in \delta((A\cap B),(P\cap R))$ implies that $\exists e_2 \in E\backslash(A\cap B)$. Note that since $u \notin \delta((A\cup B),(P\cup R))$, u has no edge incident with it that is not in $A\cup B$. Therefore $e_2 \in (A\cup B)\backslash(A\cap B) = (A\backslash B)\cup(B\backslash A)$, and hence $e_2 \in A\backslash B$ or $e_2 \in B\backslash A$. The previous observation implies that $e_2 \in E\backslash B$ or $e_2 \in E\backslash A$. These imply that $u \in \delta((A,P)$ or $u \in \delta((B,R)$.

CASE 3. $u \notin \delta((A\cap B),(P\cap R))$ and $u \in \delta((A\cup B),(P\cup R))$: Similar to the first two parts, $u \in \delta((A\cup B),(P\cup R))$ implies that $u \in \delta(A,P)$ or $u \in \delta(B,R)$. ■

Consider a weighted search strategy for a given graph $G = (V,E,w)$. For a sequence of pairs of subsets of the edge set E, $(X_0,Y_0),(X_1,Y_1),\ldots,(X_n,Y_n)$, where $Y_i \subseteq E\backslash X_i$, for $0 \leq i \leq n$ and $X_0 = Y_0 = Y_n = \emptyset, X_n = E$, consider the sequence $(X_0,X_1,\ldots,X_n)$, where $\forall i = 0,1,\ldots,n$ and $\forall e \in X_i$, there exists a step j such that $w_j(e) = 0$. Then $(X_0,X_1,\ldots,X_n)$ is called a *crusade* associated with $(X_0,Y_0),(X_1,Y_1),\ldots,(X_n,Y_n)$ if for all $1 \leq i \leq n$

$$|X_i\backslash X_{i-1}| + |Y_i\backslash Y_{i-1}| \leq 1. \tag{2.4}$$

We say that a crusade uses at most k searchers if $|\delta(X_i, Y_i)| \leq k$ for all $0 \leq i \leq n$.

A crusade is *progressive* if the X_i's form a nested sequence, i.e., $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_n$ and, for all $1 \leq i \leq n$,

$$|X_i \backslash X_{i-1}| + |Y_i \backslash Y_{i-1}| = 1. \tag{2.5}$$

Lemma 18 If $ws(G) \leq k$, then there exists a crusade using at most k searchers.

PROOF. Let $ws(G) \leq k$ and let $(A_0, P_0, Z_0), (A_1, P_1, Z_1), \ldots, (A_n, P_n, Z_n)$ be a weighted search strategy for G. Then $|Z_i| \leq k$ for $0 \leq i \leq n$. From the definition of $\delta(\cdot, \cdot)$, we know that if $v \in \delta(A_i, P_i)$ then v is an exposed vertex. Hence Equation 2.1 implies that $\delta(A_i, P_i) \subseteq Z_i$ and thus $|\delta(A_i, P_i)| \leq |Z_i| \leq k$. Each A_i corresponds to a set of clean edges at step i, hence $\forall e \in A_i, \exists j \leq i$ such that $w_j(e) = 0$. Equation 2.4 holds because of the definition of a weighted search since each step corresponds to only one action. Therefore associated with $(A_0, P_0), (A_1, P_1), \ldots, (A_n, P_n)$, the sequence $A_0, A_1, \ldots, A_n$ is a crusade that uses at most k searchers. ■

Lemma 19 If there exists a crusade using at most k searchers, then there exists a progressive crusade using at most k searchers.

PROOF. To each sequence of pairs $(X_0, Y_0), (X_1, Y_1), \ldots, (X_N, Y_N)$ one can associate two numbers $a(N) = \sum_{i=0}^{N}(|\delta(X_i, Y_i)|+1)$ and $b(N) = \sum_{i=0}^{N} |X_i|$. Among all crusades $(X_0, X_1, \ldots, X_N)$ using at most k searchers and associated to $(X_0, Y_0), (X_1, Y_1)$, $\ldots, (X_N, Y_N)$ we will pick the one for which

1. $a(N)$ is minimum and

2. $b(N)$ is minimum subject to condition (1).

We denote such a crusade by $C = (X_0, X_1, \ldots, X_n)$.

If $|Y_i \backslash Y_{i-1}| = 1$, then $|X_i \backslash X_{i-1}| = 0$ due to Equation 2.4. Instead, assume that $|Y_i \backslash Y_{i-1}| = 0$ and $|X_i \backslash X_{i-1}| = 0$. Then $|Y_{i+1} \backslash Y_{i-1}| \leq 1$ and $|X_{i+1} \backslash X_{i-1}| \leq 1$. Therefore $(X_0, X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ is a crusade with respect to $(X_0, Y_0), (X_1, Y_1), ..$

., (X_{i-1}, Y_{i-1}), $(X_{i+1}, Y_{i+1}), \ldots, (X_n, Y_n)$. For this sequence $a(N)$ takes a smaller value than for C, which contradicts our assumption. Therefore $|X_i \backslash X_{i-1}| = 1$.

We only need to show that X_i's form a nested sequence. Observe that if

$$|\delta(X_{i-1} \cup X_i, Y_{i-1} \cup Y_i)| < |\delta(X_i, Y_i)|,$$

then $(X_0, X_1, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_n)$ is a crusade with respect to (X_0, Y_0), $(X_1, Y_1), \ldots, (X_{i-1}, Y_{i-1}), (X_{i-1} \cup X_i, Y_{i-1} \cup Y_i), \ldots, (X_n, Y_n)$. For this sequence $a(N)$ takes a smaller value than for C, hence

$$|\delta(X_{i-1} \cup X_i, Y_{i-1} \cup Y_i)| \geq |\delta(X_i, Y_i)|. \tag{2.6}$$

Combining Equations 2.3 and 2.6, we have

$$|\delta(X_{i-1} \cap X_i, Y_{i-1} \cap Y_i)| \leq |\delta(X_{i-1}, Y_{i-1})|.$$

From the result above we observe that $(X_0, X_1, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_n)$ is a crusade with respect to $(X_0, Y_0), (X_1, Y_1), \ldots, (X_{i-2}, Y_{i-2}), (X_{i-1} \cap X_i, Y_{i-1} \cap Y_i)$, $(X_i, Y_i), \ldots, (X_n, Y_n)$. From the minimality of $b(N)$ for C we must have

$$|X_{i-1} \cap X_i| \geq |X_{i-1}|.$$

Hence $X_{i-1} \subseteq X_i$. ■

2.4.2 Monotonicity

Showing that weighted searching is monotonic; that is proving that a monotonic weighted search does not require more searchers, will make us conclude on the membership of the WEIGHTED EDGE SEARCHING problem in the NP class. In general, monotonicity is a crutial property for proving complexity results of the decision problems.

Before giving Theorem 22, which is the main result of this chapter we need two lemmata. The first one implies that from a weighted search strategy, we can always

construct another weighted search strategy that has only one partially cleaned edge at every step and at the same time it does not require more searchers.

Lemma 20 If there exists a weighted search strategy $S = (A_i, P_i, Z_i)_{i=0}^{n}$ for a weighted graph $G = (V, E, w)$ that uses k searchers, then there exists a weighted search strategy $S' = (A'_j, P'_j, Z'_j)_{j=0}^{m}$ for G such that $|P'_j| \leq 1 \ \forall j = 0, 1, \ldots, m$ and S' uses k searchers as well. Furthermore, for S' the following hold:

1. If $e \in P'_j$ and $w_j(e) = 1$, then $w_{j+1}(e) = 0, P'_{j+1} = \emptyset$ and $A'_{j+1} = A'_j \cup \{e\}$.

2. If $P'_j = P'_{j-1} = \{e\}$, then $A'_j = A'_{j-1}$.

PROOF. From S, we construct the required strategy S' that uses the same number of searchers to clean G. First, if an edge $e = uv, w(e) \geq 2$ is partially cleaned at step i in S by sliding a searcher σ_1 from u to v, in S' we remove σ_1 from u and place it on v. We modify S' step by step so that according to its final version G will be cleaned. While modifying S' we only need to consider edges that have capacity at least two, since edges of unit capacity are never in any P'_j.

In S during the steps that reduce the weight of an edge $e = uv$, there is a step that results in enough searchers on the ends of e to clean e. If this happens at the kth step in S, then during S' the cleaning can be done successively instead of the kth step. So in S' we will have $w(e_{k_1}) = w(e) - 1, w(e_{k_2}) = w(e) - 2, \ldots, w(e_{k_{w(e)-1}}) = 1$, and $w(e_{k_{w(e)}}) = 0$. We have to show that we can clean every edge in this way.

If a searcher σ_1 ends up on u in S after the kth step but σ_1 ends up on $v \neq u$ in S' after the $k_{w(e)}$th step, then we remove σ_1 from v and place it on u at steps $k_{w(e)+1}$ and $k_{w(e)+2}$.

Let e_0 be the first edge with weight at least 2 that gets cleaned in S. If $e_0 = uv$ is a pendant edge, where $\deg(v) = 1$, then we need at least two searchers to clean it, since $w(e_0) \geq 2$. If these two searchers are put on one or both ends of e_0 for the first

time at step k in strategy S, in S' we clean e_0 in steps $k_1, k_2, \ldots, k_{w(e_0)}$ by letting one of them guard u and the other slide on e until it becomes clean.

If e_0 is not a pendant edge, then we need at least 3 searchers, two to guard the ends of e_0 and one to slide along e_0, when $w(e_0) \geq 2$. If these three searchers are put on ends of e_0 for the first time at step k in strategy S, in S' we clean e_0 in steps $k_1, k_2, \ldots, k_{w(e_0)}$.

Assume that we continue cleaning edges according to S and construct S' in this way. Let $e = uv$ be the next edge that is cleaned according to S at step i.

Note that e might have been cleaned and contaminated during S before step i. But we know that there exists a step $j < i$ in S such that $w_{j-1}(e) = w(e), w_j(e) = w(e) - 1, w_{i-1}(e) = 1, w_i(e) = 0$ and there exists no k such that $j < k < i$ and $w_k(e) = w(e)$. In other words, e does not become recontaminated between step j and step i.

If e is not a pendant edge, then just before the ith step, there must be at least one searcher located on each of u and v.

CASE 1. *e is not a pendant edge and $w(e) = 2$*

At some step, say j in S, $w_{j-1}(e) = 2$ and $w_j(e) = 1$. Here j is a step between the last time e was cleaned and the ith step.

(1) If at the jth step two searchers were located on u and v, one on each vertex, and a third searcher was sliding along e, either from u to v or from v to u, in S' we can clean e in two steps, j_1 and j_2, using the same three searchers.

(2) If at the $(j-1)$th step two searchers, σ_1 and σ_2, were located on u and at the jth step one searcher, σ_2, slid along e from u to v, there are two possibilities to reduce the weight from 1 to 0. If a third searcher slides along e at step k, then we clean e in S' in two steps, k_1 and k_2, with these 3 searchers. Otherwise σ_2 may slide along e from v to u at step k (or σ_1 may slide along e from u to v at step k, which can be transferred to S' similarly). Notice that all of the edges incident to v

are contaminated at step j (since they are on an unguarded path to a contaminated edge e). Furthermore at step $k-1$ all edges incident to v, except for e, must be clean. Otherwise when σ_2 slides along e from v to u, it would not be partially cleaning e. Hence, all edges incident to v, except for e, must be cleaned between the jth step and the kth step and they all have weight 1. At some step l, such that $j < l < k$, one of those edges, say e_1, gets clean by a searcher σ_3 sliding along e_1 either starting from v or ending at v. Therefore in S', we clean e in steps l_1, l_2 where σ_3 slides two times along e.

CASE 2. e *is not a pendant edge and* $w(e) \geq 3$

Since the number of searchers needed for reducing the weight from 3 to 2 and 2 to 1 is the same as reducing the weight from n to $n-1$ and from $n-1$ to $n-2$ for $n \geq 3$, it is enough to consider the case $w(e) = 3$.

At some point, say j during S, $w_{j-1}(e) = 3$ and $w_j(e) = 2$. If this is done by using three searchers, then e can be cleaned in S' in three steps j_1, j_2, j_3 which would replace the jth step of S. If in the jth step two searchers, σ_1 and σ_2, are used by placing both of them on u and sliding one of them to v, after this step, u or v cannot be left unguarded. Hence, to reduce the weight from 2 to 1 we need one more searcher, say σ_3, which is going to slide along e at step k in S. Accordingly, in S', we replace step k with steps k_1, k_2, k_3 where σ_3 slides back and forth along e.

If $e = uv$ is a pendant edge where $\deg(v) = 1$, we consider two cases.

CASE 3. $e = uv$ *is a pendant edge and* $w(e) = 2$

The weight of e should go from 2 to 1 in S at some step, say at k. If this is done by a searcher σ_1 sliding from u to v, then there must be another searcher on u. In S', we replace the kth step with steps k_1, k_2 in which σ_1 slides back and forth along e twice.

If the weight of e is reduced from 2 to 1 by a searcher σ_1 sliding from v to u, then after this step u must always be guarded by a searcher. There are two ways to reduce

the weight from 1 to 0.

Another searcher, say σ_2, may slide along e at the kth step of S. In S' we replace the kth step with steps k_1, k_2 in which σ_2 slides back and forth along e twice.

In S, all the edges incident to u may get clean and σ_1 may slide back from u to v. Then during cleaning of an edge $e_1 \neq e$ incident to u, there must be a searcher σ_2 either sliding from u or ending at u, say at the lth step. Accordingly, in S', just after the lth step, we insert steps l_1 and l_2 in which σ_2 guards u and σ_1 slides twice back and forth along e.

CASE 4. $e = uv$ *is a pendant edge and* $w(e) \geq 3$

Again, we only need to consider the case $w(e) = 3$. At some step the weight of e should go from 3 to 2 during S, say at step j.

If this is done by a searcher σ_1 sliding from u to v, then there must be another searcher on u. In S', we replace the jth step with steps j_1, j_2, j_3 in which σ_1 slides back and forth along e three times.

If a searcher σ_1 slid from v to u to reduce the weight of e from 3 to 2, then, after this step u cannot be left unguarded, otherwise the edge would be recontaminated. Now, to reduce the weight from 2 to 1, another searcher, say σ_2, has to slide along e. If this happens at the kth step of S, in S' we replace the kth step with steps k_1, k_2, k_3 in which σ_2 slides back and forth along e. ■

Lemma 21 Assume that S is a weighted search strategy for $G = (E, V, w)$ that uses k searchers. Then there exists a progressive crusade $(X_0, X_1, \ldots, X_n)$ associated with $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$, where $Y_i \subseteq E \backslash X_i$, for $0 \leq i \leq n$, that uses at most k searchers such that both the following conditions hold:

1. If $|Y_i \backslash Y_{i-1}| = 0$, then either

   (a) $Y_i = \emptyset, Y_{i-1} = \emptyset, X_i \backslash X_{i-1} = \{e\}$, and $w(e) = 1$, or;

   (b) $Y_i = \emptyset, Y_{i-1} = \{e\}, X_i \backslash X_{i-1} = \{e\}$, and $w(e) \geq 2$.

2. If $|Y_i \backslash Y_{i-1}| = 1$, then $Y_{i-1} = \emptyset, Y_i = \{e\}, X_i \backslash X_{i-1} = \emptyset, X_{i+1} \backslash X_i = \{e\}$, and $w(e) \geq 2$.

PROOF. Using the procedure given in the proof of Lemma 20, we construct a weighted search (A'_i, P'_i, Z'_i) such that $|P'_i| \leq 1, \forall i$. Next we delete the (A'_i, P'_i, Z'_i)'s for which $|P'_i| = 1$ and $\exists j \neq i$ such that $(A'_j, P'_j) = (A'_i, P'_i)$ except for the (A'_i, P'_i, Z'_i)'s such that $w_i(e) = 1$ where $\{e\} = P'_i$. We apply Lemmata 18 and 19 to this reduced sequence and obtain a progressive crusade. The two conditions of the theorem follow from the construction of S' and the implications of Lemma 20 if we let $X_i = A_i$ and $Y_i = P_i$. If two consecutive partially cleaned sets are empty, then S' is cleaning an edge of weight 1, which corresponds to part 1(a). Part 1(b) is the same as part (1) of Lemma 20. In both of them an edge e which is partially clean at step $i-1$ becomes clean at step i. Finally, during the consecutive steps where an edge is partially cleaned no other edge is cleaned. This is part (2). ■

In the proof Theorem 22 we will consider the indices of the sets X_i and Y_i as levels. Hence the levels consist of steps. An ith level may not correspond to the ith step in the strategy due to the construction of the progressive crusade in the proof of Lemma 21.

Theorem 22 If there exists a weighted search S using at most k searchers for a weighted graph G, then there exists a monotone edge search S' using at most k searchers for G.

PROOF. Lemma 20 implies that from S we can construct a weighted search S' that uses at most k searchers and $|P'_i| \leq 1$ for every step i. Lemma 21 ensures that there exists a progressive crusade $X_0, X_1, \ldots, X_n$ associated with $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$ which can be obtained from S' and it uses at most k searchers. We construct a monotone weighted search strategy inductively that cleans the edges in the order $e_1, e_2, \ldots, e_m$, where for each e_j there exists i such that $X_i \backslash X_{i-1} = \{e_j\}$. Assume

that we cleaned the edges $e_1, e_2, \ldots e_{l-1}$ in this order and no edge other than these is cleaned. Assume that we finished cleaning e_{l-1} at the end of $(i-1)$th level. We show that in the next steps we will clean e_l. We will use the implications of Lemma 21. We consider three cases.

CASE 1. $|Y_i \backslash Y_{i-1}| = 0$, $Y_i = \emptyset$, and $Y_{i-1} = \emptyset$. Here $X_i \backslash X_{i-1} = \{e_l\}$, and $w(e_l) = 1$. We show below that the discussion given in [11] implies that e_l can be cleaned with at most k searchers. A weighted mixed search is defined as a combination of weighted edge search and node search. According to node searching when $w(e) = 1$, e is cleaned when there are searchers at both of its ends.

First we show the induction step for a weighted mixed search, then we show how to convert it to a weighted search.

Assume that G has no pendant edges. Let $N = \{u, v\}$ where $e_l = uv$. If $|N \cup \delta(X_{i-1}, Y_{i-1})| \leq k$, then we can put searchers at the end points of e_l to clean it. Otherwise, $|N \cup \delta(X_{i-1}, Y_{i-1})| > k$. Since $\delta(X_{i-1}, Y_{i-1}) \leq k$, $N \not\subseteq \delta(X_{i-1}, Y_{i-1})$. Let $v \in N \backslash \delta(X_{i-1}, Y_{i-1})$. Then none of the neighbors of v are in X_{i-1}, hence $v \in \delta(X_i, Y_i)$.

If $u \notin \delta(X_{i-1}, Y_{i-1})$, then none of the neighbors of u and v are in X_{i-1}. Therefore $u \in \delta(X_i, Y_i)$, hence e_l is cleaned by node search.

If $u \in \delta(X_{i-1}, Y_{i-1})$ and all neighbors of u except for e_l are in X_{i-1}, then the searcher on u can slide along e_l to clean it. If $u \in \delta(X_{i-1}, Y_{i-1})$ and not all of the neighbors of u except for e_l are in X_{i-1}, then $u \in \delta(X_i, Y_i)$. Hence, e_l is cleaned by node searching.

If G has pendant edges, then observe that if G' is obtained from G by adding a loop of weight 1 to all its vertices, G and G' will have the same mixed search numbers.

Next, we show how to convert a weighted mixed search to a weighted edge search. Given G construct G' by replacing each edge e of weight 1 by a path $P = ee'$ of length two with edge weights 1. Since G and G' have the same reduction, they have the same mixed search number as well.

If $e_l = uv$ is cleaned by node search at step i, then just before it is cleaned, there must be searchers on u and v. In the weighted edge search S', we will replace the ith step with 3 steps. If e' was contaminated before step i in S, in S' we will remove the searcher σ on v, put it on u and slide it along e_l. If e' was cleaned before step i in S, in S' we will slide the searcher σ on v along e to u, remove it from u and put it on v.

As a result, e_l is cleaned using at most k searchers.

CASE 2. $Y_i = \emptyset, Y_{i-1} = \{e_l\}$. We have $w(e_l) \geq 2$. We consider the cases whether e_l is a pendant edge or not.

(1) If e_l is not a pendant edge, then at the $(i-1)$th level, there should be at least one searcher on each end of e_l, say σ_1 on u and σ_2 on v. If $|\delta(X_{i-1}, Y_{i-1})|+1 \leq k$, then we can finish the cleaning of e_l by the searcher that is free, i.e., the one that is not on any exposed vertex. Otherwise $|\delta(X_{i-1}, Y_{i-1})| + 1 > k$ and since $|\delta(X_{i-1}, Y_{i-1})| \leq k$, we must have $|\delta(X_{i-1}, Y_{i-1})| = k$. There are four subcases to consider when e_l is not a pendant edge.

In the first subcase let each of u and v have at least one edge incident to them other than e_l that is already clean, hence in X_{i-1}. Then before the $(i-1)$th level there must be two searchers located on each of u and v. In this case, the only way that e_l became partially clean for the first time is that a third searcher, other than the ones on u and v, slid along e_l. Hence, e_l can be cleaned by this third searcher in the next steps.

Consider together the subcase where there are no clean edges incident to u or v and the subcase where only one of the end points of e_l, say u, has an edge incident to it that is clean and an edge that is contaminated. In both of these cases the exposed vertices in level $i-1$ are the same as in level i, hence $\delta(X_{i-1}, Y_{i-1}) = \delta(X_i, Y_i)$. Hence none of the searchers can move from their places during the steps between these levels. On the other hand, we know that $|X_i \backslash X_{i-1}| = \{e_l\}$. But there is no possible way to clean e_l when none of the searchers are moving. Hence we arrive at a contradiction.

The last subcase is where all edges other than e_l incident to one of the end points of e_l, say u, are already clean and all edges incident to v are contaminated. We know that $u \in \delta(X_{i-1}, Y_{i-1})$ and $u \notin \delta(X_i, Y_i)$ since at level i all edges incident to u are in X_i (since X_i's are nested). The only way this can happen is that either $w_{i-1}(e_l) = 1$, and we are done, or $w_{i-1}(e_l) > 1$ and a third searcher slides along e_l to clean it totally during the steps between these levels, which is impossible if $\delta(X_{i-1}, Y_{i-1}) = k$ since none of the searchers can move.

(2) If e_l is a pendant edge, let $\deg(v) = 1$ and $\deg(u) > 1$. We only need to consider two subcases. When there is at least one edge incident to u that is clean and hence in X_{i-1}, then before the $(i-1)$th level there must have been a searcher, say σ_1, located on u. As in the previous subcase, the only way for e_l to become partially clean is that a searcher other than σ_1, say σ_2, slides along e_l at the step that corresponds to level $i-1$. Hence, e_l can be cleaned with σ_2 together with σ_1 which will be kept on u as a guard.

Finally, if all edges incident to u other e_l are contaminated, observe that $u \in \delta(X_{i-1}, Y_{i-1}) = \delta(X_i, Y_i)$. This implies that none of the searchers could move during the steps between the levels $i-1$ and i, which contradicts $|X_i \backslash X_{i-1}| = \{e_l\}$.

CASE 3. $|Y_i \backslash Y_{i-1}| = 1$. We have $Y_{i-1} = \emptyset, Y_i = \{e_l\}$. We know that $X_i \backslash X_{i-1} = \emptyset, X_{i+1} \backslash X_i = \{e_l\}$, and $w(e) \geq 2$. This case reduces to the previous case by shifting all the observations to the levels i and $i+1$. Applying the same procedures we can finish the cleaning of e_l at the successive steps after level i. ■

Theorem 22 implies that $ws(G) = mws(G)$. From this we deduce that the WEIGHTED SEARCHING problem belongs to NP, since we only need to guess in which order the edges are cleaned and then to check whether the edges can be cleaned according to this sequence using at most k searchers. In Section 1.4 we noted that the problem is NP-hard. It follows from these two observations that the problem is NP-complete. Hence we have the following.

Corollary 23 The WEIGHTED SEARCHING problem is NP-complete.

This complexity result leads to many possible research directions. A constant factor approximation algorithm with a small constant factor is desirable. Another direction would be restricting the weighted searching problem to particular classes of graphs.

Chapter 3

Fast Search

In this chapter we consider *fast searching* as a variant of the edge searching problem. Fast searching corresponds to an internal monotone search in which every edge is traversed exactly once and the searchers are not allowed to jump. We present a linear time algorithm to compute the fast search number of trees. We investigate the fast search number of complete bipartite graphs. We also propose a general cost function for evaluating search strategies that utilizes both searching and fast searching.

3.1 Preliminaries

For any graph G, it is immediate from the definitions in Section 1.5 that $s(G) \leq s_f(G)$. The gap $s_f(G) - s(G)$ is zero for some graphs. We continue by observing that this holds for K_n.

For K_n, the complete graph on n vertices, where $n \geq 4$, recall that $s(K_n) = n$ [42].

Lemma 24 *For every $n \geq 4$, $s_f(K_n) = n$.*

PROOF. Label the vertices $v_1, v_2, \ldots, v_n$. Place the first $n-1$ searchers, $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$

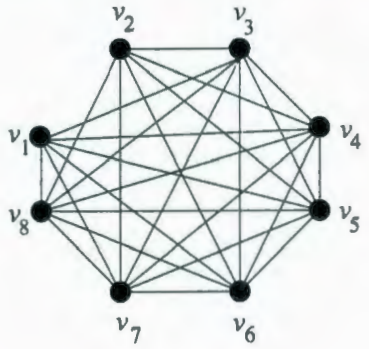


Figure 3.1: $K_8 - e$ where $e = v_1v_2$.

on v_1 and the nth searcher, σ_n, on v_2. First clean v_1 by sliding all the searchers located on it.

When n is even, the K_{n-1} induced by $\{v_2, v_3, \cdots, v_n\}$ contains an Eulerian circuit which is cleaned by σ_n and there is no contaminated edge left.

When n is odd, σ_n cleans the Eulerian circuit contained in the K_{n-2} induced by $\{v_2, v_3, \cdots, v_{n-1}\}$. At this step, the only contaminated edges are those incident to v_n and there is a searcher on each neighbor of v_n and two searchers on v_2. We clean v_n by sliding all these searchers, except for σ_n, to v_n. ■

Furthermore, it is known that the edge search number is critical for complete graphs, that is, deleting any single edge from K_n will reduce the search number by one [17]. Next we show that this also holds for the fast search number.

Lemma 25 For any $e \in E(K_n)$, $s_f(K_n - e) = n - 1$.

PROOF. Let $K_n - e = (V, E)$. Label the vertices $v_1, v_2, \ldots, v_n$ so that for $v_1, v_2 \in V, \deg(v_1) = \deg(v_2) = n - 2$. Refer to Figure 3.1 when $n = 8$. Place the first $n - 2$ searchers on v_2 and the $(n-1)$th searcher on v_{n-1}.

As a first case, let n be odd. Let all $n - 2$ searchers on v_2 slide along the edges incident to v_2. In this way v_2 is cleaned. Next we are going to clean the K_{n-2} induced

by $\{v_3, v_4, \cdots, v_n\}$. Since K_{n-2} has an Eulerian circuit, we can clean the graph by moving the $(n-1)$th searcher as the first $n-2$ searchers remain in their places. Finally, we let the $n-2$ searchers move to v_1 and finish cleaning.

Otherwise, let n be even. Using $n-2$ searchers we clean v_2 as in the previous case. Next we clean an Eulerian circuit in K_{n-3}, induced by $\{v_3, v_4, \ldots, v_{n-1}\}$, using the $(n-1)$th searcher, σ_{n-1}, starting from and ending at v_{n-1}. Then σ_{n-1} may slide along $v_{n-1}v_n$ and then v_nv_1. Now, the only contaminated edge incident to v_{n-1} is $v_{n-1}v_1$ and hence σ_{n-3} may slide along that edge and clean v_{n-1}. The only contaminated edges left in the graph are those of the induced $K_{2,n-4}$ where the bipartition sets are $\{v_1, v_n\}$ and $\{v_3, v_4, \cdots, v_{n-2}\}$. The induced $K_{2,n-4}$ can be cleaned by keeping σ_{n-1} and σ_{n-2} on v_1 and v_n respectively and letting σ_{n-3} slide along the edges. Hence the graph is cleaned. ■

On the other hand, $s_f(G) - s(G)$ can be arbitrarily large for some graphs. Consider $K_{1,n}$, the complete bipartite graph with bipartitions of size 1 and n. Then $s(K_{1,n}) = 2$ whereas $s_f(K_{1,n}) = \lceil \frac{n}{2} \rceil$. For this example the ratio $s_f(G)/s(G)$ is also notably large.

Furthermore, there are graphs for which the connected fast search number may be much larger than the fast search number. Construct G' by subdividing every edge of $K_{1,n}$. Then for every $n \geq 2$, $s(G') = 2, s_f(G') = \lceil \frac{n}{2} \rceil$ whereas the connected fast search number is $n-1$ (see Figure 3.2). Notice that this graph has a much smaller order than the one in [55].

Let V_o be the set of vertices in G with odd degree. Observe that for each vertex $v \in V_o$, there exists a searcher for which v is either a start or end vertex. Otherwise one of the edges incident to v would be traversed at least twice.

Since it is possible that a searcher starts at a vertex in V_o and stops at another vertex in V_o, we have the following lemma.

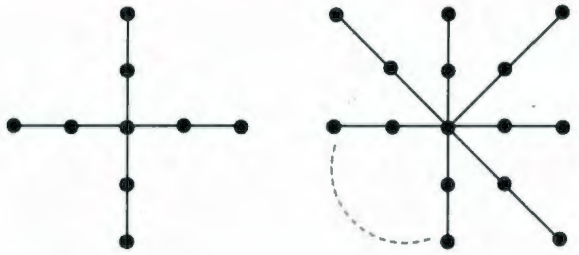


Figure 3.2: The subdivisions of $K_{1,4}$ and $K_{1,n}$.

Lemma 26 If V_o is the set of odd degree vertices in a graph G, then

$$\frac{|V_o|}{2} \leq s_f(G).$$

Example 8 Let $G = \mathcal{P}_{m-1} \square \mathcal{P}_{n-1}$ be an $m \times n$ grid, where $m, n \in \mathbb{Z}$. Lemma 26 implies that $m + n - 4 \leq s_f(G)$. We demonstrate a fast search that uses $m + n - 2$ searchers. Label the vertices of G so that v_{ij} is the jth vertex in the ith row. Figure 3.3 exemplifies this labeling by v_{44}, v_{45} and v_{46}.

Place two searchers, σ_1 and σ_2, on v_{11} and a searcher on each of $v_{12}, v_{13}, \ldots, v_{1(m-1)}$ and $v_{21}, v_{31}, \ldots, v_{(n-1)1}$, denoted $\sigma_3, \sigma_4, \ldots, \sigma_m$ and $\sigma'_1, \sigma'_2, \ldots, \sigma'_{n-2}$ respectively. This accounts for $2 + m - 2 + n - 2 = m + n - 2$ searchers. We construct a fast search so that each $\sigma_i, i = 1, 2, \ldots, m$ cleans a column and each $\sigma'_j, j = 1, 2, \ldots, n - 2$ cleans a row.

The strategy starts with cleaning the first column. First let σ_1 slide along $e = v_{11}v_{21}$. Then let σ_2 slide along $e = v_{11}v_{12}$ and then let σ'_1 slide along $e = v_{21}v_{22}$. We repeat this for all $i = 2, 3, \ldots, (n-1)$ so that we first let σ_1 slide along $e = v_{i1}v_{(i+1)1}$ and then let σ'_i slide along $e = v_{(i+1)1}v_{(i+1)2}$. Finally we let σ_1 slide along $e = v_{n1}v_{n2}$.

At this point, the first column and the first edge of every row is clean. Also, there is a searcher on every vertex of the second column. We clean the second column by σ_2, third column by σ_3 and so on by a similar fashion. Figure 3.3 shows how the fast

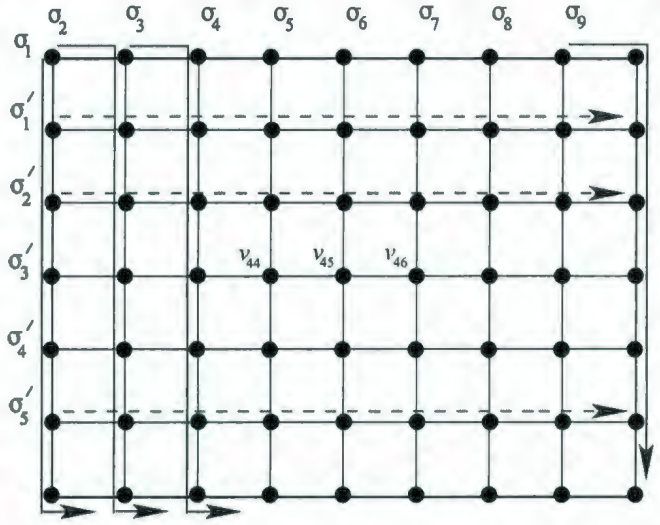


Figure 3.3: Illustration of Example 8 for $m = 9$ and $n = 7$.

search strategy proceeds. Each $\sigma_i, i = 1, 2, \ldots, m$ follows a vertical arrow and each $\sigma'_j, j = 1, 2, \ldots, n-2$ follows a dashed horizontal arrow.

Thus we have

$$m + n - 4 \leq s_f(\mathcal{P}_{m-1} \Box \mathcal{P}_{n-1}) \leq m + n - 2.$$

As we mentioned earlier, the theory built on graph minors plays an important role in edge searching and many variants of edge searching are minor closed. However this

Figure 3.4: The graph G on the left and its subgraph H on the right where $s_f(G) = 3$ and $s_f(H) = 5$.

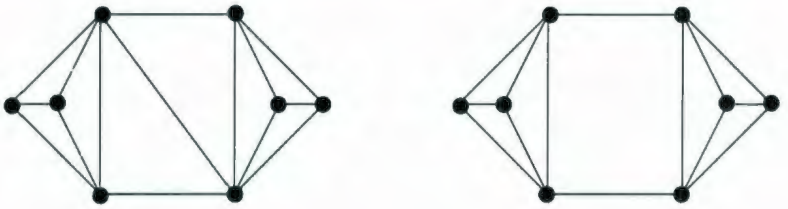


Figure 3.5: The graph G on the left and its subgraph H on the right where $s_f(G) = 5$ and $s_f(H) = 6$.

is not true for fast searching. Figure 3.4 shows an example where G has a smaller fast search number than its subgraph H. Hence fast searching is not even subgraph closed. Notice that H has more odd vertices than G and due to Lemma 26 this result is easy to achieve. On the other hand, Figure 3.5 shows an instance where the subgraph has more even vertices than the original graph, but has a higher fast search number.

Next we give an example that shows that the difference can be arbitrarily large.

Example 9 Let K_n be a complete graph on n vertices where n is odd and $n \geq 5$. Let $G = K_n \Box K_2$ be the Cartesian product of K_n and K_2. Let H be the graph obtained from G by deleting $n-1$ edges between the two maximal cliques in G. Thus, all vertices in G have odd degree n, and in H, two vertices have odd degree n and all other $2n-2$ vertices have even degree $n-1$. We can show that $n \leq s_f(G) \leq n+2$ and $s_f(H) = 2n-1$. Thus, the difference $s_f(H) - s_f(G)$ can be arbitrarily large.

Even more interesting, let $G = K_n \Box \mathcal{P}_m$, where $\mathcal{P}_m$ is a path on $m \geq 2$ edges, and let H be a subgraph of G that is a path of $m+1$ copies of K_n where any two consecutive K_n are connected by a single edge. Then we can show that $s_f(G) \leq n+m+1$ and $s_f(H) = n(m+1) - m$. Hence, the ratio $s_f(H)/s_f(G)$ can be arbitrarily large.

Following the development of edge searching it is interesting to characterize graphs where $s_f(G)$ is small. If k searchers are enough to fast search G, then we say that G

is k-fast-searchable.

We define *reduction* here as reducing any path with consecutive vertices of degree two to a single edge. Thus there is no vertex of degree 2 in the reduced graph. Note that reduction does not change the fast search number, and that $s_f(G) = 1$ whenever G can be reduced to a single edge.

The characterization of graphs G such that $s(G) \leq 2$ is given in [37]. For the characterization of graphs G such that $s_f(G) \leq 2$, we have the following result.

Theorem 27 For any graph G, $s_f(G) \leq 2$ if and only if when G is reduced it consists of a path with vertex set $\{v_0, v_1, \ldots, v_n\}$ together with the following conditions:

1. For every $i = 1, \ldots, n-2$ there are exactly two parallel edges between each pair of consecutive vertices v_i and v_{i+1}.

2. For every $i = 1, \ldots, n-1$ there may be an arbitrary number of loops attached to each v_i.

3. If v_0 and v_1 (resp. v_{n-1} and v_n) are connected by a single edge, there may also be a pendant edge attached to v_1 (resp. v_{n-1}); else if v_0 and v_1 (resp. v_{n-1} and v_n) are connected by two parallel edges, then any number of loops can be attached to v_0 (resp. v_n).

Remark 3 In order to construct reduced biconnected 3-fast-searchable graphs, we notice that $|V_0| \leq 6$ by Lemma 26. Also observe that the set of start vertices for the 3 searchers cannot have size 3. This is due to the biconnectedness of the graph. Thus $|V_0| \leq 4$. We further conjecture that a biconnected 3-fast-searchable graph has at most 2 odd vertices.

3.2 Trees

EDGE SEARCHING is NP-complete for general graphs [37] and planar graphs with maximum degree three [38]. However there exist polynomial time algorithms to solve EDGE SEARCHING when restircted to trees [37]. In this section we show that FAST SEARCHING is solvable in linear time when restircted to trees and construct the corresponding fast search strategy.

For any graph $G = (V, E)$, since $\sum_{v \in V} \deg(v) = 2|E|$, we know that the number of odd degree vertices in G must be even. For a tree T, the following algorithm partitions T into edge disjoint paths such that both of the end vertices of each path have odd degree in T.

Algorithm TP(T)

1. Initially $i = 1$.
2. Arbitrarily select a leaf u in T. Let $uv \in E(T)$ be the edge incident to u. Mark uv with color i and set $T \leftarrow T - uv$.
3. If v is an isolated vertex in T, then go to Step 4; otherwise, let $vw \in E(T)$ be an edge incident to v. Mark vw with color i, $T \leftarrow T - vw$, $v \leftarrow w$, and go to Step 3.
4. If there is no leaf in T, then stop; otherwise, set $i \leftarrow i + 1$ and go to Step 2.

Lemma 28 The Algorithm TP(T) defines a valid coloring for T and the edges with the same color form a path such that the two end vertices of this path have odd degree.

PROOF. Let $e \in T$. In Step 2 when e gets a color, it is deleted from the graph and the algorithm proceeds. Thus every edge gets a color at most once. Also the

algorithm continues as long as there is a leaf in T, thus every edge is colored. Hence every edge is colored exactly once.

The edges that are assigned the same color always form a connected subgraph of the original tree T due to Step 3. Denote that graph as P_i for the ith color. Since a tree is acyclic, no vertex is repeated in P_i, therefore P_i is a path.

Initially one end vertex of P_1 is the leaf chosen by the algorithm at Step 2, thus it has odd degree. The edges are colored 1 according to $\texttt{TP}(T)$ until an isolated vertex is reached in the deformed T which has to be a leaf in T. Hence both end vertices of P_1 have odd degree. Note that after we delete each edge of P_1 from T, the odd-even degree state of each vertex in T does not change except for the end vertices of P_1. Thus a leaf vertex chosen by the algorithm as a start vertex for P_2 and the other end vertex of P_2 found by the algorithm corresponds to either a leaf in the original T or a vertex with odd degree in T. This holds for every $P_i, 2 \leq i$. Hence both end vertices of every path have odd degree in T. ■

Using the decomposition of Algorithm $\texttt{TP}(T)$, we can compute a fast search strategy for a tree T as follows.

Algorithm $\texttt{FS}(T)$

1. Call Algorithm $\texttt{TP}(T)$. Let k be the number of colors used by $\texttt{TP}(T)$ and $P_i, 1 \leq i \leq k$, be the path formed by all edges with color i.

2. For each $P_i, 1 \leq i \leq k$, place searcher σ_i on one end of P_i.

3. Arbitrarily select a searcher, say σ_i, on a leaf, and slide it along P_i until it stops at a vertex v that

   (a) has degree 1 in T, or,

   (b) has degree more than 2 in T and which contains only the searcher σ_i, or,

   (c) is the other end vertex of P_i.

4. Update T by deleting from T all edges cleaned by σ_i in Step 3 after σ_i slides along these edges of P_i. If T contains no edges, then stop; otherwise, go to Step 3.

In Figure 3.6 the application of Algorithm TP(T) and three iterations of Step 3 of Algorithm FS(T) are illustrated.

We now prove the correctness of Algorithm FS(T) by using a counting argument.

Lemma 29 In Algorithm FS(T), if T has at least one edge in Step 4, then when the procedure goes back to Step 3, there must exist a path P_i such that the searcher σ_i on a vertex $u \in V(P_i)$ can clean an edge of P_i incident to u by sliding along this edge.

PROOF. We will assume the opposite of the statement and show that it leads us to a contradiction. Suppose that in Algorithm FS(T) there is a moment at which T has at least one edge and every searcher is located on an isolated vertex or a vertex of degree more than 2 which is occupied by only one searcher. Thus no searcher can move.

For any searcher $\sigma_i, 1 \leq i \leq k$, located on an isolated vertex, we know that all the edges of P_i has been cleaned and deleted. Let F be the forest obtained from T by deleting all isolated vertices. Note that each searcher on F must be located on a vertex that is occupied by only one searcher and is incident with at least two differently colored edges in T.

Since every connected component in F contains at least one searcher, we know that each pair of leaves in F is incident to different colored edges. Thus, the number of different colors in F is greater than or equal to the number of leaves in F.

Furthermore since the degree of every guarded vertex is more than 2, every component has at least 2 leaves. Hence the number of leaves in F is strictly greater than the number of searchers on F.

Thus, the number of distinct colors in F is greater than the number of searchers

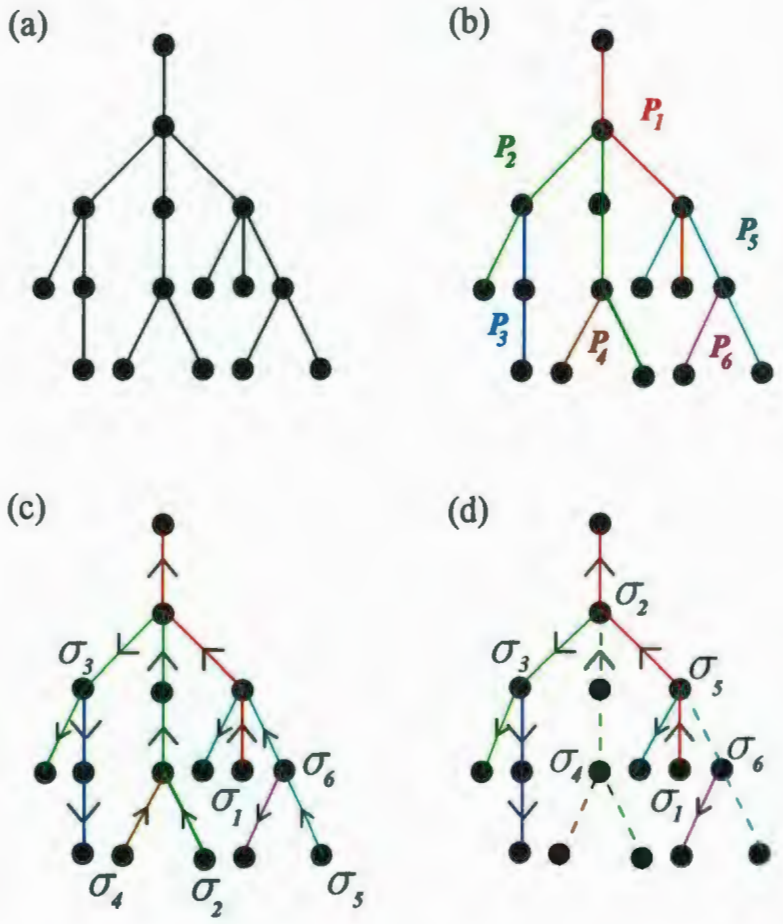


Figure 3.6: The tree algorithm: (a) A tree. (b) An example of paths found by Algorithm TP(T). (c) Searchers' initial placements that also define the direction according to which the edges of each path will be cleaned. (d) State of the graph where the dashed edges are clean due to partial application of Algorithm FS(T).

on F. This contradicts the fact that each path of edges with the same color in F must have one searcher. ■

From the above two algorithms, we can find the fast search number of a tree.

Theorem 30 If T is a tree and V_o is the set of vertices in T with odd degree, then

$$s_f(T) = \frac{|V_o|}{2}. \tag{3.1}$$

PROOF. Let k be the number of colors used by Algorithm TP(T). It follows from Lemma 28 that Algorithm TP(T) decomposes T into k edge disjoint paths $P_i, 1 \leq i \leq k$, such that the two end vertices of each path have odd degree in T. From Lemma 29, we can use k searchers to clean T in such a way that each path P_i is cleaned by one searcher sliding from one end of the path to the other end. Thus, $s_f(T) \leq k = \frac{|V_o|}{2}$. On the other hand, it follows from Lemma 26 that $s_f(T) \geq \frac{|V_o|}{2}$. Therefore, $s_f(T) = \frac{|V_o|}{2}$. ■

Theorem 31 Both Algorithm TP(T) and Algorithm FS(T) can be implemented with linear time.

PROOF. By the depth-first-search algorithm on trees, we know that Algorithm TP(T) can be implemented with linear time. For Algorithm FS(T), Steps 1 and 2 take linear time. In Step 3, after a searcher slides along an edge, we can delete this edge immediately. Since each edge is deleted exactly once, we know that Steps 3 and 4 take linear time. ■

Let us construct a tree for which the gap between the search number and the fast search number is large. From this example we find an infinite family of such graphs.

Definition 10 For a tree $T = (V, E)$, we say that a subtree T' of T is a *branch* of T at $v \in V$ if v has degree one in T' and T' is a maximal subtree with this property.

We quote the following lemma from Parsons' initial paper [42].

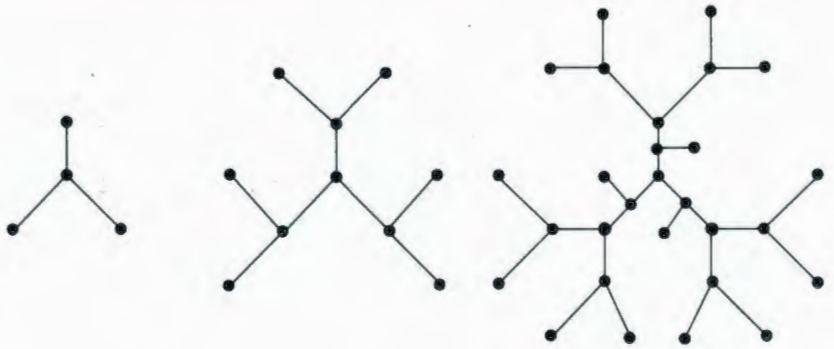


Figure 3.7: The trees T_1, T_2 and T_3 constructed in Example 10.

Lemma 32 [42] For any tree T and positive integer k, $k+1 \leq s(T)$ if and only if T has a vertex v at which there are three or more branches that have search number k or more.

Example 10 Let $T_1 = K_{1,3}$. Take three copies of T_1 and choose a vertex of degree one from each copy. To construct T_2 identify these three vertices. Continue in an inductive fashion. Thus T_k is constructed from three copies of T_{k-1} by identifying a vertex of degree one from each copy. Figure 3.7 depicts T_1, T_2 and T_3.

We show that $s(T_k) = k+1$ for all $k \geq 1$. Observe that $s(T_1) = 2$ and $s(T_2) = 3$. Assume that $s(T_i) = i+1$ for all $i \leq k-1$. Lemma 32 implies that $k+1 \leq s(T_k)$. Call the vertex of T_k obtained by identifying a vertex of degree one from each three copy of T_{k-1} as v. We place a searcher at v and clean each copy of T_{k-1} incident to v by $k-1$ searchers, which is possible due to the induction hypothesis. Thus $s(T_k) = k+1$.

Also, each vertex in T_k is of degree 1 or 3. Thus $V(T_k) = V_o(T_k)$. Since $|V(T_k)| = 3^k + 1$, by Theorem 30 $s_f(T_k) = \frac{3^k+1}{2}$.

3.3 Complete Bipartite Graphs

In this section we consider complete bipartite graphs $K_{m,n}, m \leq n$, and characterize their fast search number. We start with the following initial cases, where $m \leq 2$ or $m = 4$.

Lemma 33 If $1 \leq m \leq n$, then

$$s_f(K_{m,n}) = \begin{cases} \lceil \frac{n}{2} \rceil, & m = 1, \\ 2, & m = n = 2, \\ 3, & m = 2, n \geq 3, \\ 6, & m = 4, n \geq 4. \end{cases}$$

We quote the following theorem on the search number of complete bipartite graphs.

Theorem 34 [4] If $3 \leq m \leq n$, then $s(K_{m,n}) = m + 2$.

The next lemma gives an upper bound for even m where $m \geq 6$.

Lemma 35 If $6 \leq m \leq n$ and m is even, then $s_f(K_{m,n}) \leq m + 3$.

PROOF. Let $K_{m,n}$ have bipartition $V_1 = \{v_1, v_2, \ldots, v_m\}$ and $V_2 = \{u_1, u_2, \ldots, u_n\}$. We construct a fast search strategy that uses $m + 3$ searchers.

Denote the searchers as $\sigma_1, \sigma_2, \ldots, \sigma_{m+3}$. We clean the vertices in the order $u_1, u_2, \ldots, u_{n-1}, v_1, v_2, \ldots, v_m, u_n$ when n is even, and $u_1, u_2, \ldots, u_{n-2}, v_4, v_5, \ldots, v_m, u_{n-1}, v_2, v_1, u_n, v_3$ when n is odd.

First, place σ_1 on v_1, σ_2 on v_2 and σ_3 on v_3 (Figure 3.8). Then place $\sigma_4, \sigma_5, \ldots, \sigma_{m+3}$ on u_1 and then clean u_1 by these searchers so that σ_4 will be on v_1, σ_5 will be on $v_2, \ldots$, and σ_{m+3} will be on v_m. We keep $\sigma_4, \sigma_5, \ldots, \sigma_{m+3}$ on their respective vertices until just before the end of the strategy. We clean $u_2, u_3, \ldots, u_{n-1}$ by moving σ_1, σ_2 and σ_3. Slide σ_1 along the edge $e = v_1u_2$, slide σ_2 along the edge $e = v_2u_2$ and σ_3

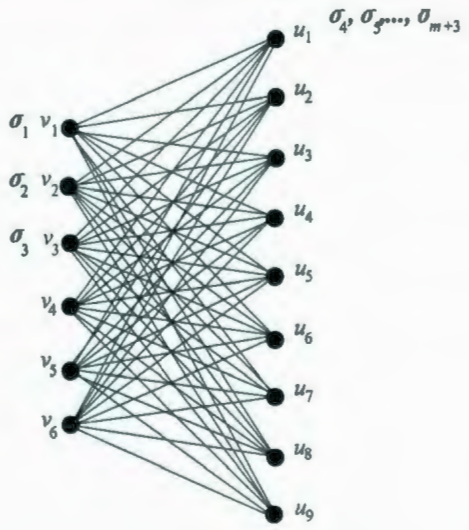


Figure 3.8: The initial placement of the searchers for $K_{6,9}$ according to Lemma 35.

along the edge $e = v_3u_3$. Keeping σ_2 on u_2 and σ_3 on u_3, we clean all the edges between $\{v_4, v_5, \ldots, v_m\}$ and $\{u_2, u_3\}$ by σ_1. Hence σ_1 visits the following vertices in the given order: $u_2, v_4, u_3, v_5, u_2, v_6, u_3, \ldots, v_{m-1}, u_2, v_m, u_3$. After this σ_1 will end up on u_3 since the graph induced by $\{v_4, v_5, \ldots, v_m\}$ and $\{u_2, u_3\}$ contains an Eulerian path and $|\{v_4, v_5, \ldots, v_m\}|$ is odd. Next, σ_1 will slide along u_3v_1. The only contaminated edge incident to u_2 is u_2v_3, hence σ_2 may slide along u_2v_3 and clean u_2. Similarly σ_3 may slide along u_3v_2 and clean u_3. Hence u_2 and u_3 are cleaned.

Next we clean u_4 and u_5 similarly. Slide σ_1 along the edge $e = v_1u_4$, σ_2 along $e = v_3u_4$ and σ_3 along the edge $e = v_2u_5$. Keeping σ_2 on u_4 and σ_3 on u_5, we clean all the edges between $\{v_4, v_5, \ldots, v_m\}$ and $\{u_4, u_5\}$ by σ_1. Then, σ_1 slides along u_5v_1, σ_2 slides along u_4v_2 and σ_3 slides along u_5v_3.

In the same way we clean u_6 and u_7, u_8 and u_9, and so on. If n is even, after cleaning u_{n-2} and u_{n-1} we let σ_i slide along $e = v_{i-3}u_n$ cleaning v_{i-3} for all $i = 4, 5, \ldots, m+3$. In this way we finish cleaning u_n. When n is odd, we follow a similar

strategy. ■

The next theorem shows that the bound in Lemma 35 is best possible.

Theorem 36 If $6 \leq m \leq n$ and m is even, then $s_f(K_{m,n}) = m + 3$.

PROOF. Lemma 35 states that $s_f(K_{m,n}) \leq m + 3$. Also, Theorem 34 implies that $m + 2 = s(K_{m,n}) \leq s_f(K_{m,n})$, we only need to show that $m + 2$ searchers do not suffice to fast search $K_{m,n}$. We will use proof by contradiction. Suppose that there exists a fast search strategy to clean $K_{m,n}$ that uses $m + 2$ searchers. Let $K_{m,n}$ have bipartition V_1 and V_2 with $|V_1| = m$ and $|V_2| = n$. Let u_1 be the first cleaned vertex and t be the step at which u_1 is cleaned. We now consider the case that $u_1 \in V_2$. The case when $u_1 \in V_1$ can be proved similarly.

Since at t, only u_1 is cleaned and all vertices in V_1 are contaminated, each vertex of V_1 must be occupied by at least one searcher. Let us denote these searchers as $\sigma_3, \sigma_4, \ldots, \sigma_{m+2}$. This accounts for m searchers. We show that no strategy can clean the graph using $m + 2$ searchers by considering each placement of the searchers σ_1 and σ_2.

CASE 1. Both of σ_1 and σ_2 are on V_1.

CASE 1.1: Suppose that σ_1 and σ_2 are on the same vertex, say v_1. Since fast searching is a monotone search and none of the searchers are located on $u_2, u_3, \cdots, u_n$, we see that all edges incident to these vertices are contaminated. Since there are no parallel edges, we can slide σ_1 and σ_2 only to different vertices in V_2, say u_2 and u_3. After this step it is not possible to move any of the searchers since they are all on vertices that have more than one contaminated edge incident to them.

CASE 1.2: Suppose that σ_1 and σ_2 are on different vertices, say v_1 and v_2 respectively. First assume that σ_1 and σ_2 slide to the same vertex, say u_2, at steps $t + 1$ and $t + 2$ respectively. Then σ_2 may leave u_2 and slide to any vertex other then v_1 or v_2 in V_1, say v_3 at step $t + 3$. After this step σ_1 is stuck on u_2. At step $t + 4$, σ_2 may

slide to any vertex other than u_1, u_2 in V_2 and it is stuck at that vertex.

Otherwise, let σ_1 and σ_2 slide to different vertices, say u_2 and u_3. But since u_2 and u_3 have at least two contaminated edges incident to them, neither σ_1 nor σ_2 can move. Therefore all searchers are trapped at step $t+2$.

CASE 2. Both of σ_1 and σ_2 are on V_2.

CASE 2.1: Suppose that σ_1 and σ_2 are both on u_1. None of the searchers can move since u_1 is clean and no edge is traversed twice.

CASE 2.2: Suppose that σ_1 and σ_2 are on the same vertex $u_2 \neq u_1$. If u_2 has more than two contaminated edges incident to it, then all searchers are stuck after at most two steps since only one of the searchers, say σ_1 can leave u_2, σ_2 is stuck after this step and σ_1 is stuck at step $t+2$. If u_2 has exactly two contaminated edges incident to it, then u_2 can be cleaned in the next two steps. At the end of step $t+2$, σ_1 and σ_2 are located on different vertices, say v_1 and v_2 in V_1. If both of σ_1 and σ_2 move to different vertices in V_2, then they are both unable to move. Hence assume that σ_1 slides along v_1u_3 and σ_2 slides along v_2u_3. Since all edges incident to u_3 were contaminated before step $t+4$, only one searcher can leave u_3 at step $t+5$. Hence at most six steps later all searchers are stuck. If u_2 has exactly one contaminated edge incident to it, only one searcher, say σ_1, can leave u_2 and the other searcher is stuck. Similar to the previous cases, σ_1 is also stuck after two steps.

CASE 2.3: Suppose that σ_1 and σ_2 are on different vertices, say u_2 and u_3 respectively, where u_2 and u_3 are both different from u_1. The searcher σ_1 (resp. σ_2) can leave the vertex u_2 (resp. u_3) if and only if u_2 (resp. u_3) has exactly one contaminated edge incident to it. When σ_1 and σ_2 move to the same vertex in V_1, this subcase reduces to Subcase 1.1, and all searchers are stuck after at most 4 steps. Similarly when σ_1 and σ_2 move to different vertices in V_1, it reduces to Subcase 1.2. and all searchers are stuck after at most 6 steps.

CASE 2.4: Suppose that σ_1 is on u_1 and σ_2 is on u_2, where u_2 is different from u_1. Now σ_1 cannot move since all edges incident to u_1 are clean. If there is more than one contaminated edge incident to u_2, then σ_2 cannot move. Otherwise, σ_2 can slide along the only contaminated edge incident to it, say $e = u_2v_1$, at step $t+1$. At the next step, either σ_2 or σ_3, say σ_2, can slide along any contaminated edge incident to v_1, say v_1u_3. But none of the searchers can move after this step.

CASE 3. The searcher σ_1 is on $v_1 \in V_1$ and σ_2 is on a vertex in V_2.

CASE 3.1: Suppose that σ_2 is on u_1. Then σ_2 is trapped on u_1 and σ_1 can make at most two moves.

CASE 3.2: Suppose that σ_2 is on $u_2 \neq u_1$. If u_2 has only one contaminated edge incident to it, then there are two cases to consider. If that contaminated edge is v_1u_2 and it is cleaned next, then the problem reduces to Subcase 1.1 if σ_2 slides to v_1 or it reduces to Subcase 2.2 if σ_1 slides to u_2. If v_1u_2 is not cleaned in the next step, then the only searcher that can move is σ_1 and all the searchers are stuck after this step. Otherwise if the contaminated edge is v_2u_2, then σ_2 may slide along that edge and all searchers have the same positions as in Subcase 1.2. If u_2 has exactly two contaminated edges incident to it, then the searchers are stuck after at most one step in the case that none of these contaminated edges are incident to v_1 (only σ_1 can move once). Otherwise, σ_2 and σ_1 are both unable to move at step $t+3$. If there are three contaminated edges incident to u_2, then the problem reduces to Subcase 1.2. ■

Now we consider the complete bipartite graphs $K_{m,n}$ where $3 \leq m \leq n$ and m is odd. Let us first give the following upper bound which can be obtained using the same strategy used in Lemma 35.

Lemma 37 If $3 \leq m \leq n$ where m is odd and n is even, then $s_f(K_{m,n}) \leq n+3$.

During the strategy given in Lemma 35, we cleaned the graph by decomposing it into $K_{2,m-3}$'s and the parity of $m-3$ allowed us to use the same searcher in the next

$K_{2,m-3}$. However in this case when m is even, the parity of $m-3$ forces us to use a different searcher for all $K_{2,m-3}$ to fast search the graph if the same strategy is used.

We know, from Theorem 34, that $m+2 = s(K_{m,n}) \leq s_f(K_{m,n})$. On the other hand, when m is odd, Lemma 26 states that $s_f(K_{m,n})$ is bounded below by $\frac{m+n}{2}$ when n is odd and by $\frac{n}{2}$ when n is even. This gives us another lower bound for $s_f(K_{m,n})$.

Next we give another upper bound which improves Lemma 37 for some m and n.

Lemma 38 If $3 \leq m \leq n$ where m is odd, then

$$s_f(K_{m,n}) \leq m + \left\lceil \frac{n}{2} \right\rceil .$$

PROOF. Let $K_{m,n}$ have bipartition $V_1 = \{v_1, v_2, \ldots, v_m\}$ and $V_2 = \{u_1, u_2, \ldots, u_n\}$.

CASE 1: $n = 4k+1$. Place m searchers on u_1 and denote them as $\sigma_2, \sigma_3, \ldots, \sigma_{m+1}$. Place a searcher, say σ_1, on v_1 and a searcher on each of u_2 and u_3, denoted as σ_{m+2} and σ_{m+3} respectively. Place a searcher on each of u_{4l+2} and u_{4l+3} for $l = 1, \ldots, k-1$. In this way we use 2 new searchers for every 4 vertices in $V_2 \backslash u_1$. In total we use $m + 1 + \frac{n-1}{2}$ searchers.

First clean u_1 so that each vertex in V_1 contains a searcher, except for v_1 which contains two searchers. Let σ_1 traverse all edges of the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_2, u_3\}$ and hence clean it. Now σ_{m+2} may slide along $u_2 v_m$ and σ_{m+3} may slide along $u_3 v_m$ and clean u_2 and u_3.

Next let σ_{m+2} slide along $v_m u_4$ and σ_{m+3} slide along $v_m u_5$. Again, let σ_1 clean all edges of the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_4, u_5\}$. This also cleans u_4 and u_5.

We clean the graph by repeating this procedure for all of u_{4l+2} and u_{4l+3} where $l = 1, \ldots, k-1$. First clean the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_{4l+2}, u_{4l+3}\}$ with σ_1. Move the searcher on u_{4l+2} along $u_{4l+2} v_m$, $v_m u_{4l+4}$ and the searcher on u_{4l+3} along $u_{4l+3} v_m$, $v_m u_{4l+5}$. Then clean the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_{4l+4}, u_{4l+5}\}$ with σ_1.

CASE 2: $n = 4k+2$. Place the searchers as in Case 1 hence $m+1+\frac{n-2}{2} = m+\frac{n}{2}$ searchers are placed on the graph. Clean all vertices in V_2 except for u_n with the same strategy used in Case 1. Observe that the only contaminated edges are the ones incident to u_n and there is a searcher located on each vertex in V_1. We let σ_2 slide along v_1u_n and clean v_1. Then let σ_3 slide along v_2u_n and clean v_2. Similarly we clean all vertices in V_1 and finally clean u_n.

CASE 3: $n = 4k+3$. Once again place the searchers as in Case 1. Place another searcher on v_m. Hence we used $m+1+\frac{n-3}{2}+1 = m+\frac{n+1}{2}$ searchers. Use the same strategy as in Case 1 to clean every vertex in V_2 except for u_{n-1} and u_n. Now there is a searcher on every vertex in V_1 except for v_1 and v_m on which there are two searchers. Then we let the two searchers on v_m slide along the only contaminated edges incident to v_m which are v_mu_{n-1} and v_mu_n, hence cleaning v_m. Finally, σ_1 cleans the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_{n-1}, u_n\}$. In this way the graph is cleaned.

CASE 4: $n = 4k$. Place a searcher on every vertex in $\{v_1, v_2, \ldots, v_{m-1}, u_1, u_2, \ldots, u_{2k}\}$ and place a second searcher, say σ_1, on v_1. Hence we use $m+\frac{n}{2}$ searchers. We let σ_1 clean the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_1, u_2, \ldots, u_{2k}\}$. Then let each searcher σ_{m+i} located on $u_i \in \{u_1, u_2, \ldots, u_{2k}\}$ slide along u_iv_m and clean $\{u_1, u_2, \ldots, u_{2k}\}$. Next let each σ_{m+i} slide along v_mu_{2k+i} and clean v_m. Now each vertex in $\{u_{2k+1}, u_{2k+2}, \ldots, u_n\}$ contains a searcher. Then we let σ_1 clean the Eulerian graph induced by $\{v_1, v_2, \ldots, v_{m-1}\}$ and $\{u_{2k+1}, u_{2k+2}, \ldots, u_n\}$. This cleans every vertex in $K_{m,n}$. ■

Let us summarize the results on the fast scarch number of a complete bipartite graph $K_{m,n}$ when m is odd. Theorem 34 and Lemmata 26, 37 and 38 imply that when m is odd, n is even and $3 \leq m \leq n$, we have

$$\max\left\{m+2, \frac{n}{2}\right\} \leq s_f(K_{m,n}) \leq \min\left\{n+3, m+\frac{n}{2}\right\}. \tag{3.2}$$

When m and n are odd and $3 \leq m \leq n$, we have

$$\max\left\{m+2, \frac{m+n}{2}\right\} \leq s_f(K_{m,n}) \leq m + \frac{n+1}{2}. \tag{3.3}$$

3.4 Cost Function

Given a graph $G = (V, E)$, let s be the number of searchers used in a search strategy to clean G. Certainly, $s(G) \leq s$. For each value of s, there is a strategy that cleans G in the minimum number of steps; that is, the minimum time. We define the minimum number of steps for each s to be $t(s)$. Certainly, $|E| \leq t(s)$.

In some real-life scenarios, the cost of a searcher may be relatively low in comparison to the cost of allowing an intruder to be free for a long period of time. Thus, it may be beneficial to use more searchers than $s(G)$. Since the minimum number of steps to clean a graph is $|E|$ (each edge must be cleaned), it will never be necessary to use more searchers than the minimum needed to clean the graph in $|E|$ steps. So we can bound the number of searchers above by $s_f(G)$, giving $s \leq s_f(G)$.

When we construct a cost function for searching we may consider the following parameters:

- α: Cost per searcher.
- β: Cost per searcher per step.
- γ: Cost per step.

Several combinations of the parameters above can be considered. Fomin and Golovach [21] introduced a cost function in the node searching problem, which is the sum of the number of searchers in every step of the node search process. For a fixed graph G, we choose to consider the following cost function in the edge searching problem:

$$C_G(s,t) = \alpha s + \beta st + \gamma t, \tag{3.4}$$

where $s(G) \leq s \leq s_f(G)$ and $t = t(s) \geq |E|$. Instead of trying to minimize s, which corresponds to (edge) searching, or to minimize t, which corresponds to fast searching, we may attempt to minimize C_G. However, in order to formulate these problems (and bound s) it is necessary to know both the search number and the fast search number.

Consider the n-star, $K_{1,n}$, where $n \geq 3$. We recall that $s(K_{1,n}) = 2$, and that $s_f(K_{1,n}) = \lceil \frac{n}{2} \rceil$. It is not difficult to see that if $2 \leq s \leq \lceil \frac{n}{2} \rceil$, then the minimum number of steps for such a strategy is $t(s) = 2n - 2s$. (The s searchers would begin at distinct leaves of the graph, and then traverse the edges to the central vertex, using s moves. They would then use 2 moves, moving out from and then back towards the central vertex, for each of $n - 2s$ pendant edges. Finally, each searcher would move out from the central vertex along one of the s remaining pendant edges).

Substituting into Equation 3.4, we obtain

$$C_{K_{1,n}}(s) = \alpha s + \beta s(2n - 2s) + \gamma(2n - 2s) = -2\beta s^2 + (\alpha + 2n\beta - 2\gamma)s + 2n\gamma. \quad (3.5)$$

But this is clearly a quadratic function in s, which has a maximum value at its critical point. We obtain its minimum value at the minimum or maximum value for s, that is when either $s = 2$ or $s = \lceil \frac{n}{2} \rceil$. (Both are possible depending on choice of α, β, and γ).

More informally, the cost of cleaning the n-stars is minimized by either treating it as an edge searching problem, when searchers are expensive and time is cheap, or by treating it as a fast searching problem when searchers are cheap and time is expensive.

Chapter 4

On the Characterization of 4-Searchable Graphs

In this chapter we return our attention to edge searching. We give results on the extremality characterizations of 4-searchable graphs. This was posed as an open problem in [37] where authors noted the difficulty of the question. The first section deals with the preliminary results and definitions. In the second section we give the main result proven in this chapter which is the complete characterization of 4-searchable biconnected outerplanar graphs. Finally, we present results related to the forbidden minor characterizations of 2-outerplanar graphs.

4.1 Preliminaries

Before giving the definitions and notation that will be used in this chapter, let us present the following result which restricts our attention to planar graphs when constructing the obstruction set for 4-searchable graphs.

Theorem 39 If $s(G) \leq 4$, then G is a planar graph.

PROOF. Let $s(G) \leq 4$. We know that $s(K_5) = 5$. Also Theorem 34 implies that $s(K_{3,3}) = 5$. Assume that G contains $K_{3,3}$ or K_5 as a minor. Since edge searching is minor closed, $s(G) \geq 5$. This is a contradiction. Hence neither $K_{3,3}$ nor K_5 can be a minor of G. Therefore G must be a planar graph. ■

For brevity we look at reduced graphs, those that are obtained from replacing every suspended path with a single edge with the same end points. As we noticed before, reduction does not change the search number.

In this chapter, all of the graphs that we consider are reduced multigraphs. Therefore, we hereafter omit the description "reduced multigraph"; reduced multigraphs will simply be called graphs.

Definition 11 A graph is said to be *outerplanar*, or *1-outerplanar*, if it can be drawn in the plane in such a way that all vertices are on the boundary of the unbounded face, i.e., the outer face. A k-outerplanar graph is defined recursively. For $k > 1$, a graph is *k-outerplanar* if there exists a planar embedding of G which has an outer face so that by removing the vertices on the outer face, we get a $(k-1)$-outerplanar graph.

Note that not every planar graph is outerplanar. For instance, K_4 is planar but not outerplanar.

Definition 12 A *tent* is a multigraph $3C_3$, i.e., each consecutive vertex of C_3 is connected with three parallel edges (Figure 4.1). A *house* is a multigraph $H = (V, E)$ where $V = V(C_4) = \{v_0, v_1, v_2, v_3\}$ and $E = E(C_4) \cup \{e_5 = v_0v_1, e_6 = v_0v_1\}$. Given a house H, the edge $e = v_2v_3$ where $\deg_H(v_2) = \deg_H(v_3) = 2$ is called the *base* of the house.

It is known that the boundary of the outer face of a biconnected outerplanar graph is a spanning cycle [52]. Assume that G is a biconnected outerplanar graph. We fix an embedding of G and label its vertices as $v_1, v_2, \ldots, v_n$ so that they consecutively

Figure 4.1: On the left: A tent. On the right: A house with thick base edge.

lie on a cycle. Here we denote the graph induced by the vertices $\{v_i, v_{i+1}, \ldots, v_j\}$ as P_{ij}. Thus P_{ij} denotes the boundary path from v_i to v_j together with all the chords between the vertices in $\{v_i, v_{i+1}, \ldots, v_j\}$.

Let v_i and v_j be two vertices of a biconnected outerplanar graph G. Denote the two graphs induced by the boundary paths joining v_i and v_j as $P_1 := P_{ij}$ and $P_2 := P_{ji}$. Hence v_i and v_j are the only common vertices of P_1 and P_2.

If neither P_1 nor P_2 have a tent or a house as a minor such that the base of the house is a chord of P_1 or P_2, then we say that G is a *generalized bipolar graph*. The vertices v_i and v_j are called the *poles* of G.

4.2 4-Searchable Outerplanar Graphs

In this section we present the characterization of 4-searchable biconnected outerplanar graphs.

Theorem 40 For a biconnected outerplanar graph G the following are equivalent:

1. $s(G) \leq 4$.
2. G does not contain any of the graphs in Figure 4.2 as a minor.
3. G is a generalized bipolar graph.

PROOF. We label the vertices of G as $v_1, v_2, \ldots, v_n$ such that they consecutively lie on a cycle. We show that *(1)⇒(2)⇒(3)⇒(1)*.

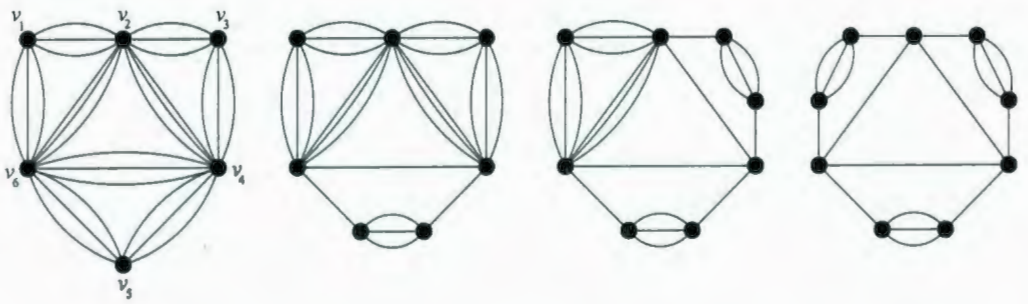


Figure 4.2: Forbidden minors for an outerplanar graph with search number 4.

(1)⇒(2). All of the graphs in Figure 4.2 have search number strictly greater than 4. Hence they cannot be minors of a graph G where $s(G) \leq 4$. They are also maximal in the sense that any minor of these graphs has search number at most 4.

Without loss of generality we may assume a monotone strategy [34]. Let us show that $s(G) > 4$ for the left most graph in Figure 4.2. The graph G has 6 vertices and 27 edges. Label the vertices as in Figure 4.2. If a search starts with cleaning a vertex v with $|N(v)| = 4$, then it will use at least 5 searchers. Since the graph is symmetric, we can start with any vertex v with $|N(v)| = 2$. Hence let the first vertex cleaned be v_1. To clean v_1 we need at least 4 searchers. When v_1 is cleaned, we must keep one searcher on each of v_2 and v_6. Hence there are two free searchers. Observe that these two free searchers do not suffice to clean any other vertex. Hence a second vertex cannot be cleaned using only 4 searchers. Thus $s(G) > 4$.

Also notice that deleting any vertex or edge from G or contracting any edge will reduce the search number to 4. Hence G is a forbidden minor.

Similar arguments suffice for the other graphs in Figure 4.2.

(2)⇒(3). Assume that G does not contain any of the graphs in Figure 4.2 as a

minor. Let $P := P_{i,j}$ be the graph induced by a maximal length boundary path with end vertices v_i and v_j such that P does not contain a tent or a house as a minor. Let v_{i-1} be adjacent to v_i and $v_{i-1} \notin P$. Also let v_{j+1} be adjacent to v_j and $v_{j+1} \notin P$. We denote the boundary path induced by $V(P) \cup V'$ for $V' \subseteq V(G)$ as $P + V'$.

Since P is maximal, $P + \{v_{i-1}\}$ will contain a tent (or a house) as a minor. Similarly $P + \{v_{j+1}\}$ will contain a tent (or a house) as a minor. Since G is outerplanar, these tents (or houses or a tent and a house) are edge disjoint.

Let P' be the the graph induced by the boundary path joining v_{j+1} and v_{i-1}, thus $P' = P_{j+1,i-1}$, and P' is disjoint from P. Then P' cannot contain any tent or house as a minor since otherwise G would have one of the graphs in Figure 4.2 as a minor by the discussion in the previous paragraph.

Assume that $P' + \{v_i\}$ has a tent or a house as a minor. Denote that minor as H. Then $v_i \in H$, since P' does not contain a house or a tent as a minor. Further, there exists a vertex $u \in \{v_{j+1}, v_{j+2}, \ldots, v_{i-2}\}$ such that $uv_i \in E(H)$. Thus v_{i-1} has no neighbor in P other than v_i, since G is outerplanar. Hence $P + \{v_{i-1}\}$ is a longer boundary path without any tent or a house as a minor, contradicting the maximality of P. Therefore $P' + \{v_i\}$ does not have a tent or a house as a minor.

Similarly $P' + \{v_j\}$ cannot have a tent or a house as a minor either. Hence $P' + \{v_i, v_j\}$ does not contain any tent or a house as a minor. Thus none of the boundary paths connecting v_i and v_j, namely, P and $P' + \{v_i, v_j\}$, contain a tent or a house.

Therefore G is a generalized bipolar graph with v_i and v_j as poles.

(3)⇒(1). Let G be a generalized bipolar graph and without loss of generality assume that v_1 and v_i are the poles of G. Let P_1 and P_2 be the two boundary paths

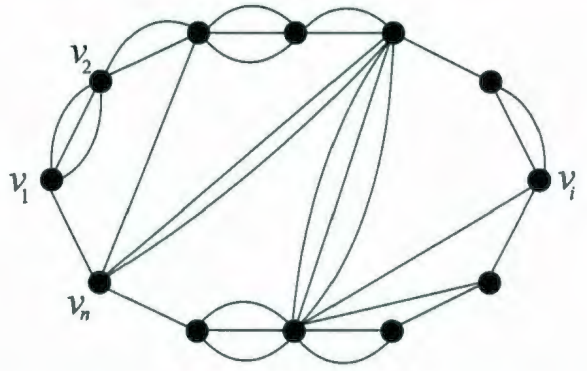


Figure 4.3: A biconnected generalized bipolar graph with poles v_1 and v_i.

connecting v_1 and v_i. First we put a searcher on each of v_1 and v_n. Denote them as σ_1 and σ_2. Clean all the edges between v_1 and v_n by a third searcher, say σ_3. During the search we always keep σ_1 on P_1 and σ_2 on P_2 as guards. The other two searchers are used to clean boundary edges, edges parallel to the boundary edges and cross chords connecting a vertex in P_1 with a vertex in P_2. By the outerplanarity of G, the cross chords can always be cleaned using a searcher while the others are used to guard its end vertices.

We show how to clean P_1 using σ_1, σ_3 and σ_4 where the vertices are cleaned consecutively. Similarly one cleans P_2 using σ_2, σ_3 and σ_4. Assume that $v_1, v_2, \ldots, v_{j-1}$ are cleaned and let σ_1 be on v_j.

CASE 1: There are no cross chords incident to v_j.

If v_j has at most two contaminated edges incident to it, say $e_1 = v_j v_{j+1}$ and $e_2 = v_j v_k$, we put σ_3 on v_j and we clean v_j by sliding σ_1 along e_1 and σ_3 along e_2. If $k = j + 1$, both searchers are on v_{j+1} and we proceed to clean v_{j+1}. If $k = j + 2$, we put σ_4 on v_{j+1} and clean all the edges between v_{j+1} and v_{j+2} by σ_4. There are no cross chords incident to v_{j+1} due to the outerplanarity. Hence v_{j+1} is cleaned. Else if $k \geq j + 3$, since P_1 does not contain a house, the

boundary path connecting v_{j+1} and v_k can be cleaned with two searchers for which we use σ_1 and σ_4.

If v_j has more than two contaminated edges all of which are incident to one of v_{j+1} or v_k where $k \geq j+1$, the path connecting v_j and v_k and all edges induced by $\{v_j, v_{j+1}, \ldots, v_k\}$ can be cleaned by σ_1, σ_3 and σ_4 since there are no houses or tents contained as a minor.

Otherwise, assume that v_j has at least three contaminated edges incident to it and that these edges are incident to at least three distinct vertices on P_1 other than v_j. Let the last vertex be v_k so that $e = v_j v_k$ is contaminated and $k \geq j+3$. The only boundary edges in P_{jk} that may have at least two edges parallel to them are $e_1 = v_j v_{j+1}$ and $e_2 = v_{k-1} v_k$, since otherwise P_1 would contain a house as a minor. Hence we put σ_3 on v_j as a guard and let σ_1 and σ_4 clean P_{jk} and all edges incident to v_j.

CASE 2: There is a cross chord $e = v_l v_j$ incident to v_j where $v_l \in P_2$.

We clean P_2 until σ_2 reaches v_l. We let σ_1 stay on v_j and σ_2 stay on v_l as guards and we clean e and all edges parallel to it by σ_3.

We repeat this procedure until either σ_1 reaches v_i and σ_2 reaches v_{i-1}, or, σ_2 reaches v_i and σ_1 reaches v_{i+1}. Finally we clean the remaining contaminated edges, i.e., those that are between v_i and v_{i-1} in the former case, and those that are between v_i and v_{i+1} in the latter case. Hence the graph is cleaned using 4 searchers.

■

4.3 On 4-Searchable 2-Outerplanar Graphs

In the previous section we have given the obstruction set for outerplanar graphs. Here we will give the partial results for the obstruction set for 2-outerplanar graphs. Our intention is to generalize the character of such a set for any k since any finite graph is k-outerplanar for some k.

As noted before, for a fixed k the set of forbidden minors of k-searchable graphs is finite. Nevertheless, the construction of the elements of the obstruction set and its size are unknown. Furthermore, it is shown that from a finite description of a minor closed family of graphs, there is no general algorithm to compute the obstruction set [20].

Before we state our next theorem, we introduce a notation for 2-outerplanar graphs G. First fix a planar embedding of G. One can deduce from Definition 11 that the vertex set V of a biconnected 2-outerplanar graph G can be partitioned into two sets V_1 and V_2 so that:

(1) the elements of V_1 induce an outerplanar graph, and,

(2) the elements of V_2 are those that are in the boundary face of G.

According to this definition we denote the class of biconnected 2-outerplanar graphs G with partition sets V_1 and V_2 where $n_1 = |V_1|$ and $n_2 = |V_2|$ as $O[n_1, n_2]$. Note that a graph may belong to two different classes depending on its embeddings. Here out of all possible embeddings in $O[n_1, n_2]$ of a 2-outerplanar graph, we choose the one which minimizes n_1.

Let F denote the set of biconnected forbidden minors for 2-outerplanar 4-searchable graphs. Note that F does not have any element of order 3 or less, thus $F \cap O[1, 2] = \emptyset$.

The next theorem shows that there is a unique graph that is a forbidden minor in each of the first three classes of interest.

Theorem 41 The graphs G_1, G_2 and G_3 in Figure 4.4 are elements of F, where

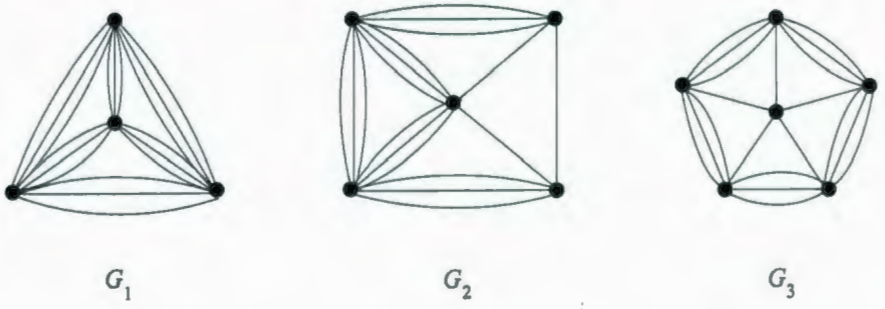


Figure 4.4: Graphs G_1, G_2 and G_3.

F is the set of biconnected forbidden minors for 2-outerplanar 4-searchable graphs. Furthermore, they satisfy the following equations:

$$\{G_1\} = F \cap O[1,3], \tag{4.1}$$

$$\{G_2\} = F \cap O[1,4], \tag{4.2}$$

$$\{G_3\} = F \cap O[1,5]. \tag{4.3}$$

PROOF. We will use analysis by cases to show that G_1, G_2 and G_3 are forbidden minors for a 4-searchable 2-outerplanar graph.

The symmetry of G_1 allows us to clean the vertices in any order. Since we need at least 5 searchers to clean a first vertex, G_1 is not 4-searchable. Deleting any vertex or contracting any edge would result in a graph with 3 vertices that is clearly 4-searchable. Also, deleting at least one edge we obtain a 4-searchable graph. From this argument it follows that G_1 is the unique element in $F \cap O[1,3]$, since any minor of G_1 in $O[1,3]$ is 4-searchable and any graph in $O[1,3]$ that is not 4-searchable will contain G_1 as a minor.

We can quickly analyze all search strategies for G_2 that use only 4 searchers and see that after cleaning a first vertex all searchers are stuck. Contracting any edge would result in a minor of G_1 or a 4-searchable graph. Similarly, deleting an edge

or a vertex would give a graph with search number at most 4. Hence $G_2 \in F$. To see that any graph $G \in O[1,4]$ that is not 4-searchable contains either G_1 or G_2 as a minor we consider the following cases:

CASE 1: Consecutive vertices on the boundary are connected by at least three parallel edges.

If the vertex v_0 that is not on the boundary has at most 2 neighbors, then G is 4-searchable. If v_0 has at least 3 neighbors, then G is either 4-searchbale or contains G_1 or G_2 as a minor.

CASE 2: At least two consecutive vertex on the boundary are connected with at most two parallel edges. Verification is done in a similar way to CASE 1.

Thus $\{G_2\} = F \cap O[1,4]$.

To show the last equality we consider four cases for an arbitrary graph $G \in O[1,5]$:

CASE 1: There are no chords between non-consecutive vertices.

CASE 1.1: All consecutive vertices on the boundary are connected with at most 2 parallel edges.

We give a search strategy that cleans all such graphs by using 4 searchers only. Place two searchers σ_1 and σ_2 on the center vertex v_0 and one searcher on each of two adjacent boundary vertices v_1 and v_2. Clean all edges between v_0, v_1 and v_2 by σ_1 keeping all other searchers on their initial places. Remove σ_1 and place it on v_2. Then let the other searcher located on v_2 slide along the boundary edge connecting it to v_3. Since there are at most two such vertices, there is at most one contaminated edge incident to v_2. Clean this edge by sliding σ_1 along it. Now σ_1 can clean all edges between v_3 and v_0. We repeat this until we reach v_1. Hence all such graphs are 4-searchable.

CASE 1.2: At least two consecutive vertex on the boundary are connected with at least 3 parallel edges, at least 2 others are connected with at most two parallel edges.

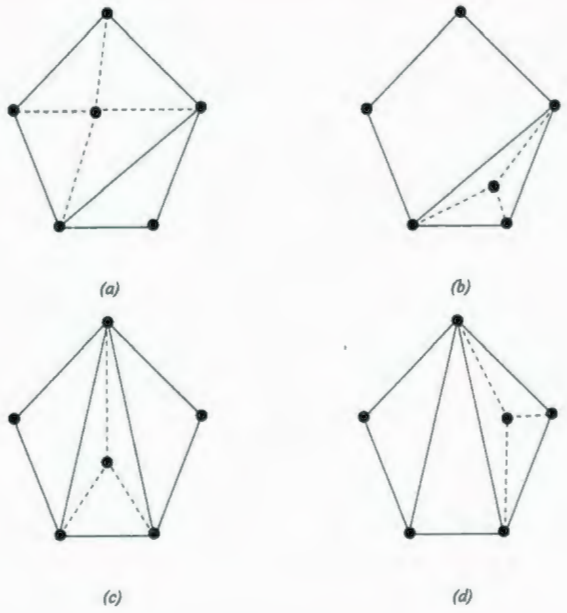


Figure 4.5: Subcases considered in Theorem 41.

Case analysis similar to the previous cases shows that all graphs in this subcase are either 4-searchable or they contain G_1 or G_2 as a minor.

CASE 1.3: All consecutive vertices in the boundary are connected with at least 3 parallel edges.

Observe that G_3 belongs to this case. It is not possible to clean this graph using 4 searchers. Also deleting any vertex or contracting any edge will result in a 4-searchable graph. If we delete any edge on the boundary we can clean the graph with 4 searchers using the strategy given in CASE 1.1. Hence $G_3 \in F$. Observe that any non 4-searchable graph G belonging to this subcase that is not a minor of G_3 is a supergraph of G_3 or it contains G_1 or G_2 as a minor. Hence G_3 is the only such graph that belongs to this subcase.

CASE 2: There are chords between non-consecutive vertices.

There are at most two such chords. In Figure 4.5 all such possible graphs are

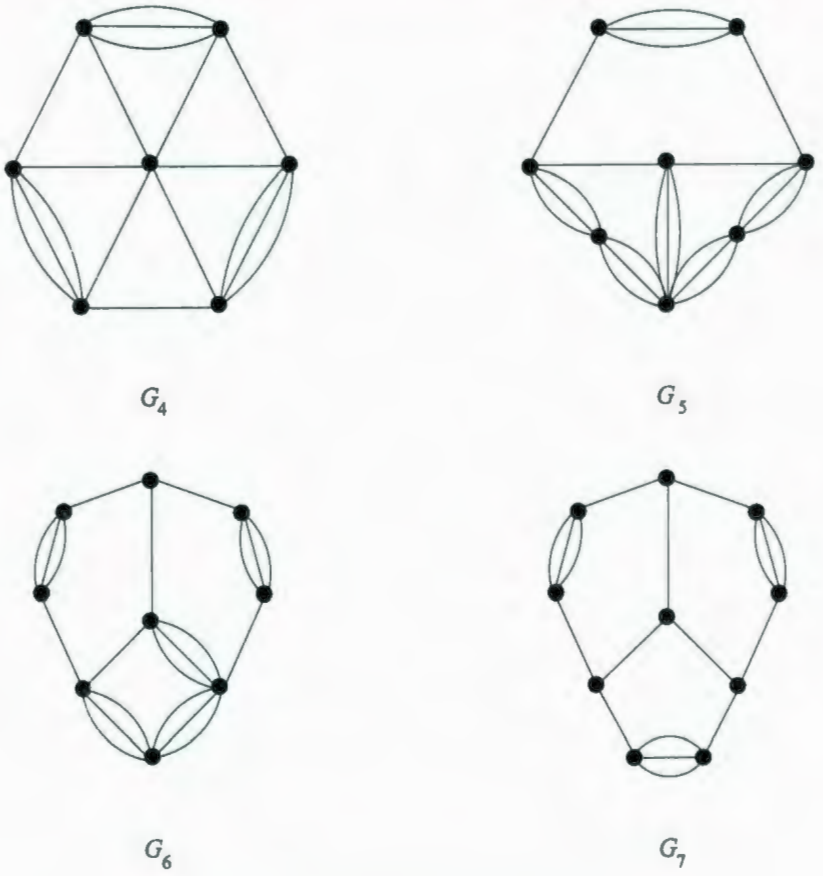


Figure 4.6: Graphs G_4, G_5, G_6 and G_7.

shown where the dotted edges may or may not exist. Although in Figure 4.5 for simplicity we put a single edge between any two consecutive vertex, they may be connected with any number of parallel edges. Again, a long case analysis similar to the previous ones reveals that all graphs that belong to these subcases are either 4-searchable or they contain G_1 or G_2 as a minor.

Therefore $\{G_3\} = F \cap O[1,5]$. ■

Using a similar discussion as in the proof of Theorem 41 we can show that the graphs G_5, G_6 and G_7 given in Figure 4.6 are in the set F. The verification that these

graphs are indeed forbidden minors is done by case analysis which becomes tediously long as the order of the graph increases. Although it seems that these graphs are the unique forbidden minors in their respective classes, we do not prove this, because of the growth of the number of cases to consider as the number of chords between non-consecutive vertices increases.

To close this chapter we give the other graphs in F that we have found so far. By analyzing these graphs we point out the difficulty of the characterization of graphs in $F \cap O[n_1, n_2]$ as n_1 or n_2 increase. Thus we do not claim that this is the complete characterization of F.

We collected the figures so that $G, H \in F$ are given in the same figure where $G \in O[n_1, n_2]$ and $H \in O[n_1, n_3]$. Here n_2 and n_3 are not necessarily equal.

Figure 4.7 shows five graphs that belong to $F \cap O[2, k], 3 \leq k$. We give some of the graphs in $F \cap O[3, k], 3 \leq k$ in Figure 4.8. The last figure in this section, Figure 4.9, shows graphs in $F \cap O[4, k], 4 \leq k$.

Although some of them share some common features, the elements of F are irregular in general. Notice that all forbidden minors we have given in Figure 4.7 induce a connected graph when the vertices on the boundary face were deleted. This also holds for most of the graphs in Figure 4.8. In contrast the graph G_{18} in Figure 4.8 is a forbidden minor in $O[3, 9]$ which induces an independent set of vertices when the outerface is deleted.

Observe that most of the forbidden minors given here are highly symmetric. On the other hand some of them, such as G_{12} and G_{15}, are not.

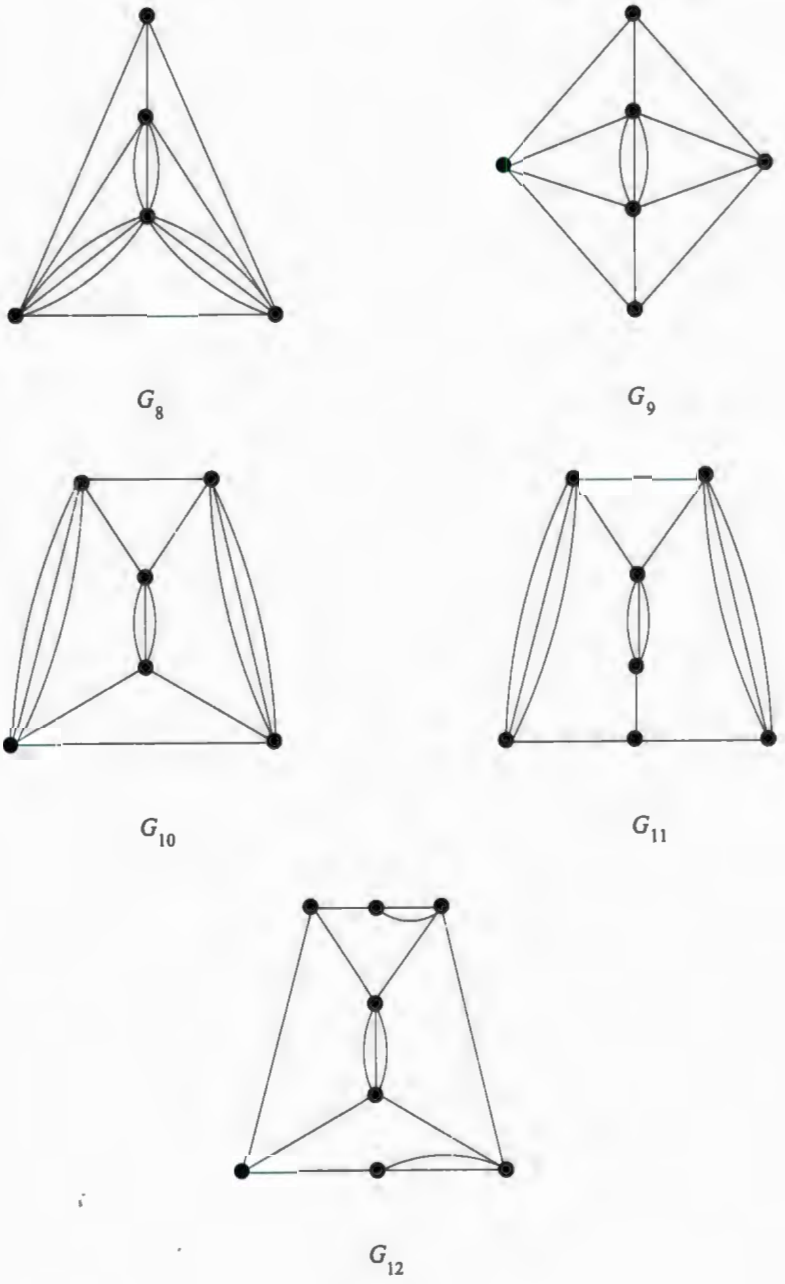


Figure 4.7: Forbidden minors in $O[2,k], 3 \leq k$.

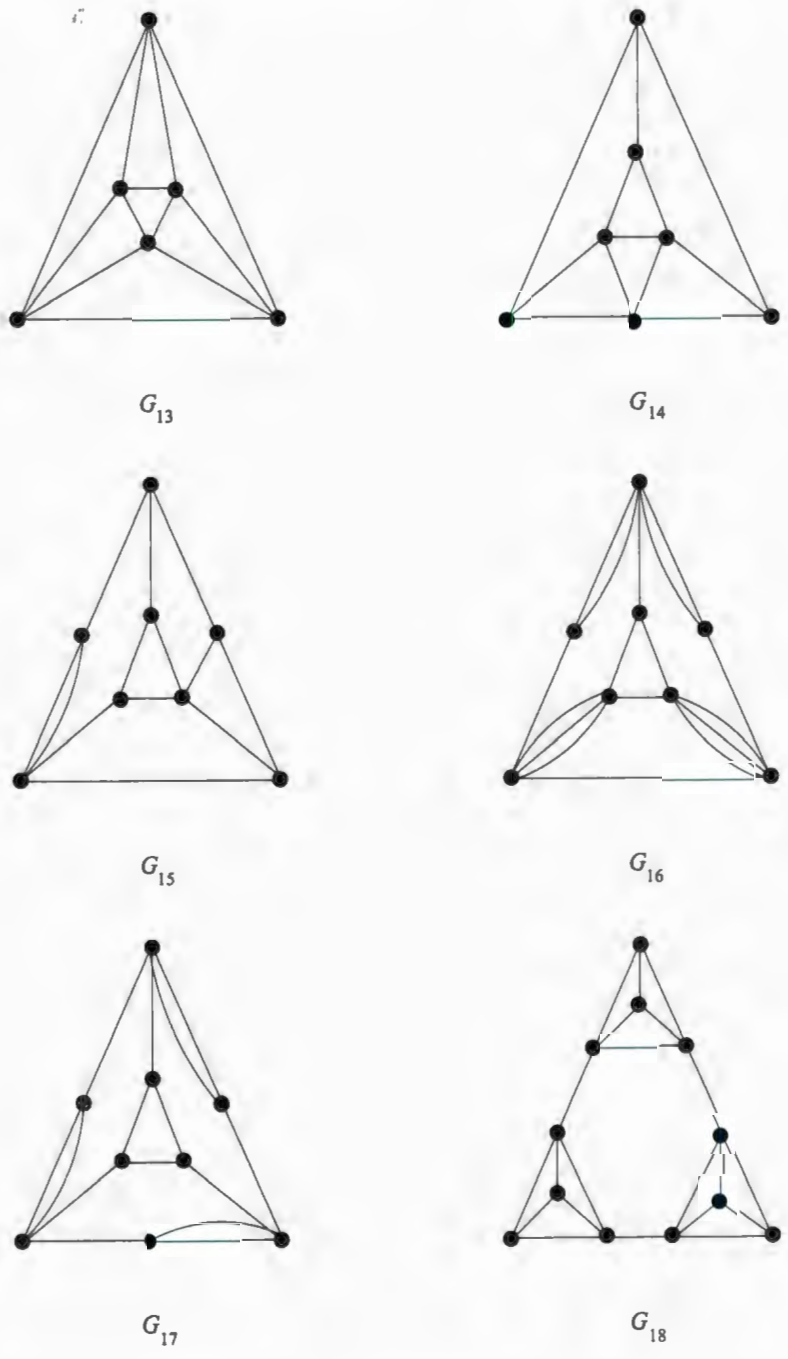
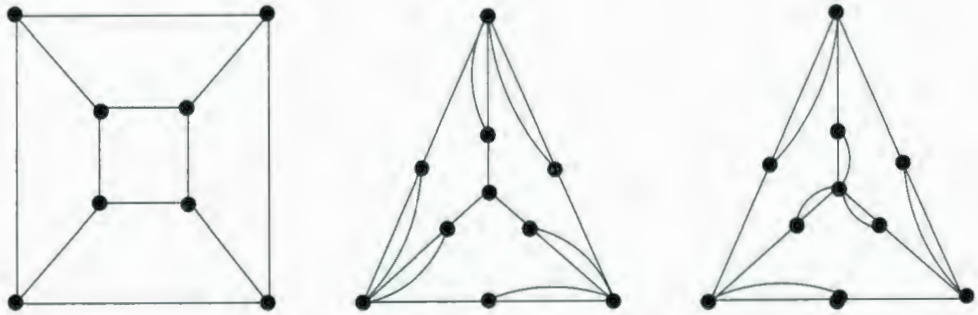


Figure 4.8: The graphs $G_{13}, G_{14}, \ldots G_{18}$.



G_{19} G_{20} G_{21}

Figure 4.9: The graphs G_{19}, G_{20} and G_{21} in $[4, k]$ for $4 \leq k$.

Chapter 5

Conclusion and Future Directions

In the first section we summarize the main results presented in this work and pose related open problems and some future directions which arise from our work. The final section is devoted to a future direction which estimates the search number of circulant graphs of prime order.

5.1 Conclusion and Open Problems

The research presented in this dissertation covers complexity results, extremality characterizations and comparisonal bounds for edge searching and two of its variations. An open problem of edge searching is partially solved.

In Chapter 2, we introduce a new weighted version of the edge searching problem. Our motivation is that there may be networks where links have different capacities or importance factors. In this setting decontamination is not the same for all edges.

For any graph G, Theorem 13 implies that the pathwidth and the weighted edge search number may differ by at most 2. This result improves the bound in Theorem 11. Furthermore, since $pw(G) \leq s(G) \leq ws(G) \leq pw(G) + 2$, if $s(G) = pw(G) + 2$, we have $s(G) = ws(G)$. From this result a characterization of graphs for which any

weight distribution would give the same search number and weighted search number can be obtained. For instance; when $G = K_{3,3}$, we have $s(G) = 5$ and $pw(G) = 3$, thus $ws(G) = 5$ for any edge weights for G.

Theorem 16 in Section 2.3 gives equivalent characterizations of 2-searchable weighted graphs. It identifies the forbidden graphs containment of which prevents the weighted search number to be at most two and explains how all such graphs are structured.

The characterization of unweighted graphs G for which the search number is at most 3 is done in [37] and it is more complicated than the characterization of graphs G for which $s(G) \leq 2$. One possible future aim is to find a characterization theorem for 3-searchable weighted graphs.

The major result of Chapter 2 is Theorem 22, namely, for any weighted graph G, we have $ws(G) = mws(G)$ [57]. Thus for every weighted graph G there exists a monotonic weighted search that uses the same number of searchers as a non-monotonic weighted search. Combining Theorem 22 and Theorem 4 we get

$$ws(G) = iws(G) = mws(G).$$

The main implication of Theorem 22 is the NP-completeness of the weighted searching problem. There are various complexity problems arising from here. Whether the problem is still NP-complete when restricted to trees or planar graphs remains open. A related problem is to to find out if the problem is polynomially solvable for special classes of graphs, such as chordal graphs.

Secondly, monotonicity implies that one can always use a monotonic strategy without any increase on the number of searchers. This is important since when considering edge searching problems it is most often simpler to work with a monotonic strategy. Monotonicity is also advantageous when there are costs related to moving along an edge and to the number of moves. If sliding along an edge is very costly, a monotonic strategy would be desired.

There are various future directions arising from weighted search. One version of a weighted search may consider searchers having different cleaning capacities. In addition to edges, we can also put weights on vertices and combine weighted edge search and weighted node search.

In Chapter 3 we introduce fast searching as a variant of edge searching. We demonstrate fast search strategies with several examples. We show that a fast search strategy for a tree can be computed in linear time. We also propose a cost function that generalizes the searching problem.

We prove several results on the fast search number of complete bipartite graphs. When m is even and $6 \leq m \leq n$, we have $s_f(K_{m,n}) = m + 3$. When m is odd and $3 \leq m \leq n$ the results are summarized by Equations 3.2 and 3.3. The gaps in Equations 3.2 and 3.3 may be large depending on m and n. The fast search number of $K_{m,n}$ can be further studied to revise these lower and upper bounds. It remains open to reduce these gaps to a small constant which does not depend on m and n.

It would also be interesting to investigate the fast search number of other classes of graphs, such as Cartesian products. In particular, the bounds given in Example 8 for grids may be improved. Besides the fact that much is known about the search number of these graphs, knowing the fast search number will allow us to examine cost functions as well.

Theorem 31 implies that the FAST SEARCHING problem is linear when restricted to trees [18]. Theorem 30 states that the fast search number of a tree can be found from its degree sequence.

We have the following conjecture.

Conjecture 1 FAST SEARCHING problem is NP-complete for general graphs.

We can consider variants of fast searching. In fast searching each edge may only be traversed once. In another model, one can consider the case, where for a given

positive integer k, each edge of a previously specified subset of edges may be traversed at most k times. This scenario is applicable, for instance, to graphs with a small sized cut set which divides the graph into two almost equal size parts. Another variant of the problem may specify the searchers' start vertices.

Chapter 4 is devoted to characterization of forbidden minors for a 4-searchable graph. The characterization for $s(G) \leq k$ is not known for any fixed $k \geq 4$ [37]. Theorem 40 gives the complete list of forbidden graphs and characterizes all biconnected 4-searchable outerplanar graphs. However we only have partial results on the obstruction set for a 2-outerplanar 4-searchable graphs. One future direction is to complete this set.

We further have the following conjecture.

Conjecture 2 If $G \in O[1,k]$ and $10 \leq k$, then G is not a forbidden minor for a 4-searchable 2-outerplanar graph. Equivalently;

$$F \cap O[1,k] = \emptyset, \forall\, k \geq 10.$$

It would be interesting to know whether or not there are forbidden minors for biconnected k-outerplanar 4-searchable graphs for $k \geq 3$ that do not contain any graph in the obstruction set of outerplanar or 2-outerplanar 4-searchable graphs.

Once the biconnected forbidden minors are known, to complete the characterization it remains to find out where the graphs with a smaller search number can be attached. This is much simpler than finding the obstruction set.

5.2 A Future Direction

As a future direction of edge searching we consider edge searching of circulant graphs of prime order and state our conjecture. Let p be a prime number and consider the

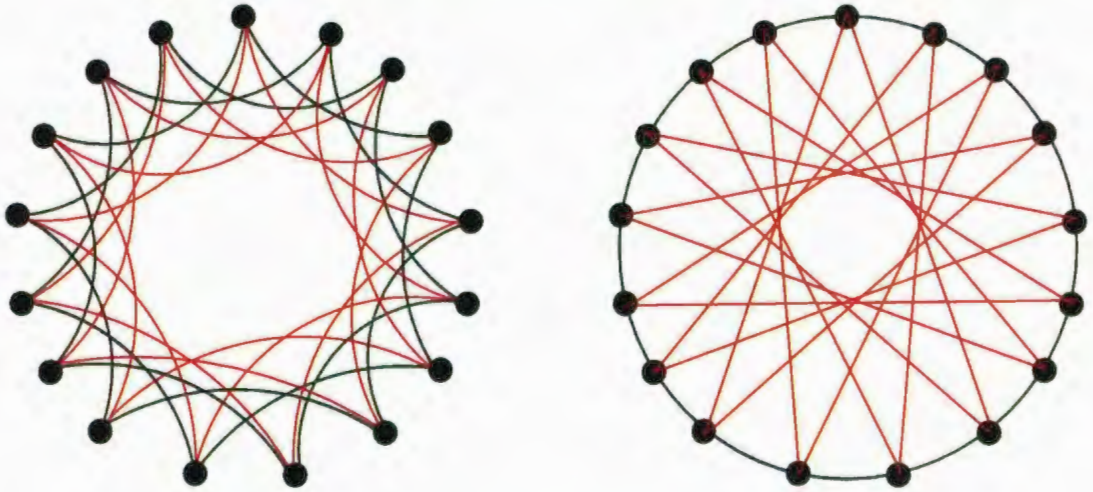


Figure 5.1: The circulant graphs circ(17; 3, 4) and circ(17; 1, 7)

circulant graph $\text{circ}(p, \mathcal{S})$ where $\mathcal{S} \subseteq (\mathbb{Z}_p \backslash \{0\})$. Thus the computations are done modulo p.

Notice that $\text{circ}(p, \mathcal{S})$ is a Hamiltonian cycle when $|\mathcal{S}| = 2$. Thus $s(\text{circ}(p, \mathcal{S})) = 2$ if $|\mathcal{S}| = 2$ and $3 \leq p$. Nevertheless, the calculation of the search number of $\text{circ}(p, \mathcal{S})$ gets complicated rapidly as the size of the set $\mathcal{S}$ increases.

For brevity, we denote a circulant graph $G = \text{circ}(p, \mathcal{S})$ with connection set $\mathcal{S} = \{a, -a, b, -b\}$ as $\text{circ}(p; a, b)$, where $1 \leq a < b \leq \frac{p-1}{2}$.

Consider $\text{circ}(p; a, b)$. Place $2b$ searchers on the following vertices: $v_1, v_2, \ldots, v_b$ and $v_{n-b+1}, v_{n-b+2}, \ldots, v_n$. Label these searchers as $\sigma_1, \sigma_2, \ldots, \sigma_{2b}$ respectively. Place σ_{2b+1} on v_1 and clean all the edges with both end vetices in $\{v_{n-b+1}, v_{n-b+2}, \ldots, v_n, v_1, v_2, \ldots, v_b\}$. The only contaminated edge incident to v_1 is v_1v_{b+1}, thus let σ_1 slide along this edge and clean v_1. Next let σ_1 clean all the edges with both end vetices in $\{v_{n-b+1}, v_{n-b+2}, \ldots, v_n, v_2, \ldots, v_b, v_{b+1}\}$. Now σ_2 can slide along v_2v_{b+2} and clean v_2. We clean $v_1, v_2, \ldots, v_b$ in the same way. We repeat this shifting of searchers on $v_1, v_2, \ldots, v_b$ to $v_{b+1}, v_{b+2}, \ldots, v_{2b}$ for every group of b consequtive vertex and clean

the whole graph. Thus we have the following result:

Theorem 42 If p is a prime number and $1 \leq a < b \leq \frac{p-1}{2}$, then

$$s(\text{circ}(p; a, b)) \leq 2b + 1.$$

In order to consider isomorphic circulant graphs, we use multiplication in $\mathbb{Z}_p$. Let $f : \{1, 2, \ldots, p\} \to \{1, 2, \ldots, p\}$ so that $f(n) = (n-1)a + 1$ where $1 \leq a < \frac{p-1}{2}$. It is a simple observation that f is an isomorphism between $\text{circ}(p; a, b)$ and $\text{circ}(p; 1, c)$ where $1 \leq a < b \leq \frac{p-1}{2}$ and $c = ba^{-1}$. Here a^{-1} denotes the multiplicative inverse of a with respect to the cyclic group $(\mathbb{Z}_p, \cdot)$. Thus for every circulant graph G with connection set of size 4, there exists an integer c where $2 \leq c \leq \frac{p-1}{2}$ such that G is isomorphic to $\text{circ}(p; 1, c)$.

From this argument it follows that the graphs in Figure 5.1, $\text{circ}(17; 3, 4)$ and $\text{circ}(17; 1, 7)$, are isomorphic and thus they have the same search number and this number is bounded above by 9 by Theorem 42.

Furthermore, we can show similarly that for any $k \geq 1$,

$$\text{circ}(p; 1, c) \simeq \text{circ}(p; k, ck).$$

Thus when analyzing the circulant graphs $\text{circ}(p; a, b)$, it is sufficient to look at circulants $\text{circ}(p; k, ck)$ for every $k \geq 1$ and $2 \leq c \leq \frac{p-1}{2}$.

The following conjecture gives a bound on the product of an element of $\mathbb{Z}_p$ and a positive integer less than or equal to the ceiling of the root of p.

Conjecture 3 For every prime p and every integer $i = 1, 2, \ldots, \frac{p-1}{2}$, there exists an integer j, $1 \leq j \leq \lceil \sqrt{p} \rceil$ such that either

$$ij \leq \lceil \sqrt{p} \rceil \pmod{p},$$

or,

$$p - ij \leq \lceil \sqrt{p} \rceil \pmod{p}.$$

We have a Maple code that mininimizes the maximum desired product. This ongoing code shows that Conjecture 3 holds for up to the 6000th prime. Thus by Theorem 42 and the isomophisms we have given, the following is true for the first 6000 primes:

$$s(\text{circ}(p, S)) \leq 2\lceil\sqrt{p}\rceil + 1 \tag{5.1}$$

for every circulant graph, $\text{circ}(p, \mathcal{S})$, where $\mathcal{S} \subseteq (\mathbb{Z}_p \backslash \{0\}), |\mathcal{S}| \leq 4$.

Bibliography

[1] S. Alpern and S. Gal, *The theory of search games and rendezvous*, International Series in Operations Research and Management Science 55, Kluwer Academic Publishers, Boston, 2003.

[2] S. Arnborg, D.G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Alg. Disc. Meth., 8 (1987) pp. 277–284.

[3] B. Alspach, *Cayley Graphs* in *Topics in Algebraic Graph Theory*, Edited by L. W. Beineke and R. J. Wilson, Cambridge University Press, 2004.

[4] B. Alspach, D. Dyer, D. Hanson, and B. Yang, *Lower bounds on edge searching*, Proceedings of the 1st International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE'07), Lecture notes in Computer Science, Vol. 4614, Springer-Verlag (2007) pp. 516–527.

[5] B. Alspach, D. Dyer, D. Hanson, and B. Yang, *Time constrained graph searching*, Theoretical Computer Science - Special Issue on Graph Searching, Vol. 399, Issue: 3 (2008) pp. 158–168.

[6] J. Barát, *Directed path-width and monotonicity in digraph searching*, Graphs Combin., 22 (2006) pp. 161-172.

[7] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, *Capture of an intruder by mobile agents.* Proceedings of the 14th annual ACM symposium on Parallel algorithms and architectures (2002) pp. 200–209.

[8] L. Barrière, P. Fraigniaud, N. Santoro and D. Thilikos, *Searching is not Jumping*, Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG'03), Lecture notes in Computer Science, Vol. 2880, Springer-Verlag (2003) pp.34–45.

[9] L. Barrière, P. Fraigniaud, N. Santoro and D. Thilikos, *Connected and Internal Graph Searching*, URL: citeseer.ist.psu.edu/636007.html.

[10] A. Berarducci and B. Intrigila, *On the Cop Number of a Graph*, Advances in Appl. Math. 14 (1993) pp. 389–403.

[11] D. Bienstock and P. Seymour, *Monotonicity in Graph Searching*, Journal of Algorithms 12 (1991) pp. 239–245.

[12] H. L. Bodlaender and D. M. Thilikos, *Computing small search numbers in linear time*, Technical Report No. UU-CS-1998-05, Dept. of Computer Science, Utrecht University (1998).

[13] H. L. Bodlaender and T. Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*, J. Algorithms, 21(2) (1996) pp. 358–402.

[14] R. L. Breisch, *An intuitive approach to speleotopology*, Southwestern Cavers 6 (1967) pp. 72–78.

[15] L. S. Chandran and T. Kavitha, *The treewidth and pathwidth of hypercubes*, Discrete Mathematics 306 (2006) pp. 359–365.

[16] F. Chung, *On the cutwidth and the topological bandwidth of a tree*, SIAM J. Algebraic Discrete Methods 6 (1985) pp. 268–277.

[17] D. Dyer, *Sweeping Graphs and Digraphs*, PhD Thesis, Simon Fraser University, 2004.

[18] D. Dyer, B. Yang and Ö. Yaşar, *On the Fast Searching Problem*, Lecture Notes in Computer Science 5034, Springer-Verlag (2008) pp. 143-154.

[19] J. A. Ellis, I. H. Sudborough, J. S. Turner, *The Vertex Separation and Search Number of a Graph*, Information and Computation 113 (1994) pp. 50–79.

[20] M. Fellows, M. A. Langston, *On search, decision and efficiency of polynomial time algorithms*, In 21th ACM Symp. on Theory of Computing (1989) pp. 501–512.

[21] F. V. Fomin and P. A. Golovach, *Graph searching and interval completion*, SIAM Journal on Discrete Mathematics 13 (2000) pp. 454–464.

[22] F. V. Fomin, P. Heggernes and J. A. Telle, *Graph searching, elimination trees, and a generalization of bandwidth*, Algorithmica 41 (2004) pp. 73–87.

[23] F. V. Fomin and D. M. Thilikos, *An annotated bibliography on guaranteed graph searching*, Theoretical Computer Science, Special Issue on Graph Searching, submitted on Nov. 2007.

[24] P. A. Golovach, *Equivalence of two formalizations of a search problem on a graph* (in Russian), Vestnik Leningrad. Univ. Mat. Mekh. Astronom. (1989) pp. 10–14. Translation in Vestnik Leningrad Univ. Math. 22, no. 1, (1989) pp. 13–19.

[25] L. J. Guibas, Jean-Claude Latombe, Steven M. LaValle, David Lin, Rajeev Motwani, *A Visibility-Based Pursuit-Evasion Problem*, International Journal of Computational Geometry and Applications 9, No: 4/5 (1999) pp. 471–493.

[26] G. Hahn, F. Laviolette, N. Sauer and R.E. Woodrow, *On cop-win graphs*, Discrete Math. 258 (2002), pp. 27-41.

[27] G. Hahn and G. MacGillivray, *A note on k-cop, l-robber games on graphs*, Discrete Math. 306 (2006) pp. 2492–2497.

[28] F. Harary and G. Prins, *The block-cutpoint-tree of a graph*, Publ. Math. Debrecen 13 (1966) pp. 103–107.

[29] T. Johnson, N. Robertson, P. Seymour and R. Thomas, *Directed tree-width*, Journal of Combinatorial Theory Series B, 82 (2001) pp. 138–154.

[30] V. İşler, S. Kannan, and S. Khanna, *Locating and capturing an evader in a polygonal environment,* Springer Tracts in Advanced Robotics, 17 (2005) pp. 251–28.

[31] N. G. Kinnersley, *The Vertex Separation Number of a graph equals its path-width*, Information Processing Letters 42 (1992) pp. 345–350.

[32] L. M. Kirousis and C. H. Papadimitriou, *Searching and Pebbling*, Theoretical Computer Science 47 (1986) pp. 205–218.

[33] L. M. Kirousis and C. H. Papadimitriou, *Interval Graphs and Searching*, Disc. Math. 55 (1985) pp. 181–184

[34] A. S. LaPaugh, *Recontamination does not help to search a graph*, Journal of ACM, 40 (1993) pp. 224–245.

[35] F. S. Makedon, C. H. Papadimitriou and I. H. Sudborough, *Topological Bandwidth*, SIAM Journal Algebraic Discrete Methods 6 (1985) pp. 418–444.

[36] F. Makedon and H. Sudborough, *Minimizing width in linear layout*, Lecture notes in Computer Science, Vol. 154, Springer-Verlag (1983) pp.478–490.

[37] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, C. H. Papadimitriou, *The Complexity of Searching a Graph*, Journal of ACM, 35 (1988) pp. 18–44.

[38] B. Monien and I. H. Sudborough, *Min-Cut is NP-complete for edge weighted trees*, Theoretical Computer Science 58 (1988) pp. 209–229.

[39] R. H. Möhring, *Graph Problems Related to Gate Matrix Layout and PLA Folding*, in G. Tinnhofer et al. eds., Computational Graph Theory, Springer, Computing Supplementum 7 (1990) pp. 17–32.

[40] R. J. Nowakowski and P. Winkler, *Vertex-to-vertex Pursuit in a Graph*, Discrete Math. 43 (1983) pp. 235–239.

[41] R. J. Nowakowski, *Search and sweep numbers of finite directed acylic graphs*, Discrete Applied Mathematics, 41 (1993) pp. 1–11.

[42] T. D. Parsons, *Pursuit-evasion in a graph*, Theory and Applications of Graphs, Lecture Notes in Mathematics, Springer-Verlag (1976) pp.426–441.

[43] T. D. Parsons, *The search number of a connected graph*, in Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory and Computing, vol. XXI of Congress. Numer. Utilitas Math. (1978) pp. 549–554.

[44] S.-L. Peng, C. Y. Tang, M.-T. Ko, C.-W. Ho and T.-s. Hsu, *Graph Searching on Some Subclasses of Chordal Graphs*, Algorithmica 27(3) (2000) pp. 395–426.

[45] N. N. Petrov, *A problem of pursuit in the absence of information on the pursued*, Differentsial'nye Uravneniya 18 (1982) pp. 1345–1352.

[46] D. Rautenbach, *Lower bounds on treespan*, Inf. Process. Lett. Vol. 96 No. 2 (2005) pp. 67–70.

[47] N. Robertson and P. D. Seymour, *Graph minors. I. Excluding a Forest*, Journal of Combinatorial Theory, Series B 35 (1983) pp. 39–61.

[48] N. Robertson and P. D. Seymour, *Graph minors - a survey*, in: I. Anderson (Ed.), Surveys in Combinatorics, Cambridge University Press, Cambridge (1985) pp. 153–171.

[49] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic Aspects of Tree-Width*, Journal of Algorithms 7 (1986) pp. 309–322.

[50] D. M. Thilikos, *Algorithms and Obstructions for linear-width and related search parameters*, Discrete Appl. Math. 105 (2000) pp. 239–271.

[51] A. Quilliot, *Thèse d'Etat*, Université de Paris VI, 1983.

[52] D. B. West, *Introduction to Graph Theory*, Prentice Hall, 1996.

[53] H. Whitney, *Congruent graphs and the connectivity of graphs*, Amer. J. Math. 54 (1932) pp. 150–168.

[54] B. Yang and Y. Cao, *Digraph Strong Searching: Monotonicity and Complexity*, in AAIM (2007) pp. 37–46.

[55] B. Yang, D. Dyer, and B. Alspach, *Sweeping graphs with large clique number*, Proceedings of 15th International Symposium on Algorithms and Computation (ISAAC'04), R. Fleischer and G. Trippen Eds., Lecture Notes in Computer Science, Vol. 3341, Springer-Verlag (2004) pp. 908–920.

[56] B. Yang, D. Dyer, and B. Alspach, *Sweeping graphs with large clique number*, To appear in Discrete Mathematics.

[57] Ö. Yaşar, D. Dyer, D. A. Pike and M. Kondratieva, *Weighted Edge Searching*, To appear in Discrete Applied Mathematics.

Index