Searching Graphs with Large Clique Number

Boting Yang, Danny Dyer and Brian Alspach

Technical Report TR-CS 2006-8
April, 2006

University of Regina
Regina, Saskatchewan, CANADA
S4S 0A2

ISBN 0-7731-0577-8 (print)
ISBN 0-7731-0578-6 (on-line)

Searching Graphs with Large Clique Number

Boting Yang[1]
Department of Computer Science
University of Regina

Danny Dyer
Department of Mathematics
Simon Fraser University

Brian Alspach[2]
Department of Mathematics and Statistics
University of Regina

1 Introduction

Imagine you arrive at one of the "big box" department stores that have sprung up around the country with a bag of live garden gnomes. No sooner do you enter the store then you trip, break you glasses, and drop your bag of gnomes. The gnomes scatter throughout the store. Fearing the wrath of the store manager, you begin to look for the gnomes. You are hampered by several problems. First, the shelves in the big box store are very high; too high to pass over. Second, with your broken glasses, the only way you can catch a gnome is by running into it. Third, the gnomes are incredibly quick - while no gnome could walk past you in an aisle, picking the wrong way at a junction would allow them to slip around behind you. Finally (and most embarrassingly), you forgot to count the number of gnomes in the sack, and don't know how many you're looking for!

You begin to look around, walking around one of the large shelves. You complete a full circuit, and think that at least there are no gnomes here... when you realize that a gnome could simply be walking directly in front of you, doing the same circuit! You could walk around the shelf forever! Clearly, you need help, and even then, you need to find a way to walk through the store in a manner to guarantee that you walk around all the shelves (and other nooks and crannies) AND guarantee that no gnomes have "doubled back" on you.

This model of searching was originated by Parsons in [14], though the problem was of interest to spelunkers earlier than that [6]. Parson's original problem dealt with finding a lost spelunker in a system of caves, but the problem has much wider application. We are interested in searching as a problem in network security, looking for methods to clean a network of a computer virus, or methods to capture a mobile intruder using software agents. In the literature, searching has been linked to the pathwidth and vertex separation of a graph [7, 10], to pebbling (and hence to computer memory usage) [10], to assuring privacy when using bugged channels [9], and to VLSI (very large-scale integrated) circuit design [8]. A brief survey of results is also available [1].

An alternate, but equally valid paradigm, to spelunking would be to consider a building filled with poison gas. The job of the searchers is to remove all the poison gas. Clearly, if a searcher comes to a junction, at which point the area behind him has been cleared of poison gas, but the area ahead of him has not, great care must be taken, else the cleared area will be recontaminated. Our goal becomes to remove all the poison gas.

We will deal primarily with graphs. We define a *reflexive graph* to be a graph in which loops are also allowed as edges, and a *multigraph* is a graph in which multiple edge may exist between pairs of vertices. The number of edges incident with a vertex v of a graph G is the *degree* of v, denoted $\deg(v)$.

In this search model, collision between a searcher and an intruder may occur on an edge. We will call this type of search a *edge search*. In Parson's original model, graphs were considered to be embedded in three-space and the motion of searchers (and intruders) in the graph was described by continuous

[1]Research was supported in part by NSERC and MITACS.
[2]Research was supported in part by NSERC and MITACS.

functions. A successful strategy would be a set of functions for the searchers such that at some time t, the function value for some searcher must be the same as that of the intruder.

We will consider a similar, but discrete, model. The specifics of edge-searching a reflexive multigraph G are as follows. Initially, all edges of G are *contaminated* (or *dirty*). To search G it is necessary to formulate and carry out a search strategy. A strategy is a sequence of actions designed so that the final action leaves all edges of G *uncontaminated* (or *cleared*). In such strategies, only the following three actions are allowed.

1. Place a searcher on a vertex.
2. Move a single searcher along an edge uv starting at u and ending at v.
3. Remove a searcher from a vertex.

Any strategy that uses the above actions will be called a *search strategy*. A strategy that restricts itself to the first two actions will be called an *internal search strategy*. Notice that in a disconnected graph, the number of searchers needed for an internal search strategy would be much higher than the number needed for a search strategy. In the former, there needs to be sufficiently many searchers in each component, while in the latter, searchers could "jump" from component to component. This paper will deal solely with connected graphs.

An edge uv in G can be *cleared* in one of two ways.

1. At least two searchers are placed on vertex u of edge uv, and one of them traverses the edge from u to v while the others remains at u.
2. A searcher is placed on vertex u, where edges incident with u, other than uv, are already cleared. Then the searcher moves from u to v.

Knowing that our goal is to end up with a graph where all the edges are cleared, a basic question is: what is the fewest number of searchers for which an search strategy exists? We call this the *search number*, denoted $\mathrm{s}(G)$. We define the *internal search number* similarly and denote it $\mathrm{is}(G)$. In fact, these two numbers are equal for connected graphs [2]. It can be seen that for any graph G the search number exists by considering the following strategy. Place a searcher on each vertex of G, and then use a single extra searcher to clear all the edges.

Further restrictions may be placed on all strategies. Let $E(i)$ be the set of cleared edges after action i has occurred. A search strategy for a graph G for which $E(i) \subseteq E(i+1)$ for all i is said to be *monotonic*. We may then define the *monotonic search number* and the *monotonic internal search number*, denoted $\mathrm{ms}(G)$ and $\mathrm{mis}(G)$, respectively. Similarly, a search strategy such that $E(i)$ induces a connected subgraph for all i is said to be *connected*, and we may define the *connected search number* $\mathrm{cs}(G)$ and the *connected internal search number* $\mathrm{cis}(G)$. Finally, a search strategy may be both connected and monotonic, giving us the *monotonic connected search number* $\mathrm{mcs}(G)$ and the *monotonic connected internal search number* mics.

LaPaugh [11] and Bienstock and Seymour [5] proved that for any connected graph G, $\mathrm{s}(G) = \mathrm{ms}(G)$. Barrière et al. [4] extended this result, giving the following relations for these numbers.

1.1 Lemma. *For any connected graph G,*

$$\begin{aligned} \mathrm{is}(G) = \mathrm{s}(G) = \mathrm{ms}(G) \leqq \mathrm{mis}(G) &\leqq \mathrm{cs}(G) = \mathrm{cis}(G) \\ &\leqq \mathrm{mcs}(G) = \mathrm{mics}(G). \end{aligned}$$

This chain of inequalities suggest several questions. For instance, can equality be achieved? Do graphs exist for which the inequalities are strict?

Parsons proved the following interesting result about the search number of trees [14].

1.2 Theorem. *Let T_1, T_2, and T_3 be vertex-disjoint trees each having at least one edge, where vertex v_j is a vertex of degree one in tree T_j, $1 \leqq j \leqq 3$. Let T be the tree formed by identifying the vertices v_1, v_2, v_3 as a single vertex v. If $\mathrm{s}(T_i) = k$, then $\mathrm{s}(T) = k + 1$.*

Thus, trees may have arbitrarily large search number. Barrière et al. [3, 4] also proved that there are at most two search numbers for a given tree. That is, for a tree T, we have

$$\begin{aligned} \mathrm{is}(T) = \mathrm{s}(T) = \mathrm{ms}(T) \leqq \mathrm{mis}(T) &= \mathrm{cs}(T) = \mathrm{cis}(T) \\ &= \mathrm{mcs}(T) = \mathrm{mics}(T), \end{aligned}$$

and the inequality may be strict.

We will show that $\mathrm{is}(K_n) = \mathrm{mics}(K_n) = n$, where K_n is the complete graph on n vertices. This means that there is exactly one search number for complete graphs.

In general, determining the search number of a graph G is NP-complete [12]. However, the search number of a tree can be computed in linear time. As any successful search strategy gives an upper bound, our goal becomes first to find the "right" way to clear the graph, using as few searchers as possible. Once this strategy is found, we must then prove that no fewer searchers will suffice. Here is where the true difficulty lies: most easily attainable lower bounds are quite poor. We will prove several lower bound results using the clique number of a graph.

1.3 Definition. The *clique number* of the graph G, denoted $\omega(G)$, is the number of vertices in a largest clique of G.

Finding the "right" strategy is also very hard, but for most of the graphs considered such strategies will be given explicitly. Work on upper bounds is also vital, and recently an upper bound on the monotonic connected internal search number of planar cubic graphs was found by considering flooding [13].

Returning to Lemma 1.1, we still need to consider the question of strict inequality. An example in [4] shows that the first inequality may be strict. The graph below, which we call the "Y-square", is another example. Moreover, this is an example with fewer vertices and edges. For this example, the search number is 3, while the monotonic internal search number is 4. We conjecture that this is the smallest graph that exhibits the strict inequality between search number and monotonic internal search number.

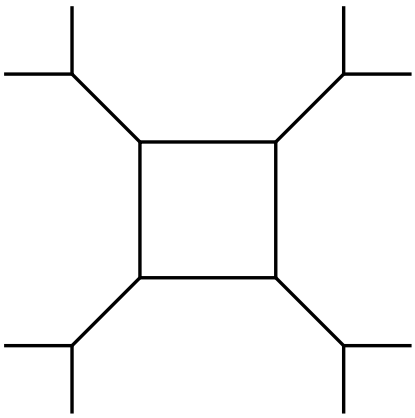

Figure 1: The Y-square.

The second inequality, $\mathrm{mis} \leqq \mathrm{cs}$, was also proved in [4]. Further, they gave an example showing that the inequality was strict. With this result, they also observed that generally, the monotonic internal search number or connected search number of a graph G may be less than the monotonic internal search number or connected search number of some minors of G. We prove these results by using large cliques as our building blocks, thereby allowing us to calculate the search numbers easily.

Whether the third inequality, $\mathrm{cs} \leqq \mathrm{mcs}$, can be strict was left as an open problem in [4]. We will show that there exists a graph G such that $\mathrm{cs}(G) < \mathrm{mcs}(G)$, and that, in fact, the difference between these two values can be arbitrarily large.

This paper is organized as follows. In Section 2, we examine the searching of cliques, and some of the immediate consequences. In Section 3, we describe the construction of a graph W such that $\mathrm{cs}(W) < \mathrm{mcs}(W)$, and include some lemmas that hint at the construction of the graph. Sections 4 and 5 are devoted to the analysis of the connected search number and monotonic connected search number of W. In Section 6, we re-examine several of the observations of [4], proving not only that inequalities can be strict, but the difference can be arbitrarily large, as well as examining variations in the number of exposed vertices in a search. The proofs of these results rely heavily on the idea of clique number. Finally, we mention several open problems in Section 7. Some of this work has appeared in preliminary form in [17].

2 Searching and Cliques

The main result of this section is Theorem 2.4. We use it to prove several lower bounds for the search number.

2.1 Definition. A vertex in a graph G is said to be *exposed* if it has edges incident with it that are contaminated as well as edges incident with it that are cleared. Following a search strategy S on G, we defined $\mathrm{ex}_S(G, i)$ to be the number of exposed vertices after the i-th step. We also define the *maximum number of exposed vertices* to be $\mathrm{mex}_S(G) = \max_i \mathrm{ex}_S(G, i)$.

2.2 Definition. A vertex v is said to be *cleared* if all the edges incident with it are currently uncontaminated.

The following theorem is a "junior" version of our eventual goal.

2.3 Lemma. [12] *At the time the first vertex becomes cleared while searching the complete graph $K_n, n \geqq 4$, there must be at least $n-1$ searchers on the vertices of K_n.*

It is easy to see that $\mathrm{s}(K_1) = 1$, $\mathrm{s}(K_2) = 1$, and $\mathrm{s}(K_3) = 2$, and only slightly more difficult to see that $\mathrm{s}(K_4) = 4$. The jump of the search numbers from 2 for K_3 to 4 for K_4 indicates that obvious methods, such as mathematical induction, will not easily prove a formula for $\mathrm{s}(K_n)$.

The following result gives several useful corollaries. For a graph G, we will denote the minimum degree of G by $\delta(G)$.

2.4 Theorem. *If G is a connected graph and $\delta(G) \geqq 3$, then $\mathrm{s}(G) \geqq \delta(G) + 1$.*

Proof. Consider a graph G with minimum degree $\delta(G)$, and a search strategy S that clears it. If the first vertex cleared by S is not of minimum degree, then it must have at least $\delta(G) + 1$ vertices adjacent to it. When it is cleared, each of the these vertices must contain a searcher and $\mathrm{s}(G) \geqq \delta(G)+1$.

We now consider the last time that the graph goes from having no cleared vertices to a single cleared vertex u. By the preceding paragraph, we may assume that u is a vertex of minimum degree. We will assume that the strategy S employs at most $\delta = \delta(G)$ searchers, and arrive at a contradiction. Let the neighbours of u be denoted $v_1, v_2, \ldots, v_\delta$. Assume, without loss of generality, that uv_1 is the final edge incident with u cleared, and that uv_2 is the penultimate such edge.

Consider the placement of searchers the moment before uv_1 is cleared. Since each of uv_i, $2 \leqq i \leqq \delta$, is cleared, there must be a searcher on each end vertex of these edges. But this uses all δ searchers. Thus the only way that uv_1 can be cleared is if the searcher at u traverses the edge uv_1 from u to v_1. Thus, all the other edges incident with v_1 must be contaminated. Since $\delta \geqq 3$, the searcher on v_1 cannot move.

Now consider the placement of searchers before the penultimate edge uv_2 is cleared. Again, as each of the edges uv_i, $3 \leqq i \leqq \delta$, is cleared, there must be a searcher on each end vertex of these edges. This accounts for $\delta - 1$ searchers. Since the next move is to clear uv_2, the single free searcher must be on either u or v_2. Sweeping from v_2 to u would instantly recontaminate the edge uv_2 which implies the

edge must be cleared from u to v_2. This leaves the searcher at v_2, and all the other edges incident with v_2 must be contaminated. Since $\delta \geqq 3$, the searcher on v_2 cannot move.

Consider a searcher on v_i, $3 \leqq i \leqq \delta$. If the vertex v_i is adjacent to v_1 and v_2, then the edges v_1v_i and v_2v_i are contaminated, and the searcher at v_i cannot move.

If the vertex v_i is adjacent to exactly one of v_1 and v_2, it must also be adjacent to some other vertex w not adjacent to u (as the degree of v_i is at least δ). As there is no searcher on w, the only way that v_iw can be cleared is if w is a cleared vertex. However, we know that u is the first cleared vertex, so that w is not cleared. Thus, the searcher at v_i cannot move.

Finally, if the vertex v_i is adjacent to neither v_1 nor v_2, it must be adjacent to two vertices w_1 and w_2 neither of which is adjacent to u. As before, these edges cannot be cleared, and thus the searcher at v_i cannot move.

As there are still contaminated edges, and none of the δ searchers can move, we have obtained the required contradiction. ■

2.5 Corollary. *For a connected graph G, let $\kappa(G)$ be the vertex connectivity and $\kappa'(G)$ be the edge connectivity of G. If $\kappa(G) \geqq 3$, then $\mathrm{s}(G) \geqq \kappa'(G) + 1 \geqq \kappa(G) + 1$.*

Proof. Since $\delta(G) \geqq \kappa'(G) \geqq \kappa(G)$ [16], the result follows from Theorem 2.4. ■

2.6 Corollary. *For $n \geqq 4$, $\mathrm{s}(K_n) = n$.*

Proof. By Theorem 2.4, we know that $\mathrm{s}(K_n) \geqq n$. We present the following search strategy for K_n using n searchers. First, clear a vertex v of K_n. This requires $n-1$ searchers, leaving one free. This free searcher may then clear all the edges of K_n that are not incident with v. ■

Corollary 2.6 actually proves a much stronger result, namely, that $\mathrm{mcs}(K_n) \leqq n$, as the search strategy described in the proof is in fact both monotonic and connected. From Lemma 1.1 and Corollary 2.6, it follows that there is only one search number for the complete graph K_n.

2.7 Corollary. *For all positive integers n,*

$$\begin{aligned} \mathrm{is}(K_n) = \mathrm{s}(K_n) &= \mathrm{ms}(K_n) = \mathrm{mis}(K_n) = \mathrm{cs}(K_n) \\ &= \mathrm{cis}(K_n) = \mathrm{mcs}(K_n) = \mathrm{mics}(K_n) = n. \end{aligned}$$

2.8 Definition. The *wheel graph* W_n , $n \geqq 3$, is the graph formed by connecting all the vertices of an n-cycle to another vertex v not on the cycle.

2.9 Corollary. *For all $n \geqq 3$,*

$$\begin{aligned} \mathrm{is}(W_n) = \mathrm{s}(W_n) &= \mathrm{ms}(W_n) = \mathrm{mis}(W_n) = \mathrm{cs}(W_n) \\ &= \mathrm{cis}(W_n) = \mathrm{mcs}(W_n) = \mathrm{mics}(W_n) = 4. \end{aligned}$$

Proof. For all $n \geqq 3$, $\delta(W_n) = 3$. Thus, by Theorem 2.4, $4 \leqq \mathrm{s}(W_4)$. It remains to show that 4 searchers are sufficient for a monotonic connected search of W_n. Label the vertex of degree n with v_0, and the remaining vertices v_1, v_2, …, v_n such that v_nv_1 and v_iv_{i+1} are edges for $1 \leqq i \leqq n-1$.

Place searchers γ_1 and γ_2 at v_0 and searchers γ_3 and γ_4 at v_1. Move γ_1 along edge v_0v_1, clearing it. Move γ_1 back to v_0. Then move γ_4 to vertex v_2, clearing v_1v_2. Move γ_1 along edge v_0v_2, clearing it. Move γ_1 back to v_0. Then move γ_4 to vertex v_3, clearing v_2v_3. Repeat until all edges of W_n are cleared. Thus, $\mathrm{mcs}(W_n) \leqq 4$, as required. ■

2.10 Theorem. *If a graph H is a minor of a graph G, then $\mathrm{s}(H) \leqq \mathrm{s}(G)$.*

PROOF. Let $\Phi : V(G) \to V(H)$ denote the function that maps the vertices of G to the corresponding vertices of H that result from vertex identifications that have taken place to form the minor H. Suppose that $\mathrm{s}(G) = k$. Whenever a searcher in G moves from a vertex u along an edge to a vertex v, the corresponding searcher does nothing in H when $\Phi(u) = \Phi(v)$. If $\Phi(u) \neq \Phi(v)$, then the corresponding searcher does nothing when $\Phi(u)$ and $\Phi(v)$ are not adjacent in Y, but traverses the edge from $\Phi(u)$ to $\Phi(v)$ when they are adjacent in Y. It is easy to see that k searchers clear all of Y if they clear X. The result follows. ■

2.11 Lemma. *If G is a connected graph, then* $\mathrm{s}(G) \leqq \min(|V(G)|, |E(G)|)$.

PROOF. We first clear G with $n = |V(G)|$ searchers. Pick a vertex $v \in V(G)$ with k neighbours. On every vertex other than v and the neighbours of v, place a searcher. Place the remaining $n - k$ searchers on v. Move a searcher from v to each neighbour u of v, clearing the edge uv. At this point, there is a searcher on every vertex of G, and the single searcher on v (which is cleared) may be used to clear all remaining edges.

We next clear G with $m = |V(G)|$ searchers. Pick a vertex $v \in V(G)$ with k neighbours. For each edge xy in G, place a searcher on whichever of x and y is closer to v. (If both x and y are equidistant, pick one of them.) Then first clear v; then clear are vertices distance 1 from v; then those of distance 2, and so on. ■

2.12 Theorem. *For $n \geqq 4$, K_{n+1} is the unique connected supergraph of K_n with the fewest number of edges such that its search number is $n + 1$.*

PROOF. First, note that K_{n+1} is a supergraph of K_n with search number $n + 1$, and K_{n+1} has n additional edges. We will show that any other supergraph G containing K_n with at most n additional edges satisfies $\mathrm{s}(G) = n$.

If G is a supergraph of K_n with only $k < n$ additional edges, consider the connected components of the graph induced by $E(G) - E(K_n)$. Place one searcher on each of the k edges of these components, and search these components as in Lemma 2.11, ending on vertices of K_n. Clear the edges between these vertices, and arbitrarily pick one of these vertices and clear it. This leaves a searcher on every other vertex of K_n, and one free searcher. This searcher may clear all remaining edges. Thus, $\mathrm{s}(G) = n$.

If G is a supergraph of K_n with n additional edges, and some vertex $v \in K_n$ is not incident with v, then as before, clear the additional edges as in Lemma 2.11, ending with searchers on vertices of K_n. There are at most $n - 1$ such exposed vertices (since v is adjacent to no additional edge) and so there is a free searcher to clear all edges between these vertices. Then, arbitrarily clear one of these vertices. This leaves a searcher on every other vertex of K_n, and the remaining searcher can clear the graph. Thus, $\mathrm{s}(G) = n$.

Finally, we consider if G is a supergraph of K_n with n additional edges, and every vertex of K_n is incident with exactly one of these edges. If G contains more than one additional vertex than those in $V(K_n)$, let u be one such vertex. To search G, first clear u. Use a free searcher to clear all the neighbours of u, then clear one of the neighbours of u. This leaves a searcher on every other vertex of K_n, and one free searcher. Use this searcher to clear K_n. Then the neighbours of every vertex in $V(G) - V(K_n)$ has a searcher on each of its neighbours, and they may clear these vertices, clearing G. Thus, $\mathrm{s}(G) = n$. ■

2.13 Theorem. *If $n \geqq 4$, then the graph of order n with the most edges and search number $n - 1$ is the complete graph K_n with one edge removed.*

PROOF. Let $K_n - \{uv\}$ denote the complete graph of order n with the edge uv deleted.We first use $n - 2$ searchers to clear the vertex v and station these searchers on the $n - 2$ neighbours of v. Then we use one free searcher to clear all the contaminated edges between the $n - 2$ neighbours of v. Finally, we use $n - 1$ searchers to clear all the remaining contaminated edges incident on u. Thus, $K_n - \{uv\}$ is $(n - 1)$-searchable. On the other hand, from Theorem 2.4, we have $\mathrm{s}(K_n - \{uv\}) \geqq n - 1$. Therefore, $\mathrm{s}(K_n - \{uv\}) = n - 1$. ■

The search number of the cartesian product of graphs has also been considered in [15], where the following result is proved.

2.14 Definition. The *cartesian product* of two graphs G and H, denoted $G\Box H$, is a graph with vertex set $V(G) \times V(H)$. Two vertices (u_1, v_1) and (u_2, v_2) are adjacent in $G\Box H$ if $u_1 = u_2$ and $v_1v_2 \in E(H)$ or if $u_1u_2 \in E(H)$ and $v_1 = v_2$.

2.15 Theorem. *For two connected graphs G and H,*

$$\mathrm{s}(G\Box H) \leqq \min\left(|V(G)| \cdot \mathrm{s}(H), |V(H)| \cdot \mathrm{s}(G)\right) + 1.$$

2.16 Corollary. *If G is a connected graph and $n \geqq 4$, then*

$$\mathrm{s}(K_n\Box G) \leqq n \cdot \mathrm{s}(G) + 1.$$

In fact, it is easy to see that we can do better than this when G is also a complete graph.

2.17 Corollary. *For $n \geqq 1$ and $m \geqq 2$, $\mathrm{mcs}(K_n\Box K_m) \leqq n(m-1)+1$.*

Proof. Label m vertex–disjoint n-cliques of $K_n\Box K_m$ by A_1, A_2, ..., A_m, with $V(A_i) = \{v_{i,j} | 1 \leqq j \leqq m\}$. Place one searcher on each of the $v_{1,j}$, and the remaining searchers anywhere on A_1. Use a free searcher to clear all the edges in A_1. There is a perfect matching between A_1 and A_2. Move n searchers to clear the perfect matching, $v_{1,j}v_{2,j}$ ending in A_2. Similarly, searchers can traverse perfect matchings from A_1 to A_i, $3 \leqq i \leqq m$. This leaves $n(m-1)$ searchers stationed on A_i, $2 \leqq i \leqq m$, and the remaining free searcher can clear all the edges between these vertices. ■

In particular, $\mathrm{mcs}(K_n\Box K_n) \leqq n(n-1)+1$. In the special case that exactly one of the complete graphs is K_2, we can say something even more precise. The graph $K_1\Box K_2$ is exactly P_2, and hence has monotonic connected search number 1. The graph $K_2\Box K_2$ is C_4, and hence has monotonic connected search number 2. For n greater than two, we have the following corollary.

2.18 Corollary. *If $n \geqq 3$, then $\mathrm{s}(K_n\Box K_2) = n+1$.*

Proof. By Theorem 2.15, we know that $\mathrm{s}(K_n\Box K_2) \leqq n+1$. But as well, $\delta(K_n\Box K_2) = n$, so by Theorem 2.4, $\mathrm{s}(K_n\Box K_2) \geqq n+1$. ■

3 Construction of the obstruction W

From Corollary 2.7 and Theorem 2.10, we obtain the following result.

3.1 Theorem. *For any graph G, if $\omega(G) \geqq 4$, then $\omega(G) \leqq \mathrm{s}(G)$.*

Since trees have clique number 2 and there exist trees with arbitrarily large search number, it might appear that the bound presented in Theorem 3.1 is not particularly useful. Nothing could be further from the truth, as Theorem 3.1 provides a basis for constructing graphs with easily calculated search number. The general idea is to start with a graph G that can be cleared with p searchers, and use G to construct a cousin G' of G that has a K_p as a subgraph, thereby forcing the search number of G' to be at least p.

We construct the graph W as shown in Figure 2. In this figure, a circle represents a complete graph on the indicated number of vertices, and double lines between two cliques A and B indicate either a perfect matching either between A and B (if $|A| = |B|$) or between A and a subgraph of B (if $|A| < |B|$). The latter is called a *saturated matching*.

If there is a saturated matching from a graph A to a subgraph of B, we use $B[A]$ to denote the graph induced by those vertices of B adjacent to vertices of A. So $B[A]$ also is a clique.

We construct W such that $A_9[C_{19}]$, $A_9[D_{19}]$, $A_9[E_{300}]$ and $A_9[F_{300}]$ are all vertex-disjoint, and similarly for A'_9. Also, $V(A_2[C_1]) \cap V(A_2[B_1]) = \emptyset$ and $V(A_4[D_1]) \cap V(A_4[B_{300}]) = \emptyset$, and similarly for A'_2 and A'_4. Finally, there are 300 cliques between A_1 and A'_1, each of which contains 280 vertices.

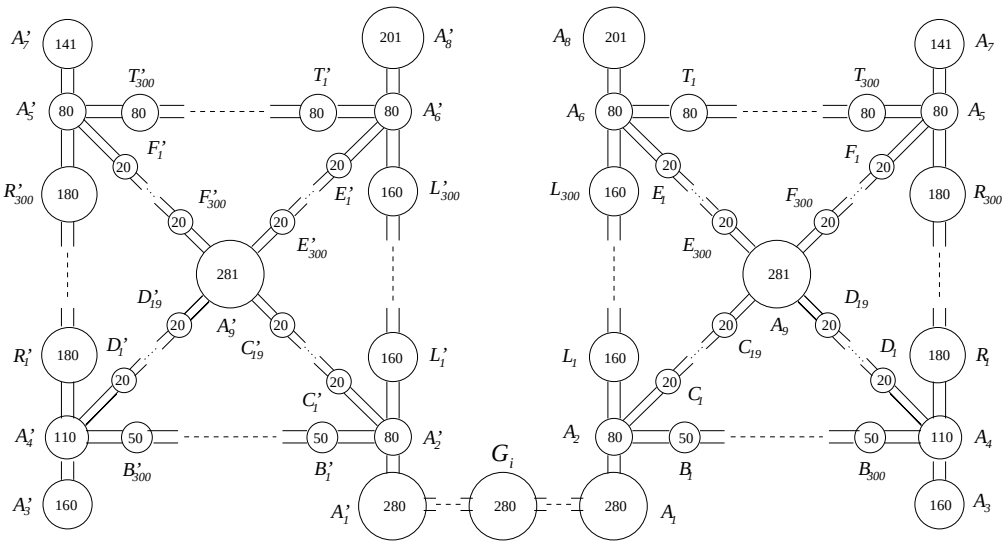

Figure 2: The graph W.

4 Computing the connected search number of W

4.1 Lemma. *In the process of a connected search of W, let v be the first cleared vertex that remains cleared for the remainder of the search. If $v \notin V(A_9) \cup V(A'_9)$, then there are more than* 281 *searchers.*

Proof. Let S be a connected search strategy such that $v \notin V(A_9) \cup V(A'_9)$. Certainly, one of A_9 and A'_9 must first obtain a cleared vertex that remains clear for the remainder of the search. Without loss of generality, assume it is A_9. Immediately after the action that clears the first vertex of A_9 there must be 280 searchers on A_9. However, as A'_9 is not cleared, there must be an exposed vertex between A'_9 and v. Thus, none of the 281 searchers used can move, necessitating at least one more searcher. ■

4.2 Theorem. *We have* $\mathrm{cs}(W) = 281$.

Proof. It follows from Corollary 2.7 that $\mathrm{cs}(A_9) = \mathrm{cs}(A'_9) = 281$ and from Theorem 3.1 that we need at least 281 searchers to clear W. To prove that this number is sufficient, we use the following search strategy.

By Lemma 4.1, we must begin in one of A_9 or A'_9. Without loss of generality, we begin by clearing a vertex in A'_9. First, place all 281 searchers on a single vertex v of $A'_9 \setminus (A'_9[C'_{19}] \cup A'_9[D'_{19}] \cup A'_9[E'_{300}] \cup A'_9[F'_{300}])$. Move 280 of them to the 280 neighbours of v. This clears v, and the single searcher remaining on v then clears all remaining edges in A'_9.

The searchers on $A'_9[C'_{19}]$ move to C'_{19} along the perfect matching, and a single free searcher clears all the edges of C'_{19}. We repeat this process until finally we place 20 searchers on $A'_2[C'_1]$, and use a single searcher to clear all edges in $A'_2[C'_1]$. Similarly, we clear the D'_i, E'_i, and F'_i subgraphs, ending with searchers on $A'_4[D'_1]$, $A'_5[F'_1]$, and $A'_6[E'_1]$. Again we use a single searcher to clear edges in these subgraphs. We now have 201 free searchers.

We send these searchers to a single vertex in A'_8 that is adjacent to $A'_6[E'_1]$. Then we clear this vertex leaving a single free searcher who clears all remaining edges in A'_8. Then the searchers in $A'_8[A'_6]$ may lift to A'_6, and a single free searcher may clear all edges in A'_6.

At this point, we have 20 searchers stationed on each of $A'_2[C'_1]$, $A'_4[D'_1]$, and $A'_5[F'_1]$, as well as 80 stationed on A'_6. This leaves 141 free searchers. We move all these searchers to a vertex of A'_7 that is adjacent to $A'_5[F'_1]$. We then clear this vertex, and with the remaining free searcher, we clear the edges of A'_7. Lifting the searchers on $A'_7[A'_5]$ along the perfect matching to A'_5, we clear A'_5. We now have 80 searchers stationed on A'_5, 80 on A'_6, and 20 each on $A'_2[C'_1]$ and $A'_4[D'_1]$ leaving 81 free searchers.

We clear along the perfect matchings in the T'_i, using the single remaining searcher to clear each of the edges in the T'_i, and finally clear the edges from T'_1 to A'_6. Now we move the 80 searchers stationed at A'_6 to $L'_{300}[A'_6]$. We use a single searcher to clear all the edges in $L'_{300}[A'_6]$, and then move 80 more searchers to the remaining vertices of L'_{300}. We use the single free searcher to clear all remaining edges in L'_{300}. Then clear to L'_{299}, and use the single free searcher to clear the edges in that subgraph, and continue until the edges in L'_1 are cleared. Then send the searchers from $L'_1[A'_2]$ to A'_2, and use a free searcher to clear all the edges in A'_2. There are now 80 searchers stationed at each of A'_2 and A'_5, and 20 stationed on $A'_4[D'_1]$. This leaves 101 free searchers.

We also clear the R'_i as we cleared the L'_i, first searching from A'_5 to $R'_{300}[A'_5]$, then using the 101 free searchers with these 80 searchers to clear all the R'_i. Then we move the searchers from $R'_1[A'_4]$ to A'_4, and use a single searcher to clear the remaining edges of A'_4. We now have 190 searchers stationed at A'_4 and A'_2, leaving 91 free searchers. We use these searchers to clear the B'_i subgraphs, and then we use 160 searchers to clear A'_3. This clears the left half of the graph W. We now have only 80 searchers stationed at A'_2.

We now use 281 searchers to clear, one by one, the 300 cliques between A'_1 and A_1, followed by A_1 itself. Then we move the searchers from $A_1[A_2]$ to A_2, and use a free searcher to clear all edges in A_2. We now have 80 searchers stationed in A_2, leaving 201 free searchers.

Pick a vertex in C_1 and clear to this vertex from $A_2[C_1]$. Then move another searcher along this edge, and to the corresponding vertex in C_2. Then another to the corresponding vertex in C_3, and so on, until finally we have placed a searcher on the corresponding vertex in $A_9[C_{19}]$. Then move a searcher to a vertex in $A_9[D_{19}]$, followed by moving a searcher to a corresponding vertex in D_{19}, then another to a corresponding vertex in D_{18}, and so on, until reaching the corresponding vertex in D_1. Finally, move one searcher to the corresponding vertex in $A_4[D_1]$. We now have 80 searchers stationed in A_2, and in total, 41 searchers along a path through the C_i, through A_9, and finally through the D_i into A_4. This leaves 160 free searchers.

Move these free searchers along this path, to the single vertex in A_3 adjacent to the path. Clear this vertex, and then use the single free searcher to clear A_3. Then the searchers on $A_3[A_4]$ may clear to A_4, and a free searcher may clear A_4. With 110 searchers stationed on A_4, 80 searchers stationed on A_2, and 40 searchers strung in that path from A_1 to A_4 through A_9, there are 51 free searchers. These searchers can clear the B_i.

We now collapse the path from A_2 to A_4 through A_9, in the following manner. The single searcher in D_1 moves to the corresponding vertex in D_2. The two searchers now in D_2 move to the corresponding vertex in D_3. Continue in this way until finally all the searchers in the path are in C_1, at which point they return to $A_2[C_1]$. By "reeling in" the path in this manner, we preserve the connectedness of this search, but the search is not monotonic.

We now have 190 searchers stationed on A_2 and A_4, leaving 91 searchers free. Of these free searchers, move 20 from A_4 to D_1, using a free searcher to clear D_1. Then clear D_2, then D_3, until finally we station 20 searchers on $A_9[D_{19}]$ and use a free searcher to clear the edges of $A_9[D_{19}]$. Now move the 110 searchers on A_4 to $R_1[A_4]$. Use a free searcher to clear the edges of $R_1[A_4]$. Place the remaining 71 free searchers on a vertex on $R_1[A_4]$ and clear it. With the single remaining free searcher, clear R_1. Then move all searchers in R_1 to R_2, and use the remaining free searcher to clear R_2. Repeating, clear to R_{300}, finally moving 80 searchers from $R_{300}[A_5]$ to A_5 and using a single free searcher to clear A_5. We now have 160 searchers stationed at A_2 and A_5, and 20 searchers stationed at $A_9[D_{19}]$. This leaves 101 free searchers.

As with the D_i, use 21 searchers to clear the C_i, eventually stationing 20 searchers at $A_9[C_{19}]$, and using a free searcher to clear the edges of $A_9[C_{19}]$. Then clear the 80 searchers from A_2 to $L_1[A_2]$, and use the 81 free searchers to first clear a vertex in $L_1[A_2]$ and then to clear L_1. We then clear all the L_i, eventually stationing 80 searchers at A_6 and using a free searcher to clear the edges of A_6. We now have 80 searchers at each of A_5 and A_6, and 20 searchers at each of $A_9[C_{19}]$ and $A_9[D_{19}]$, leaving 81 free searchers.

We use these searchers to clear the T_i. Then we clear the F_i, eventually stationing 20 searchers on $A_9[F_{300}]$, and using a free searcher to clear the edges of $A_9[F_{300}]$. We then move the 80 searchers stationed at A_5 to $A_7[A_5]$, and use a free searcher to clear $A_7[A_5]$. With 80 searchers at $A_7[A_5]$, 80

at A_6, and 60 in A_9, there are 61 free searchers. Clear a vertex of $A_7[A_5]$, and then use the single remaining free searcher to clear A_7.

As before, clear all the E_i by searching from A_6, eventually stationing 20 searchers at $A_9[E_{300}]$, and using a free searcher to clear the edges of $A_9[E_{300}]$. Then move the 80 searchers stationed at A_6 to $A_8[A_6]$, using a free searcher to clear the edges of $A_8[A_6]$. We now have 80 searchers stationed in A_9, and 80 searchers stationed in $A_8[A_6]$. The remaining 121 searchers may be used to clear a vertex in $A_8[A_6]$, and then the single remaining free searcher may be used to clear the edges of A_8.

Finally, with only 80 searchers stationed in A_9 (and thus 201 free searchers), we clear a vertex in A_9, and then use the single remaining free searcher to clear the edges of A_9, which completes the search strategy for W. This strategy, as has been noted, is connected, but is not monotonic, as the edges in the path from A_2 through A_9 to A_4 were allowed to be recontaminated. ■

5 Computing the monotonic connected search number of W

5.1 Theorem. *We have* $\mathrm{mcs}(W) = 290$.

Proof. We first show that $\mathrm{mcs}(W) \leqq 290$. Starting at A'_9, we may search as in Theorem 4.2 until we clear all the edges in A_2. At this point, we have 80 searchers stationed at A_2, and 210 free searchers. Moving 50 of these searchers to B_1, we then use another free searcher to clear the edges in B_1. Then repeat with B_2, B_3, ..., B_{300}. Finally, station 50 searchers at $A_4[B_{300}]$, and use another searcher to clear all the edges in $A_4[B_{300}]$. We now have stationed 130 searchers and have 160 free.

We send these 160 searchers to a single vertex in A_3 that is adjacent to $A_4[B_{300}]$, and then clear this vertex. Using the single remaining free searcher, we clear the edges of A_3. Then we lift the searchers from $A_3[A_4]$ to A_4, and clear all remaining edges in A_4. We now have 110 searchers stationed at A_4, and 80 stationed at A_2, leaving 100 free searchers. We use these searchers to clear the D_i, eventually placing 20 searchers on $A_9[D_{19}]$. Using another searcher, we clear the edges of $A_9[D_{19}]$. Then we move all the searchers on A_4 to $R_1[A_4]$. We have now stationed 80 searchers on A_2, 20 searchers on $A_9[D_{19}]$, and 110 on $R_1[A_4]$. This leaves 80 free searchers.

With these searchers (and the 110 already in R_1), clear the R_i, and eventually station 80 at A_5, and use another searcher to clear the edges of A_5. We now have 80 searchers stationed at each of A_2 and A_5, and 20 stationed at $A_9[D_{19}]$. This leaves 110 free searchers.

Using these searchers, clear the C_i, eventually stationing 20 at $A_9[C_{19}]$ and use a free searcher to clear the edges of $A_9[C_{19}]$. We now have 90 free searchers. Using these searchers, and those from A_2, we clear the L_i, eventually stationing 80 searchers on A_6. With 80 searchers stationed on each of A_6 and A_5, and 20 stationed on each of $A_9[C_{19}]$ and $A_9[D_{19}]$, we have 90 free searchers. Use these searchers to clear the T_i. Then use these searchers to clear the F_i, eventually stationing 20 searchers on $A_9[F_{300}]$, using a single free searcher to clear the edges of $A_9[F_{300}]$.

With 60 searchers stationed in A_9, and 80 stationed at A_6, we have 150 (including 80 at A_5) with which we can clear A_7. Then these searchers can clear the E_i, finally stationing 20 searchers at $A_9[E_{300}]$, and clearing the edges of $A_9[E_{300}]$. With 80 searchers in A_9, we have 210 searchers (including 80 at A_6), with which we can clear A_8. Finally, these 210 searchers move to A_9 and clear it. Thus $\mathrm{mcs}(W) \leqq 290$.

To prove the equality, we will show that $\mathrm{mcs}(W) > 289$. First, assume that W is 289-monotonically connected searchable. Let S be a monotonic connected search strategy using 289 searchers. Let Y be the subgraph induced by W on the "right side" (starting with A_2 and then to the right and above), and let Z be the subgraph obtained by adding the connecting 280-cliques from A'_1 through A_1.

Because of the involution automorphism interchanging the right side and left side, we may assume the first cleared edge lies to the left of the 280-clique G_{150} or has one end vertex in G_{150}.

We will make heavy use of vertex-disjoint paths determined by perfect matchings between successive cliques. The most important family of such paths is P_1,P_2,...,P_{80} consisting of the 80 vertex-disjoint paths having one end vertex in A_2 and the other end vertex in G_{150} along the chain of connecting 280-cliques.

We call a clique pseudo-cleared if it contains exactly one cleared vertex. We are interested in which of A_3, A_8 or A_9 is the first to be pseudo-cleared. Suppose A_9 is the first of the three to be pseudo-cleared.

At the moment the first vertex of A_9 is cleared, there must be 280 exposed vertices in A_9. There must be a path Q from G_{150} to A_9 in the subgraph of cleared edges. The path Q must pass through at least 19 20-cliques. Since there are most 9 additional exposed vertices, at least one of the 20-cliques, call it K, through which Q passes is cleared. From K, there are 20 vertex-disjoint paths back to A_2 not passing through A_9. Without loss of generality, assume these 20 vertex-disjoint paths terminate at the end vertices of P_1,P_2,...,P_{20}. Call the extensions of P_1,P_2,...,P_{20} to K by Q_1,Q_2,...,Q_{20}.

For each Q_i, $1 \leqq i \leqq 20$, we examine what happens as we start working back from K along the path Q_i. Since the clique K is cleared, the last vertex of Q_i is cleared. That is, the last edge of Q_i is cleared. Move to the preceding vertex. If it is not cleared, then we have encountered an exposed vertex on Q_i. If it is cleared, then we move to the preceding vertex on Q_i.

If Q_i passes through either A_6 or A_4, either we encounter an exposed vertex or the vertex u_i of Q_i in A_6 or $A_4$is cleared. But if the latter is the case, then the edge from u_i to A_8 or A_3 is cleared. Since neither A_8 nor A_3 have any cleared vertices, we have found an exposed vertex corresponding to the path Q_i.

If Q_i does not pass through A_4 or A_6, then either we encounter an exposed vertex or we reach the vertex u_i of Q_i in A_2, with u_i cleared. But now we extend a path from u_i through the L_j-cliques to A_8 and we must eventually encounter an exposed vertex.

Therefore, each path Q_i yields a distinct exposed vertex giving us at least 300 exposed vertices. We now see that A_9 cannot be the first pseudo-cleared clique amongst A_3, A_8, and A_9.

We next consider whether A_8 can be the first pseudo-cleared clique amongst the three cliques. Assume this is the case. At the moment the first vertex of A_8 is cleared, there are 200 exposed vertices in A_8. We again know that there is a path Q from G_{150} to A_8. Since Q passes through 150 280-cliques before reaching A_2, one of the 280-cliques must be cleared. This implies that each of the paths P_1,P_2,...,P_{80} contains a cleared vertex before reaching A_2.

First suppose that Q passes through the L_j-cliques. Let Q_1,Q_2,..., Q_{80} be the extensions of P_1,P_2,...,P_{80} through the L_j-cliques with end vertices in A_6. Since there are 300 of the 160-cliques, at least one of the L_j is cleared. This implies that each Q_i has a cleared vertex strictly between A_2 and A_6. Considering a fixed i, as we work from the cleared vertex in Q_i between A_2 and A_6 towards A_6, either we encounter an exposed vertex or the vertex v_i of Q_i in A_6 is cleared. But now we extend from v_i through the T_j-cliques, through A_5, down the R_j-cliques, through A_4 into A_3. We must encounter an exposed vertex at some point on this extension. Thus, each Q_i produces one exposed vertex working towards A_6, or past it as the case may be.

Now work from the cleared vertex of Q_i between A_2 and A_6 towards A_2. Either we encounter an exposed vertex or we reach the vertex u_i of Q_i in A_2. When u_i is one of the 20 vertices adjacent to a vertex of C_1, we extend towards A_9. We must encounter an exposed vertex at some point. Thus, we obtain another 20 exposed vertices giving us 300 altogether. We conclude that the path Q does not use the L_j-cliques.

Let Q_1,Q_2,...,Q_{80} be the same extensions of P_1,P_2,...,P_{80} as used above. Since each Q_i has a cleared vertex before reaching A_2 and there is no path from A_2 to A_6 through the L_j-cliques in the cleared subgraph, every Q_i has at least one exposed vertex on it that is not in A_8. This gives us 280 exposed vertices already. The path Q must pass through either C_1,...,C_{19} or B_1,...,B_{300}. If it goes through C_1,...,C_{19}, then one of the 20-cliques must be cleared or we have too many exposed vertices. But if one of the 20-cliques is cleared, then upon building 20 vertex-disjoint paths from the 20-clique to A_9, we get at least another 20 exposed vertices.

If Q uses B_1,...,B_{300}, then we get a cleared 50-clique leading to another 50 exposed vertices in A_3 using the same kind of path extensions via vertex-disjoint paths. This establishes that A_3 must be the first pseudo-cleared clique amongst A_3, A_8, and A_9.

Assume this is the case. At the moment the first vertex of A_3 is cleared, there are 159 exposed vertices in A_3. We again know that there is a cleared path Q from G_{150} to A_3. Since Q passes through 150 280-cliques before reaching A_2, one of the 280-cliques must be cleared. This implies that each of the paths P_1, P_2, ..., P_{80} contains a cleared vertex before reach A_2.

First suppose that Q passes through the R_i-cliques. Since there are 300 R_i-cliques, not every clique can contain an exposed vertex, so one of the cliques must contain none, and hence must be cleared. Call

this clique K. Then certainly, there are 20 vertex-disjoint paths that go from K to A_4 and then through the D_i cliques to A_9. Since K is completely cleared, and A_9 contains no cleared vertices, somewhere on each of these paths there must be at least one exposed vertex. Similarly, consider 20 vertex-disjoint paths that start in K, go through the R_i, through A_5, and then through the F_i to A_9. By the same argument, each of these paths must contain an exposed vertex.

If the path Q goes through the T_i, then one of the T_i must be completely cleared. Call this clique K'. From K' there are 80 vertex-disjoint paths through the T_i into A_6, and then to A_8. Again, each of these paths must contain an exposed vertex. Finally, extending the paths P_i, $1 \leqq i \leqq 20$, from A_2, through the C_i, into A_9. Each of these 20 paths must also contain an exposed vertex. But this means that the search uses $159 + 20 + 20 + 80 + 20 = 299$ exposed vertices, and hence as many searchers, a contradiction. Thus, there is no cleared path through the T_i after going through the R_i.

Instead, we consider if the path Q goes through the R_i and not through the T_i. Since the path does not go through the T_i, we consider the 80 vertex-disjoint paths that go from K, through A_5, then through the T_i to T_1. Each of these paths must contain an exposed vertex. Extending the paths P_i, $1 \leqq i \leqq 80$, from A_2, through the L_i into A_6, then to A_9, we must similarly encounter an exposed vertex on each. This means there are at least $159+80+80 = 319$ exposed vertices, and hence too many searchers. So, there is no cleared path through the R_i.

Now suppose that Q passes through the T_i cliques. Since there are 300 T_i cliques at least one must be completely cleared. Call this clique K. Consider the 80 vertex-disjoint paths going from K, through the T_i into A_5, then through the R_i to R_{300}. Each of these paths must contain at least one exposed vertex. Similarly, the 80 paths from K through the T_i to A_6 into A_9 must each contain at least one exposed vertex. This means the search must use at least $159 + 80 + 80 = 319$ searchers. So there is no cleared path through the R_i.

Suppose instead that Q passes through the L_i. Since there are 300 L_i-cliques at least one must be cleared. Call it K. From K, through the L_i to A_6 into A_9, there are 80 vertex-disjoint paths each of which must contain an exposed vertex. Since the path Q does not pass through the T_i, it goes through the E_i. Since there are 300 of these cliques, one of the E_i must be cleared. Call it K'. There are 20 vertex-disjoint paths from K' to A_9, each of which must contain an exposed vertex. Similarly the paths P_1 to P_{20} may be extended from A_2 through the C_i to A_9, and each path must contain an exposed vertex. This accounts for $159 + 80 + 20 + 20 = 279$ exposed vertices, and hence as many searchers. If Q goes through the B_i, then some clique K'' must be cleared, since there are at most 10 searchers in all the B_i. Consider 50 vertex-disjoint paths that go from K'' through the B_i into A_4, through the R_i to R_{300}. Since there are no connected paths through the R_i, each of the 50 paths must contain an exposed vertex, meaning the search must use at least 329 searchers. Thus Q must go through the D_i. But since there are at most 10 searchers on this portion of the path, one of the cliques must be cleared. Call this clique K'''. From K''' through the D_i to A_9, there are 20 vertex-disjoint paths, each of which must contain an exposed vertex, and thus this search must use 299 searchers. Thus, Q does not pass through the L_i.

Suppose that Q passes through the C_i-cliques to A_9, then through the D_i into A_4, and finally to A_{160}. If one of the C_i cliques does contain an exposed vertex, than each vertex in that clique must be cleared, and between that clique and $A_9[C_{19}]$ there are 20 vertex-disjoint paths, each of which must contain an exposed vertex and a searcher. Either way, along the C_i cliques and $A_9[C_{19}]$ there must be 20 searchers. Similarly, there must be 20 searchers on the D_i cliques and $A_9[D_{19}]$. Again, the paths P_1 to P_{80} may be extended into vertex-disjoint paths from A_2 through the L_i-cliques to L_{300}. These paths must contain at least 80 exposed vertices. At the moment that A_3 is pseudo-cleared, every vertex in A_3 is exposed or cleared. Thus, there are 110 vertex-disjoint paths from A_3 to A_4, through the R_i to R_{300}. This accounts for $20 + 20 + 80 + 110 = 230$ searchers.

We consider the next time certain vertices are cleared. Certainly, there is a clique L among the L_i that contains no cleared vertices. Similarly, there is a clique R among the R_i that contain no cleared vertices. We consider which of L, R, or A_9 first contains a cleared vertex. If A_9 is the first that is pseudo-cleared, then there are still 110 vertex-disjoint paths from R to A_3 through the R_i and A_4 that must each contain an exposed vertex. Similarly, the extensions of P_1 through P_{80} from A_2 through the L_i to L must each contain an exposed vertex. But when A_9 first obtains a cleared vertex, there must

be at least 280 searchers in A_9, meaning such a search would require $280 + 80 + 110 = 470$ searchers. If L is pseudo-cleared before A_9 or R, then we still have the 110 exposed vertices as in the previous case. There are also still at least 40 exposed vertices amongst the C_i-cliques, D_i-cliques, $A_9[C_i]$, and $A_9[D_i]$. But when L is pseudo-cleared, it must contain at least 159 searchers. But this search requires $40 + 110 + 159 = 309$ searchers. Similarly, if R is the first of the three pseudo-cleared, the search must require $40 + 80 + 179 = 299$ searchers.

Thus, the path Q must pass through the B_i. Since there are 300 B_i cliques, one must be completely cleared. Call it K. There are 50 vertex-disjoint paths that go from K, through the B_i cliques to A_4, then through the R_i cliques to R_{300}. Since there are no cleared paths through the R_i, each of these paths must contain an exposed vertex. Similarly, vertex-disjoint extensions of P_1, P_2, $\cdots$, P_{80} from A_2 through the L_i to L_{300} must each contain at least one exposed vertex. With at least 159 searchers on A_3 when it is pseudo-cleared, this accounts for $50 + 80 + 159 = 289$ searchers.

If any of the paths contain two or more exposed vertices, the search requires more than 289 searchers. So, each path must contain exactly one exposed vertex. When A_3 is pseudo-cleared, if any cleared vertex in W contains a searcher, then the search uses more than 289 searchers. If there is more than one searcher on an exposed vertex, then the search uses more than 289 searchers. Thus, there is exactly one searcher on every exposed vertex, and no other searchers. Further, there are no other exposed vertices other than the 289 already considered.

Let v be a vertex in A_4 that is not on one of the 50 vertex-disjoint paths. If v is cleared, then there is another path, vertex-disjoint from the other 50, the goes from v through the R_i to R_{300}. This path must contain an exposed vertex, a contradiction. Thus, every vertex in A_4 that is not on one of the 50 vertex-disjoint paths is incident with no cleared edges (since v cannot be exposed). Thus every vertex in A_4 that is one of the vertex-disjoint paths must be either contaminated or exposed. Certainly, since the path Q passes through A_4, at least one vertex w is exposed.

These paths also pass through B_{300}. If there is an exposed vertex u in B_{299}, consider the path that w sits on. Certainly, the w and u cannot be on the same path, as they are both exposed. Both w and u have counterparts w' and u' in B_{300} on the same vertex-disjoint path. Since w is exposed, w' must be cleared, and hence $u'w'$ is cleared. Since u is exposed, u' must be incident with no cleared edges, a contradiction. Thus, all exposed vertices on these 50 vertex-disjoint paths must be in $A_4[B_{300}]$ or in B_{300}.

Every exposed vertex in A_4 must be incident with at least 60 contaminated edges, so no searcher on these vertices can move without allowing recontamination. For every exposed vertex v in B_{300}, the edge from that vertex to $A_4[B_{300}]$ is contaminated. If any other edge incident with v is contaminated, the searcher on v cannot move. If no other vertices are contaminated, the searcher has only one possible move, to the corresponding vertex in $A_4[B_{300}]$, where it again sits on an exposed vertex and is incident with 109 exposed edges, and cannot move without allowing the graph to be recontaminated.

Since we know that there are no exposed vertices in C_1, there are also no cleared edges. Thus, every vertex in $A_2[C_1]$ is either exposed or is incident with no cleared edges. Since Q passes through A_2, we know that A_2 must contain an exposed vertex, v. Assume that there is an exposed vertex w in G_1. Both w and v have counterparts w' and v' in A_1. Since v is exposed, v' must be clear, and hence $v'w'$ must be a cleared edge. But since w is an exposed vertex w' must have no cleared edge incident with it, a contradiction. Thus, there can be no exposed vertices in G_1. By a similar argument, there can be no exposed vertices in L_2. So, of the 80 exposed vertices on the 80 vertex-disjoint paths from that have been extend from A_2 to L_{300}, all the exposed vertices must be on A_1, A_2, or L_1.

Since B_1 can contain no exposed vertices, all vertices must be clear. (If not, there would be an exposed vertex between B_1 and K.) Then, the vertices of $A_2[B_1]$ must be either clear or exposed. If they are clear, then the vertices in $A_2[C_1]$ must be exposed. The 50 vertices of $L_1[A_2[B_1]]$ must be exposed. Let $A = V(A_2) - (V(A_2[B_1]) \cup V(A_2[C_1])$. Then there are in total 10 exposed vertices in A and $B_1[A]$. Every vertex in $A_2[C_1]$ is incident with two or more contaminated edges (from A_2 to C_1 and from A_2 to L_1), and so the searchers on these vertices cannot move. The 50 searchers on the exposed vertices of $L_1[A_2[B_1]]$ likewise cannot move, as each of these vertices is incident with at least 100 contaminated edges. Finally, if v is a vertex of A, and v is exposed, and incident with two or more contaminated edges, then the searcher on v cannot move. If it is incident with only one contaminated

edge, it must be to B_1, and when the searcher clears that edge it is again on an exposed vertex, this one incident with at least 100 contaminated edges, and hence cannot move. By the same argument, if a vertex in $B_1[A]$ is exposed, it must be incident with at least 100 contaminated edges, and hence its searcher cannot move.

If the vertices of $A_2[B_1]$ are exposed, and the vertices of $A_2[C_1]$ have no cleared edges incident with them, then the vertices of A are either exposed, or contain no cleared edges. Between A and $A_1[A]$ there must be exactly 10 exposed vertices. Any vertex in A_2 that is exposed is incident with at least 20 contaminated edges, so the searcher on that vertex cannot move. Any exposed vertex in A_1 if not incident with two contaminated edges, must have a contaminated edge to A_2. Upon clearing this, the searcher is incident with at least 19 contaminated edges and cannot move.

Finally, if both the vertices of $A_2[B_1]$ and of $A_2[C_1]$ are exposed, then the other 10 exposed vertices are between A, $L_1[A]$ and $A_1[A]$. By similar arguments as before, every searcher on an exposed vertex either cannot move, or can clear at most one more edge without being able to clear any further.

Since this search strategy S is monotonic and connected, the first time there is an exposed vertex in A_4, the 80 searchers around A_2 and the 50 searchers on B_{300} or A_4 must already be in place. This leaves exactly 159 searchers to clear a vertex in A_3. As in Theorem 2.4, once this vertex is cleared, there are at least two contaminated edges incident with each vertex, and hence no searcher in A_3 can move. Since every other searcher has at most one more move before being unable to move, there is no way to finishing searching W, a contradiction. Thus, $\mathrm{mcs}(W) > 289$. ■

6 Applications of the clique method

6.1 Inequalities

Theorems 4.2 and 5.1 can be summarized with the following result.

6.1 Corollary. *For the graph W,* $\mathrm{cs}(W) = 281 < 290 = \mathrm{mcs}(W)$.

We have exhibited the power of cliques in constructing the graph W. We will now extend this technique to other graphs so that we may study other properties of searching.

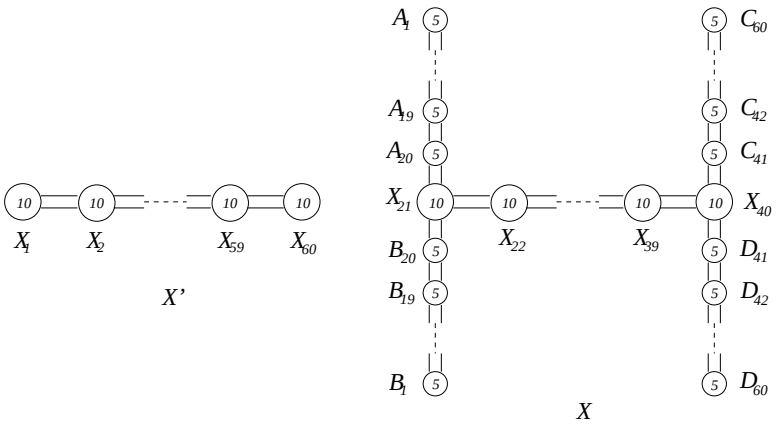

Figure 3: The graph X' and its subgraph X.

Referring to Figure 3, let $X' = K_{10} \square P_{60}$. We construct X as pictured, where each circle represents a complete graph on the number of vertices in the circle, and double lines between two cliques indicate a saturated matching.

Also, by construction, $V(X_{21}[A_{20}]) \cap V(X_{21}[B_{20}]) = \emptyset = V(X_{40}[C_{41}]) \cap V(X_{40}[D_{41}])$. It is easy to see that X is a subgraph of X'. However, we can prove $\mathrm{cs}(X) > \mathrm{cs}(X')$.

6.2 Theorem. *For X as pictured in Figure 3,*

$$\mathrm{s}(X) < \mathrm{mis}(X) < \mathrm{cs}(X).$$

PROOF. Recall from Corollary 2.18 that $\mathrm{s}(K_{10}\Box K_2) = 11$. Since X contains $K_{10}\Box K_2$ as a minor, by Theorem 2.10 we know that $\mathrm{s}(X) \geqq 11$. In fact, we can use 11 searchers to connected clear X, by first placing 5 searchers on A_1 and 5 searchers on B_1. Then a single free searcher can be used to clear all the edges in A_1. Then 5 searchers move along the perfect matching to A_2, and a single free searcher clears all the edges in A_2, and so on, finally reaching $X_{21}[A_{20}]$. The single free searcher moves to B_1, and the process is repeated, clearing the B_i and moving to $X_{21}[B_{20}]$. With 10 searchers on X_{21}, a single free searcher may then clear all the edges of X_{21}. These 11 searchers may then clear the X_i clique by clique, finally reaching X_{40}. Stationing 5 searchers on $X_{40}[D_{41}]$, the remaining 6 searchers may clear the C_i. Then the D_i may be cleared. Thus, $\mathrm{s}(X) = 11$.

The graph X can be cleared by 16 searchers in a connected search. Placing all 16 searchers on A_1, we may use one free searcher to clear the edges of A_1. Then 5 searchers may move to A_2, and a free searcher may clear the edges of A_2, and so on, until finally $X_{21}[A_{20}]$ is cleared. Then we may place 5 more searchers on the remaining vertices of X_{21}, and use a free searcher to clear the remaining edges in X_{21}. Leaving a searcher on each vertex of X_21, there are 6 free searchers. These searchers may clear the B_i clique by clique. Then the 10 searchers on X_{21} plus another free searcher may clear the X_i through X_{40}. Finally station 10 searchers on X_{40}. This leaves 6 free searchers, who can be used to clear the C_i and the D_i. Thus, $\mathrm{cs}(X) \leqq 16$. In the same manner as the proof that 289 searchers are insufficient to clear W in Theorem 5.1, it can be shown that 15 searchers are insufficient to clear X.

To obtain a monotonic internal search, first place 6 searchers on A_1, and 6 searchers on B_1. The 6 searchers on A_1 can clear the A_i, eventually stationing 5 searchers on $X_{21}[A_{20}]$. The 6 searchers on B_1 can clear the B_i, eventually stationing 5 searchers on $X_{21}[B_{20}]$. Then we may follow the same strategy as the search above. Thus $\mathrm{mis}(X) \leqq 12$.

Let S be a monotonic internal search strategy for X. Again, consider the point when the graph goes from having no cleared vertices to exactly 1 cleared vertex v. Since $\mathrm{cs}(X) = 16$, we know that this search must not be connected. Thus, at some point a vertex w is cleared such that there is no clear path from v to w. If v is in some X_i, and w is in X_i, then there are at least 22 exposed vertices, and as many searchers. If v is in some X_i, and w is in some 5-cliques, then there are at least 16 exposed vertices, and as many searchers. The same is true if v is in some 5-clique and w is in some 10-clique. Finally, we consider if v is in some 5-clique (without loss of generality, some A_i-clique) and w is in some 5-clique.

If v is in A_j, $2 \leqq j \leqq 20$, then when it is cleared its neighbouring vertices must contain at least 6 searchers. Consider the subgraph induced by the cleared edges at this point, and the component that v is in. If this component contains exactly 6 searchers, than as in the proof of Theorem 2.4, each of the exposed vertices adjacent to v must be incident with at least two contaminated edges. Thus none of the six searchers can move without allowing recontamination. When w is cleared, it has at least 5 exposed neighbours. If it contains no more that 5 searchers in the component it belongs to of the subgraph induced by the cleared edges, then none of these 5 searchers may move with allowing recontamination. Thus, there must be another searcher in the graph.

Instead, we consider if v is in A_1. If w is a vertex that is not in B_1, C_{60}, or D_{60}, then we have the same argument as in the previous case. Certainly, each of the components these two vertices belong to in the subgraph induced by the cleared edges must have at least 5 searchers belonging to it. If both have exactly 5, then since these searchers cannot move without recontaminating edges, there must be at least one more searcher. Consider the next edge cleared. If the next edge cleared in not incident with a vertex in either of the cleared components containing v or w, since the minimum degree of this graph is 5, two searchers must be used to clear the edge. This accounts for at least 12 searchers.

Instead, we assume that the next edge cleared in S must be incident with one of the cleared components. The same argument applies to the edge that is next cleared. If it is not incident with either connected component, then the search must use at least 12 searchers. This continues to be true until there is a connected path between v and w. If, in S, the connected components that contain u and v

both continue to "grow" by clearing edges incident with them, then both components must contain at least 6 searchers. We instead consider if only one of the cleared components continues to clear incident edges. Without loss of generality, assume that this component is the one containing v.

We consider how this component could expand. Before the cleared path from w to v is formed, if the component containing v ever contains a cleared vertex x in some X_i, consider the first time it happens. At the point x is cleared, it must have at least 10 exposed neighbours in X_i-cliques. Each must contain a searcher. Since that are also at least 5 searchers in the clique containing w, this accounts for at least 15 searchers. If the cleared component containing v does not contain a cleared vertex of some X_i-clique before the cleared path from v to w is formed, consider the first time that an edge between X_{20} and B_{20} is cleared. (This must occur before the path from v to w is formed. There are 5 vertex-disjoint paths from $X_{21}[A_{20}]$ to A_1. Since there is a cleared vertex in A_1, but none in X_{21}, each of these paths must contain an exposed vertex, and hence a searcher. But if an edge is cleared in between X_{21} and B_{20}, there must be a searcher in B_{20} and another in $X_{21}[B_{20}]$. Since there are at least 5 searchers in the cleared component containing w, this accounts for at least 12 searchers. Thus, every monotonic internal search strategy on X must use at least 12 searchers. ■

We now consider the graph X'.

6.3 Lemma. *For the graphs X and X', $\mathrm{cs}(X) > \mathrm{cs}(X')$.*

Proof. Since X' contains $K_{10}\square K_2$ as a minor, we know by Theorem 2.10 and Corollary 2.18 that $\mathrm{cs}(X') \geqq 11$. In fact, we can use 11 searchers to in a connected search to clear X', by placing 10 searchers on X_1, and then using a single free searcher to clear all the edges in X_1. Then 10 searchers move along the perfect matching to X_2, and a single free searcher clears all the edges in X_2, and so on, finally reaching X_{60}. Thus, $\mathrm{cs}(X') = 11$.

From the proof of Theorem 6.2, we know that $\mathrm{cs}(X) = 16$, and the result follows. ■

Since X is a subgraph of X', this lemma has an immediate consequence, as observed in [4]. If H is a minor of a graph J, then in contrast to Theorem 2.10, it does not follow that $\mathrm{cs}(H) \leqq \mathrm{cs}(G)$.

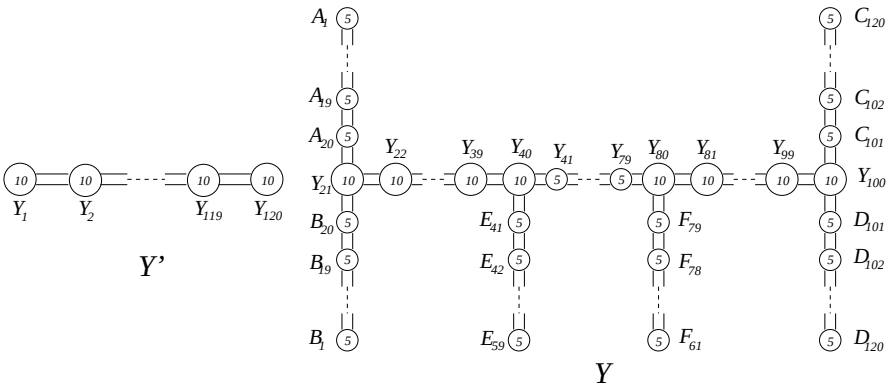

Figure 4: The graph Y' and its subgraph Y.

Continuing in the same vein, let $Y' = K_{10}\square P_{120}$, and Y be as pictured in Figure 4, where circles and double lines are defined as above. It is easy to see that Y is a subgraph of Y'.

6.4 Theorem. *For graphs Y and Y' as given, $\mathrm{mis}(Y) > \mathrm{mis}(Y')$.*

Proof. We first note that $K_{10}\square K_2$ is a subgraph of Y', and thus $11 \leqq \mathrm{mis}(Y')$. Also, Y' can be cleared using the same strategy as used for X' in Lemma 6.3. Thus, $\mathrm{mis}(Y') = 11$.

The graph Y can be cleared by 16 searchers in a monotonic internal fashion. Place 16 searchers on A_1. Use 6 searchers to clear the A_i, stationing 10 searchers on Y_{21}. Use the six remaining searchers to clear the B_i. Clear to Y_{40}, stationing 10 searchers on Y_{40}. This leaves 6 free searchers that can be used to clear the E_i. Then the searchers may clear to Y_{80}, stationing 10 searcher there. The remaining 6

free searchers may clear the F_i. Then all the searchers may clear to Y_{100}, stationing 10 searchers there. The 6 remaining searchers may clear the C_i and then the D_i. Thus, $\text{mis}(Y) \leqq 16$. In the same manner as the proof that 289 searchers are insufficient to clear W in Theorem 5.1, it can be shown that 15 searchers are insufficient to clear X. ■

As before, since Y is a subgraph of Y', there is an immediate corollary, as observed in [4]. In contrast to Theorem 2.10, if H is a minor or G, then it does not follow that $\text{mis}(H) \leqq \text{mis}(G)$.

Recall that there are three inequalities in Lemma 1.1. Corollary 6.1 shows that the final inequality can be strict, and Theorem 6.2 shows that the first pair may also be strict. This leads us to construct a single graph H for which the three inequalities strictly hold (see Figure 5).

6.5 Theorem. *For the graph H as given,* $\text{s}(H) < \text{mis}(H) < \text{cs}(H) < \text{mcs}(H)$.

This graph H has $\text{s}(H) = 561$, $\text{mis}(H) = 570$, $\text{cs}(H) = 841$, and $\text{mcs}(H) = 850$. The proofs of these claims follow in the same manner as the proofs of Theorems 4.2, 5.1, and 6.2.

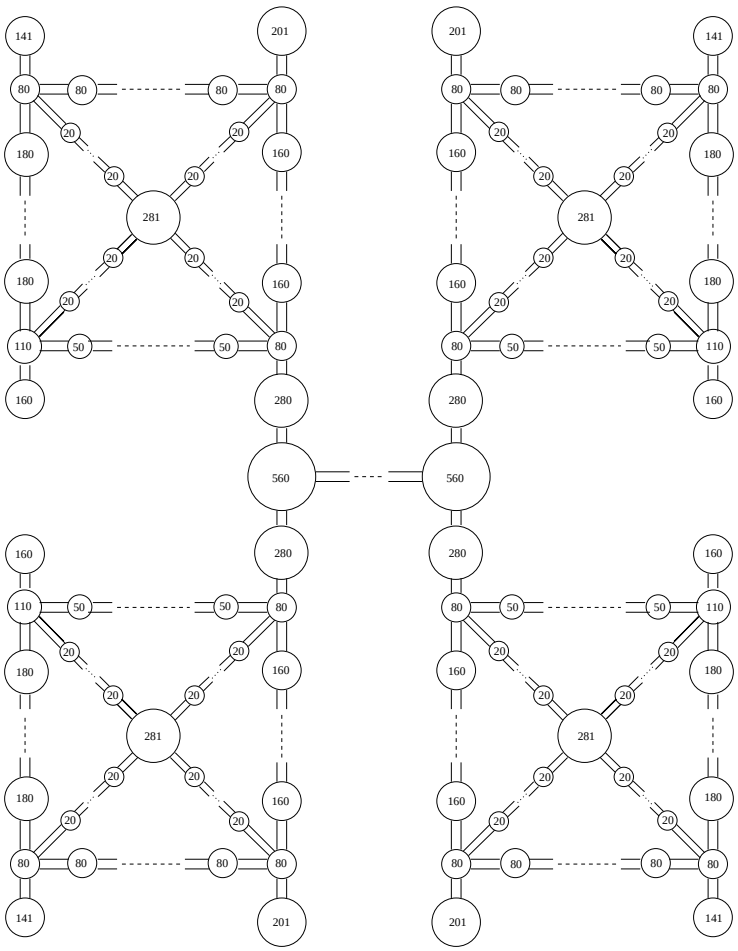

Figure 5: The graph H.

6.2 Differences between search numbers

In Corollary 6.1 we constructed a graph W with $\text{cs}(W) < \text{mcs}$. The graph W is quite large, containing almost $400,000$ vertices. The large strings of 300 identically sized cliques were constructed so that they

would be impossible to "sneak" through, and the C_i and D_i were constructed so that they could be "sneaked" through.

We showed that the difference between $\mathrm{cs}(W)$ and $\mathrm{mcs}(W)$ was 9, though in fact, the difference can be much smaller. For instance, we could reduce the size of cliques and length of "paths" by an approximate factor of 5 and then prove that $\mathrm{cs}(W_{\frac{1}{5}}) = 57$ and $\mathrm{mcs}(W_{\frac{1}{5}}) = 58$. (This corresponds to letting $k = \frac{1}{5}$ in the following construction.) However, these results, while valid, are more easily demonstrated by using larger cliques and longer paths to make the difference more believable.

From the graph W, we may define a family W_k of graphs constructed as follows for $k \geqq 1$. In W_k, the 300 cliques of the R_i (and R'_i) are replaced by $300k$ cliques of order $180k$. Similarly, the T_i (and T'_i) are replaced by $300k$ cliques of order $80k$, the B_i (and B'_i) are replaced by $300k$ cliques of order $50k$, the L_i (and L'_i) are replaced by $300k$ cliques of order $160k$, the E_i (and E'_i) are replaced by $300k$ cliques of order $20k$, the F_i (and F'_i) are replaced by $300k$ cliques of order $20k$, and the G_i are replaced by $300k$ cliques of order $280k$. The cliques A_2, A_5, A_6, A'_2, A'_5, and A'_6 are replaced by cliques of order $80k$. The cliques A_4 and A'_4 are replaced by cliques of order $110k$. The cliques A_3 and A'_3 are replaced by cliques of order $160k$. The cliques A_1 and A'_1 are replaced by cliques of order $280k$. The 19 cliques that make up the C_i are replaced by $20k - 1$ cliques of order $20k$. Similarly for the C'_i, D_i, and D'_i. The cliques A_8 and A'_8 are replaced by cliques of order $200k + 1$, the cliques A_7 and A'_7 are replaced by cliques of order $140k + 1$, and the cliques A_9 and A'_9 are replaced by cliques of order $280k + 1$.

The resulting graphs W_k are "scaled up" version of W, with appropriately changed search numbers. Using an argument similar to those in the proofs of Theorems 4.2 and 5.1, we can prove the following theorem.

6.6 Theorem. *For $k \geqq 1$, $\mathrm{cs}(W_k) = 280k + 1$, and $\mathrm{mcs}(W_k) = 290k$.*

In the same manner, we may create families of graphs X_k (and X'_k) and Y_k (and Y'_k) based on X (and X') and Y (and Y'). This is done by replacing cliques of order 5 with cliques of order $5k$ and cliques of order 10 with cliques of order $10k$, and lengthening "paths" of cliques of the same order by a factor of k. (For instance, in X, rather than having 20 cliques of order 10 make up the X_i, they would be replaced by $20k$ cliques of order $10k$.) The results for these families are summarized below.

6.7 Theorem. *For $k \geqq 1$, $\mathrm{cs}(X_k) = 15k + 1$; $\mathrm{mis}(X_k) = 10k + 2$; $\mathrm{cs}(X'_k) = 10k + 1$; $\mathrm{mis}(Y) = 15k + 1$; and $\mathrm{mis}(Y') = 10k + 1$.*

This result tells us that the difference between the monotonic internal search number of a graph and the connected search number can be large. As well, the results tell us that in the case of connected and monotonic internal searchs, a subgraph may need arbitrarily more searchers than a supergraph.

Finally, the graph in Figure 6 is a similarly "scaled up" version of the Y-square (pictured in Figure 1). Here, edges are replaced by "paths" of cliques, with each path containing k^2 cliques of size k. This increases the search number to $3k + 1$, and the monotonic internal search number to $4k$, again showing that the difference in these values can be quite large.

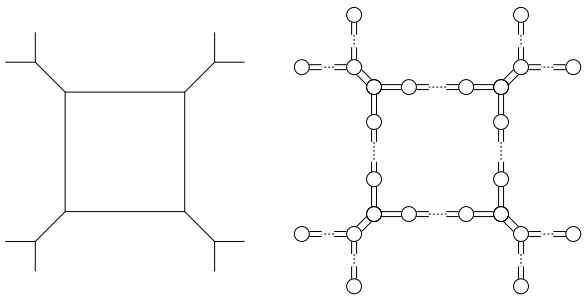

Figure 6: The kY-square.

6.3 Variation on the required number of searchers

We know the maximum number of exposed vertices in a connected graph is at most the search number of the graph. Of course, most of the time, the number of exposed vertices is much less than this maximum. In a real world situation, most searchers not on exposed vertices could "go away," and would only return when needed. So we would be interested in search strategies that minimizes the number of exposed vertices at each step. For a graph G, the sequence of $\mathrm{ex}_S(G, i)$ for any S could vary greatly. The following theorem illustrates just how great this variance can be.

We first construct a graph Z as pictured in Figure 7, where the a_i are positive integers, and $M = \max_{1 \leqq i \leqq n} a_i + 5$. (The value 5 is added as a safety margin.)

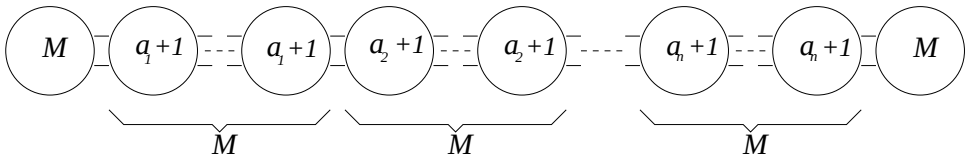

Figure 7: The graph Z.

6.8 Theorem. *Given a finite sequence of positive integers, $a_1, a_2, \ldots, a_n$, then with Z as given, every monotonic connected search strategy S of Z that uses* $\mathrm{mcs}(Z)$ *searchers and minimized the number of exposed vertices at each step has the property that there exists $k_1 < k_2 < \cdots < k_n$ such that $\mathrm{ex}_S(Z, k_i) \geqq a_i$.*

Proof. Since Z contains an M-clique, we know that $\mathrm{s}(Z) \geqq M$. Further, there is a monotonic connected search strategy using M searchers. First, clear M, then the first $a_1 + 1$ clique, then the next, and so on, moving from left to right. Thus, $\mathrm{mcs}(Z) = M$.

Let v be the first vertex cleared in a monotonic connected search strategy S on Z using M searchers. If v is not in one of the M-cliques, then there is a cleared edge e in some other clique. There are at least two vertex-disjoint paths that pass through the vertices of e to either M. The first time that a vertex is cleared in either M-clique, there are $M - 1$ searchers on that clique. But at the same time, there are two vertex-disjoint paths from the vertices of e to the other M-clique. These two paths must contain at least one exposed vertex, and hence one searcher. But this search then uses $M + 1$ searchers. Thus, v must be in one of the M-cliques.

Let $i < j$. Consider a_i and a_j. Assume that no $a_i + 1$ clique obtains a cleared vertex before all the $a_j + 1$ cliques. Let w be a cleared vertex in one of the $a_j + 1$ cliques. Since S is a connected search strategy there is a cleared path between v and w. But this pass must pass through the $a_i + 1$ cliques, of which there are $M + 1$. Since these cliques contain no cleared vertices, there must be at least one exposed vertex in each of the $a_i + 1$ cliques, and hence at least $M + 1$ searchers in the $a_i + 1$ cliques. Since this uses too many searchers, some $a_i + 1$ clique must contain a cleared vertex before all the $a_j + 1$ cliques do. When the $a_i + 1$ clique first contains a cleared vertex, there are at least a_i exposed vertices. ■

7 Conclusions

The main purpose of this paper was to demonstrate the increased ease of proof allowed by constructing graphs with large clique number. This has been adequately demonstrated through our constructions of the graphs W, X, Y and Z, as well as H and the kY-square, which have enable us to solve the open problem of the existence of a graph where the connected search number and monotonic connected search number differ. Besides the fact that large cliques in a graph imply lower bounds on the search number, they also allow us to restrict how a graph is cleared by setting up situations where "paths" of cliques must be cleared clique by clique rather than "sneaking" through them.

Having solved one of the open problems in [4], we are compelled to mention the other: find an upper bound for the ratio $\mathrm{mcs}(G)/\mathrm{s}(G)$. For the special case of trees, the bound was discussed in [4], and the authors believe that it is true for all connected graphs.

As mentioned in the introduction, the Y-square is the smallest known graph with search number strictly less than monotonic internal search number. For the other inequalities, we have given examples of graphs which show that the inequalities can be strict, but these graphs are very large. Smallest graphs with these properties would be interesting to find to examine underlying structures.

While constructing the graph W and demonstrating that $\mathrm{cs}(W) < \mathrm{mcs}(W)$, many additions had to be made to W to simplify the proof. Essentially, these were made so that any search strategy would go from the clique A'_9 to the clique A_9, or vice versa; obviously, because of the strong symmetry in the graph, the search could go both ways. We pose the following problem relating to search structure: is there a graph G such that every monotonic connected search of G using $\mathrm{mcs}(G)$ searchers must start at a particular vertex set U and end at a particular vertex set V, with $U \cap V = \emptyset$?

Acknowledgements:

The authors would like to extend special thanks to Denis Hanson and Xiangwen Li both for their ongoing support and for the fascinating discussion they have provided in our weekly MITACS seminars.

References

[1] B. Alspach, Searching and sweeping graphs: A brief survey, Le Matematiche, 34pp, to appear.

[2] B. Alspach, X. Li, and B. Yang, *Searching Graphs and Directed Graphs*, Manuscript, 194 pages, 2004.

[3] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, Capture of an Intruder by Mobile Agents , Proc. of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA 2002), pp. 200-209, 2002.

[4] L. Barrière, P. Fraigniaud, N. Santoro and D. Thilikos, Searching is not Jumping. In Proc. 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003), Lecture Notes in Computer Science, 2880, pp. 34-45, 2003.

[5] D. Bienstock and P. Seymour, Monotonicity in graph searching, *Journal of Algorithms* **12** (1991), pp. 239–245.

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

[7] J. Ellis, I. Sudborough and J. Turner, The vertex separation and search number of a graph, *Information and Computation* **113** (1994), pp. 50–79.

[8] M. Fellows and M. Langston, On search, decision and the efficiency of polynomial time algorithm. *21st ACM Symp. on Theory of Computing (STOC 89)*, (1989) pp. 501–512.

[9] M. Frankling, Z. Galil, and M. Yung. Eavesdropping games: A graph-theoretic approach to privacy in distributed systems, *Journal of ACM*, **47** (2000), pp. 225–243.

[10] L. M. Kirousis and C. H. Papadimitriou, Searching and pebbling, *Theoret. Comput. Sci.* **47** (2) 1986), pp. 205–218.

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

[12] N. Megiddo, S. L. Hakimi, M. Garey, D. Johnson and C. H. Papadimitriou, The complexity of searching a graph. *Journal of ACM* , **35** (1988), pp. 18–44.

[13] R. J. Nowakowski and N. Zeh, Boundary-Optimal Triangulation Flooding. (Preprint).

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

[15] R. Tošić, The search number of the Cartesian product of graphs, Univ. Novom Sabu Zb. Rad. Prirod.-Mat Fak. Ser. Mat. **17** (1987), pp. 239-243.

[16] D.B. West, *Introduction to Graph Theory*, Prentice-Hall, p. 136, 1996.

[17] B. Yang, D. Dyer, and B. Alspach, Sweeping graphs with large clique number (extended abstract), Proceedings of 15th International Symposium on Algorithms and Computation, Rudolf Fleischer and Gerhard Trippen Eds., Lecture Notes in Computer Science, **Vol. 3341**, Springer-Verlag, Berlin (2004), pp. 880–892.