Megiddo, Nimrod; Hakimi, S.L.

Working Paper

Pursuing Mobile Hiders in a Graph

Discussion Paper, No. 360

Provided in Cooperation with:
Kellogg School of Management - Center for Mathematical Studies in Economics and Management Science, Northwestern University

Suggested Citation: Megiddo, Nimrod; Hakimi, S.L. (1978) : Pursuing Mobile Hiders in a Graph, Discussion Paper, No. 360, Northwestern University, Kellogg School of Management, Center for Mathematical Studies in Economics and Management Science, Evanston, IL

This Version is available at:
https://hdl.handle.net/10419/220720

DISCUSSION PAPER NUMBER 360

Pursuing Mobile Hiders in a Graph[1]

by

Nimrod Megiddo[2] and S. L. Hakimi[3]

December, 1978

[1]This work was suppored in part by U.S. Air Force Office of Scientific Research, Systems Command, under Grant AFOSR-76-3017.

[2]Graduate School of Management, Northwestern University, Evanston, Illinois 60201, and Department of Statistics, Tel Aviv University, Tel Aviv, Israel.

[3]Department of Electrical Engineering and Computer Science, and Department of Engineering Science and Applied Mathematics, Northwestern University, Evanston, Illinois, 60201.

ABSTRACT

The pursuit-evasion game considered here takes place in an undirected graph. The central problem is how many pursuers are required to guarantee the capture of all evaders, and how should the pursuers act. A linear-time algorithm for this problem is developed for tree-graphs. The difficulty of the problem is demonstrated by a characterization of graphs that require no more than two pursuers and by examples of minimal graphs requiring more than three pursuers.

1. Introduction

Pursuit-evasion problems have been thoroughly studied in the context of differential games, as initiated by Isaacs [3]. In all these models, velocities and their derivatives play essential roles. In certain versions of these problems, where there is some lack of information (on the part of the pursuers, for example) these problems have a probabilistic flavor. In those cases, when the pursuers have no information as to the exact location of the evader, the latter is called a hider ([1]).

When the game takes place in a one-dimensional domain (say, a graph), and there is more than one pursuer, then the combinatorial and the computational aspects become more dominant. Asel'derova [2], for example, deals with such a game on a graph with edge-lengths, assuming certain speed limits.

Our present paper deals with a pursuit-evasion game which is purely combinatorial (both the differential and the probabilistic elements are eliminated). The model was first suggested by Parsons [4,5]. A definition is given below.

For our purpose it is convenient to define a graph as a union of a finite number of closed straight line segments in some euclidean space. Thus, a subgraph of a graph $G \subset R^n$ is a subset of G which itself is a graph, namely, consists of a finite number of closed straight line segments. Obviously,

if a graph is given in the combinatorial form of vertices and edges, then it can be embedded in a euclidean space[4] so as to conform with our definition. Multiple edges and loops may be handled by means of introducing additional vertices. Henceforth we shall conceive a graph both as a subset of a euclidean space and as a collection of pairs of vertices.

Our game takes place within a connected graph. Each one of the evaders has the capability of moving with no speed limit within the graph. The pursuers have some finite speed limit. Furthermore, the pursuers have no information as to the location of any evader, unless the latter has already been captured. We shall henceforth use the term hider instead of evader. Capture in this game means collision. The central question considered here is how many pursuers are required to guarantee capture of all hiders, and how should the pursuers act.

It can be shown that the answer to this question is independent of the number of hiders, the lengths of the line segments and the speed limit of the pursuers.

Numerous applications could be mentioned. One can think of a search of a city for the mobile headquarters of a terrorist organization, or for some stolen merchandise. Other applications are to purifying a contaminated transportation network in the event of a chemical warfare, to the control of water pollution, and to the inspection of motorized vehicles.

---

[4] A 3-dimensional space suffices.

In Section 2 we describe a linear time algorithm for finding the minimum number of pursuers (the search number) required for a tree graph. The algorithm computes a search plan as well. In Section 3 we describe a method to compute the search number of a general graph. This method is then applied to derive some bounds for the search number. In Section 4 we characterize graphs with search number 2 (via minimal graphs of search number 3). Section 5 provides examples of minimal graphs with search number 4.

2. Pursuit-evasion on trees.

In this section our graph is a tree (i.e. it is connected and has no cycles) and is denoted by T. The number of pursuers required to guarantee capture is denoted by s(T) and is called the "search number" of T. We develop here a linear-time algorithm for finding s(T) as well as a supporting search plan.

There does not seem to exist a simple rule that states how essentially should a tree be searched. The tree shown in Figure 1 may be a good example.

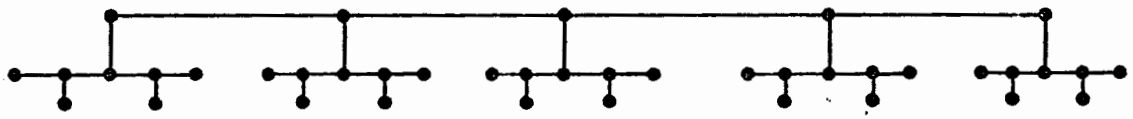

Fig. 1

This tree has search number 3. The three pursuers should act as follows. They have to search the five subtrees, one at a time, while one of them is guarding at the root of the subtree. The other two clear the subtree, starting from the two left-hand leaves, say, and ending at the two right-hand leaves.

We find it convenient to define several other parameters of the tree that are related to the search problem, and then compute the values of these parameters and s(T) simultaneously. Even though our game is played over an undirected tree, we will arbitrarily fix one vertex r as the root of the tree. This defines for every vertex x a subtree T(x) rooted at x, namely, the subtree of T induced by all vertices y such that x lies on the path from y to the root r of T.

Definition of s*(T): If T is a tree rooted at r then by s*(T) we mean the search number s(T*) of a tree T* obtained by adjoining to T one more vertex and one more edge incident upon r.

Fact 1: $$s(T) \leq s^*(T) \leq s(T) + 1.$$

We shall now define some modified versions of our pursuit-evasion game, from which we derive additional parameters of T that are useful in the computation of s(T). By Version 1 we mean the original game over T.

Version 2: This game is basically the same as the original one but with the following exception. We say that the hider

"escapes" (i.e. cannot be captured anymore) if he reaches the root r when none of the pursuers is there.

Definition of $u(T)$: The minimal number of pursuers required to guarantee capture in Version 2.

Note that in order to guarantee capture in Version 2, one pursuer must start at the root. Also, it is sufficient to place a "guard" at the root and employ $s(T)$ pursuers as in Version 1.

Fact 2: $$s^*(T) \leq u(T) \leq s(T) + 1.$$

Version 3: Basically the same as Version 1, but with the following exception. The hider is free to get through the root in and out of the tree. However, the rules are such that he is considered captured if he is located outside of the tree when the search is terminated. Thus, the hider may use the outside as a temporary shelter.

By reversing the direction of time, it can be shown that the minimal number of pursuers required in this game, is equal to $u(T)$.

Definition of $w(T)$. Given two rooted trees T_1, T_2 whose sets of vertices are disjoint, we define the composition $T_1 + T_2$, to be the tree obtained by connecting the two roots by means of a new edge. Now we define

$$w(T) = \mathrm{Max}\{u(T') : s(T+T') = s(T)\}.$$

Intuitively, $w(T)$ is the maximal number of pursuers, out of the $s(T)$ who are searching T, that could be assigned to external search tasks, starting at the root of T and carried out during the search of T.

Fact 3: If $s(T) = u(T)$ then $w(T) = s(T)$.
(This is because the search of T, in case $s(T) = u(T)$, can be carried out with all pursuers terminating at the root, and then all of them can start another task at the root.)

Fact 4: If $s^*(T) = s(T) + 1$ then $w(T) = 0$.
(This is because adjoining just a single edge to T requires an additional pursuer.)

Assuming that T is the given rooted tree and x is any vertex of T, we will in short denote $s(x)$, $s^*(x)$, $u(x)$ and $w(x)$, for $s(T(x))$, $s^*(T(x))$, $u(T(x))$ and $w(T(x))$, respectively. We will show that once the values of $s(y)$, $s^*(y)$, $u(y)$ and $w(y)$ are known for every son y of a vertex x, then the values of $s(x)$, $s^*(x)$, $u(x)$ and $w(x)$ can be easily calculated. The case of x being a leaf is trivial. Also, without loss of generality, we assume that there are no vertices of degree 2 in T. Henceforth, we assume that x is a vertex that has at least two sons. We denote the set of sons of x by X. The evaluation of the four functions requires distinguishing cases. The

different cases are covered in the following assertions.

Assertion 1. If there are at least three sons y_1, y_2, y_3 of x such that $s^*(y_i) = \text{Max}\{s^*(y):y\in X\}$ $(i=1,2,3)$ then

$$s(x) = s^*(x) = u(x) = w(x) = s^*(y_1) + 1.$$

Proof: We rely on a lemma proved by Parsons [4,Lemma 4]: "Let T_1, T_2, T_3 be disjoint trees each having at least one edge. For $j=1,2,3$ let v_j be a vertex of degree 1 in T_j. Let T be the tree obtained by identifying v_1, v_2, v_3 into a single vertex v. If $s(T_j) = k$ for $j=1,2,3$, then $s(T) = k+1$." It follows that in our case $s(x) = s^*(y_1) + 1$. Furthermore, in an optimal search plan for T(x), one of the pursuers is positioned at x throughout. This implies the equalities asserted. □

Assertion 2. If there are precisely two sons, y_1 and y_2, of x such that $s^*(y_1) = s^*(y_2) = \text{Max}\{s^*(y):y\in X\}$, and they satisfy $u(y_i) = s^*(y_i)$, $i=1,2$, then $s(x) = s^*(y_1)$, $s^*(x) = \text{Max}\{s^*(y_1),2\}$, $u(x) = s^*(y) + 1$ and $w(x) = s^*(y_1) - 1$.

Proof: Following is a search plan for T(x) with $s^*(y_1)$ pursuers. First, let all $s^*(y_1)$ pursuers clear the subtree $T(y_1)$, in such a way that when they terminate at y_1, the hider is known to be outside of $T(y_1)$. This is possible in the light of the assumption $u(y_1) = s^*(y_1)$ and the discussion of Version 3. Next, let all $s^*(y_1)$ pursuers proceed from y_1 to x. Now, place one of them as a guard at x and let the others clear all subtrees

$T(y)$, $y \in X \setminus \{y_1, y_2\}$, one at a time, if any. This is possible since $s^*(y) \leq s^*(y_1) - 1$ for $y \in X \setminus \{y_1, y_2\}$. Finally, let all $s^*(y_1)$ pursuers proceed from x to y_2 and clear the subtree $T(y_2)$, making sure that the hider does not reach y_2. This is possible in the light of the assumption $s^*(y_2) = u(y_2)$ and the definition of $u(y_2)$. Thus, $s(x) = s^*(y_1)$.

It follows from the search plan we described, that if $s^*(y_1) \geq 2$, then $s^*(x) = s(x)$. On the other hand, if $s^*(y_1) = 1$, then by Parsons' lemma (see the proof of Assertion 1) $s^*(x) = 2$. Thus, $s^*(x) = \text{Max}\{s^*(y_1), 2\}$. To compute $u(x)$, note that during a search of $T(x)$ there must be an interval of time, during which all the $s(x)$ pursuers are located inside $T(y_1)$. There also must be an interval of time during which all the $s(x)$ searchers are located inside $T(y_2)$. Thus, when playing Version 2, one additional guard is required for preventing the hider from escaping by reaching x. In other words, $u(x) = s(x) + 1 = s^*(y_1) + 1$. Moreover, since all the $s(x)$ pursuers can proceed together through x during the search, it follows that $w(x) = s(x) - 1 = s^*(y_1) - 1$. □

Assertion 3: If there are precisely two sons, y_1 and y_2, of x, such that $s^*(y_1) = s^*(y_2) = \text{Max}\{s^*(y): y \in X\}$, and if $u(y_1) = s^*(y_1) + 1$ then

$$s(x) = s^*(x) = u(x) = w(x) = s^*(y_1) + 1.$$

Proof: First, $s^*(y_1) + 1$ pursuers certainly suffice for Versions 1, 2, 3. This is because, one may place a guard at x and let the other $s^*(y_1)$ pursuers clear the subtrees $T(y)$, $y \in X$, one at a time. Secondly, note that $s^*(y_1)$ pursuers must spend a positive amount of time inside $T(y_1)$ and a positive amount of time inside $T(y_2)$. However, since $s^*(y_1) < u(y_1)$, it cannot be guaranteed(unless we employ at least $s^*(y_1) + 1$ pursuers) that the hider has not sneaked back into $T(y_1)$ while the sursuers were leaving this subtree. (This corresponds to Version 3 played on $T(y_1)$). Nor can it be guaranteed that the hider has not sneaked out of $T(y_1)$ while the pursuers were entering this subtree. (Think of Version 2 played on $T(y_1)$). Thus, the presence of an additional guard at x is necessary: $s(x) = u(x) = s^*(y_1) + 1$. It follows from Facts 1-3 that also $s^*(x) = w(x) = s^*(y_1) + 1$. □

Assertion 4. *If there is precisely one son y_1 of x such that $s_1^*(y) = \text{Max}\{s^*(y): y \in X\}$ and if $u(y_1) = s^*(y_1)$ then* $s(x) = s^*(x) = u(x) = w(x) = s^*(y)$.

Proof: Following is a search plan for $T(x)$ with $s^*(y_1)$ pursuers. First, clear $T(y_1)$ as if Version 3 were played on this subtree (note that $u(y_1) = s^*(y_1)$). Then let all $s^*(y_1)$ pursuers proceed to x. Next, one of them is positioned at x and all the others clear the subtrees $T(y)$, $y \in X \setminus \{y_1\}$ one at a time. This plan implies our claim. □

If y_1 is a son of a vertex x then by $T(x/y_1)$ we mean the subtree rooted at x and consisting of all subtrees T(y) and edges (x,y) for $y \in X \setminus \{y_1\}$.

Assertion 5. Suppose that there is precisely one son y_1 of x such that $s^*(y_1) = \text{Max}\{s^*(y): y \in X\}$ and this son satisfies $u(y_1) = s^*(y_1) + 1$. Under these conditions:

$$s(x) = \begin{cases} s^*(y_1) + 1 & \text{if} \quad w(y_1) < u(T(x/y_1)) \\ s^*(y_1) & \text{otherwise,} \end{cases}$$

$$s^*(x) = \text{Max}\{s(x), 2\},$$

$$u(x) = s^*(y_1) + 1, \quad \text{and}$$

$$u(x) = \begin{cases} s^*(y_1) + 1 & \text{if } w(y_1) < u(T(x/y_1)) \\ w(y_1) & \text{if } w(y_1) = u(T(x/y_1)) = \text{Max}\{u(y): y \in X \setminus \{y_1\}\} \\ w(y_1) - 1 & \text{if } w(y_1) > \text{Max}\{u(y): y \in X \setminus \{y_1\}\}. \end{cases}$$

Proof: First, observe that the search plan described in the proof of Assertion 4 implies that $u(x) = u(y_1) = s^*(y_1) + 1$. Next, by the definition of w, $s(x) = s^*(y_1)$ if and only if $w(y_1) \geq u(T(x/y_1)))$. Thus, if $w(y_1) < u(T(x/y_1))$ then $s(x) = s^*(y_1) + 1$, since one additional pursuer acting as a guard at x certainly suffices. This also implies that if $w(y_1) < u(T(x/y_1))$ then $s^*(x) = w(x) = s^*(y_1) + 1$. We still have to deal with $s^*(x)$ and $w(x)$ in case $w(y_1) \geq u(T(x/y_1))$.

It is easy to verify that if $w(y_1) = 1$ then $s^*(x) = s(x) + 1$ and if $w(y_1) \geq 2$ then $s^*(x) = s(x)$.

For the evaluation of $w(x)$ consider first the case $w(y_1) > \text{Max}\{u(y): y \in X \setminus \{y_1\}\}$. In this case $w(x) = w(y_1)$ by the following plan. During the search of $T(y_1)$, $w(y_1)$ pursuers may leave through y_1. One of them is placed as a guard at x, while the others clear the subtrees $T(y)$ $(y \in X \setminus \{y_1\})$. Then all the $w(y_1)$ searchers may leave through x.

If there is precisely one son $y_2 \in X \setminus \{y_1\}$ such that $w(y_1) = \text{Max}\{u(y): y \in X \setminus \{y_1\}\} = u(y_2)$, then $T(x/y_1)$ should be cleared by $u(y_2)$ pursuers in the following way. Place a guard at x, let the others clear the subtrees $T(y)$ for $y \in X \setminus \{y_1, y_2\}$, one at a time, if any, and then let all $u(y_1)$ pursuers enter $T(y_2)$. This implies that in this case $w(x) = w(y_1) - 1$.

Finally, if there are at least two sons $y_2, y_3 \in X \setminus \{y_1\}$ such that $w(y_1) = \text{Max}\{u(y): y \in X \setminus \{y_1\}\} = u(y_2) = u(y_3)$ then $u(T(x/y_1)) = w(y_1) + 1$ and this has already been analyzed. This completes the proof of our assertion. □

Assertions 1-5 lead to a linear-time algorithm for evaluating the values $s(T)$, $s^*(T)$, $u(T)$ and $w(T)$. It should be noticed that in all the cases, the information about the values of these functions at all $y \in X$ suffices for evaluating them at x. In the case of Assertion 5, the value $u(T(x/y_1))$ has to be computed, but that is possible by the rules of the function u.

The actual search plans associated with these four functions are implicit in the computation based on our case-analysis.

3. Some general results

3.1 Computing s(G) in the general case

Parsons [4] originally formulated the search problem in terms of finding continuous functions that describe the motion of the different pursuers. These functions have to guarnatee capture regardless of the continuous function chosen by the hider. Thus, it was not clear whether or not the search number was even computable.

Later, Parsons [5] reformulated the problem in a way that led to an algorithm for computing the search number s(G) of a given graph. We describe below another algorithm for finding s(G). Our algorithm seems to shed some more light on the problem, and is shown to be useful for proving results about the search number.

Suppose that we have a valid search plan for a graph G = (V,E) with k pursuers. At any time t the point set of G (recall that G is conceived also a subset of some euclidean space) is partitioned into two subsets, namely, the set of clear points and the set of unclear points. Specifically, a point x is clear at time t,if it follows from the search plan that the hider cannot be located at x,from time t through the end of the search. At the start, the clear points can be only

initial positions of the pursuers. At the end, every point is clear. Moreover, at any time t the two subsets are separated by some of the points that are currently occupied by pursuers.

It follows from this observation, that for every edge e of the graph, there is a time t(e) when e becomes entirely clear. Without loss of generality, we may assume that during the search, one pursuer moves at a time. Thus, one edge becomes clear at a time. In other words, every valid search plan induces a linear order over the set of edges E, namely, the order in which the edges become entirely clear. Thus, the problem of finding the search number reduces to finding the optimal order in which the edges have to be cleared. However, given a sequence, $p = (e_1,\ldots,e_m)$ of the edges of G, it is easy to find the minimal number k(p) of pursuers that are required for searching G in such a way that the edges become clear in the order $e_1,\ldots,e_m$. An algorithm for computing k(p) is described in the Appendix. It follows that the question whether or not $s(G) \leq k$ (given G and k) is decidable in polynomial time on a non-deterministic Turing machine (i.e. it belongs to the class NP) and in exponential time on a deterministic machine. We have not as yet been able to prove NP-completeness of the problem.

3.2 Searching a complete graph

The observation, that the search number problem may be stated as a problem of finding an optimal permutation of the edges, is shown to be useful by the following results.

Let K_n denote the complete graph with n vertices. Parsons [4,5] stated without proof (except for the case n=4) that for $n \geq 4$, $s(K_n) = n$. It is easy to see that $s(K_n) \leq n$, since n pursuers can guarantee capture in the following way. They all start at vertex 1. Then, pursuer i (i=2,3,...,n) proceeds from vertex 1 to vertex i and occupies vertex i. Finally, pursuer 1 clears all edges (i,j), for $2 \leq i, j \leq n$.

We shall now prove that n-1 pursuers do not suffice for K_n (if $n \geq 4$). Suppose, to the contrary, that n-1 pursuers do suffice for K_n. Thus, they can clear the edges in some order $\pi = (e_1,\ldots,e_m)$ $(m = \frac{1}{2} n(n-1))$. For every vertex v $(1 \leq v \leq n)$, let i(v) be the index such that $e_{i(v)}$ is the first edge in the sequence π that is incident upon v. Without loss of generality assume that $i(1) \leq i(2) \leq \ldots \leq i(n)$. Consider the situation when $e_1,\ldots,e_{i(n-1)}$ are the clear edges. Obviously, each one of the vertices 1,2,...,n-1 has both a clear and an unclear edge incident upon it, namely, $e_{i(v)}$ is clear and (v,n) is unclear (v=1,...,n-1). Thus, the n-1 pursuers must currently be positioned at the vertices 1,...,n-1. Suppose that $e_{i(n-1)} = (x,n-1)$ $(1 \leq x \leq n-2)$. Thus, for every j $(1 \leq j \leq n-2, j \neq x)$ the edge (j,n-1) is unclear. In other words, x is the only vertex

that may have no more than one unclear edge incident upon it. Hence, $e_{i(n-1)+1} = (x,n)$. We arrive at the following contradiction. When the edges $e_1,\ldots,e_{i(n-1)+1}$ are the clear ones, the pursuers must be located at the vertices $j (j=1,\ldots,n,\ j \neq x)$. However, each one of these vertices has at least two unclear edges incident upon it, so that the pursuer positioned at j cannot move. □

3.3 Some bounds on s(G)

Let max-clique(G) denote the maximal number k such that G contains a subgraph homeomorphic to the complete graph K_k.

Proposition 1. If max-clique(G) $\geq$ 4 then s(G) $\geq$ max-clique(G).

This is a direct consequence of our result in Section 3.2. The following bound can be proved very similarly, hence we omit the proof.

Proposition 2. If G is neither a simple path nor a simple cycle then s(G) $\geq$ min-degree(G) + 1.

(By min-degree(G) we mean the minimum degree of a vertex in G.)

Let vertex-cover(G) denote the minimum cardinality of a set of vertices of G such that each edge is incident upon a vertex in the set.

Proposition 3: $s(G) \leq \text{vertex-cover}(G) + 2$.

Proof: Consider the connected components of G obtained by removing all the vertices of some covering set (however, no edge is removed). Each one of these components is a tree with diameter not greater than 2. Thus, by placing guards at vertices of a covering set, we need no more than two additional pursuers, who will clear all these components one at a time.

Proposition 4: If T is a tree with n vertices then $s(T) \leq \log_3(n-1)+1$.

This follows from our algorithm in Section 2 and may be proved directly by induction. Moreover, this bound is tight, as is shown by the trees defined recursively as follows. Let T_1 be a single edge tree. Construct T_{k+1} by identifying (coalescing) three leaves, one from each one of three disjoint copies of T_k. The first three trees in the sequence are shown in Figure 2.

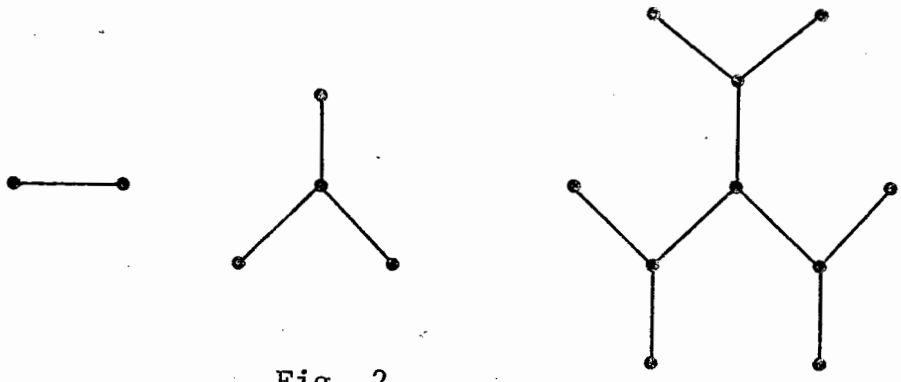

Fig. 2

It can be verified that $s(T_k) = k$ and T_k has $1 + 3^{k-1}$ vertices.

4. Graphs with search number 2

Consider the three graphs shown in Figure 3:

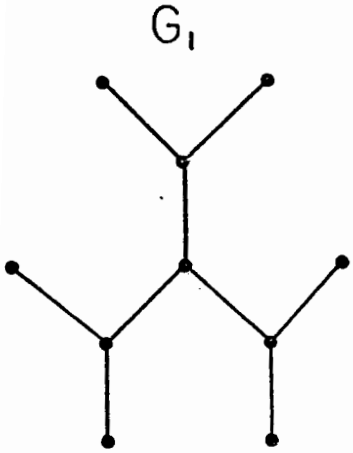

Fig. 3(a)

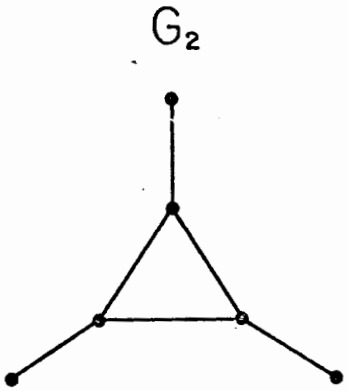

Fig. 3(b)

Fig. 3(c)

It can be easily verified that three pursuers suffice for each one of these graphs. It can also be shown (by techniques similar to the one we used in Section 3 for showing $s(K_n) = n$) that none of these graphs can be cleared by two pursuers. Furthermore, we claim that these graphs are minimal in the following sense. Recall that our graphs are embedded in a euclidean space and that a subgraph is a subset of a graph which itself constitutes a graph. Thus, the graph shown in Figure 4, for example, is a subgraph of graph G_2 (shown in Figure 3(b)) according to our definition.

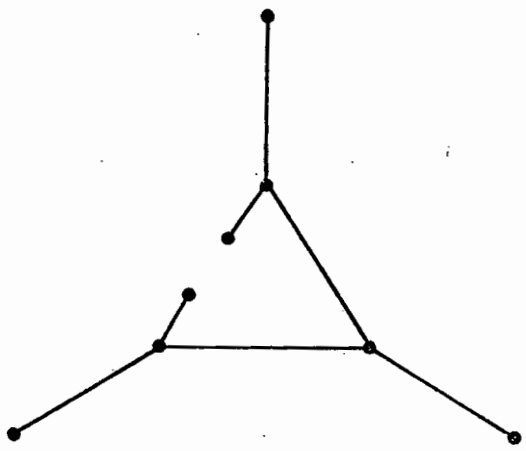

Fig. 4

We claim that G_i (i=1,2,3) does not contain a subgraph which is not homeomorphic to G_i and whose search number is 3. This fact can be easily verified.

It is interesting to note that G_1, G_2 and G_3 are the only minimal graphs with search number 3.

This is essentially the claim of the following theorem.

Theorem. A graph G has $s(G) \leq 2$ if and only if G does not have a subgraph homeomorphic to one of the graphs G_i (i=1,2,3).

Proof: In view of our previous discussion in this section, we have to prove only the "if" part. Thus, assume that G is a graph that does not contain a homeomorph of G_i (i=1,2,3). Let V^* denote the set of all vertices of degree greater than 2 in G. Define a graph $G^* = (V^*, E^*)$ (possibly with multiple edges and loops) as follows. For every pair $u, v \in V^*$ (possibly u=v) and for each path $u = w_0, \ldots, w_{p+1} = v$ in G, such that the w_j's (j=1,...,p) have degree 2 in G, let there be an edge linking u and v in G^*, corresponding (in a one-to-one fashion), to this path in G. Since G is connected, it is clear that G^* is also connected. Since G does not contain a homeomorph of G_2, G^* does not contain a cycle of length greater than 2. Since G does not contain a homeomorph of G_3, there can be no more than two edges in G^* linking the same pair of distinct vertices. Furthermore, since G does not contain a homeomorph of G_1, each $v \in V^*$ has no more

than two neighbors in G^*. It follows that G^* must contain a simple path such that each edge of G^* which is not on this path is either a loop or linking two consecutive vertices on this path.

All these properties of G^* imply the following valid search plan for G with two pursuers. Let $v_1, v_2, \ldots, v_p$ be the order in which the vertices of G^* appear on the path mentioned above. Note that these are also vertices of G. The two pursuers start at v_1. One of them waits at v_1, while the other one clears all the edges that are reachable from v_1 without passing through the edges that lead towards v_2. Then both pursuers proceed from v_1 to v_2, either along different paths (there can be two different paths at most) or along the same path, if there is only one path from v_1 to v_2. Next, one waits at v_2 while the other is clearing all unclear edges, that are reachable from v_2 without passing through the edges that lead towards v_3. Then both proceed to v_3, and so on. □

A typical graph with search number 2 is shown in Figure 5.

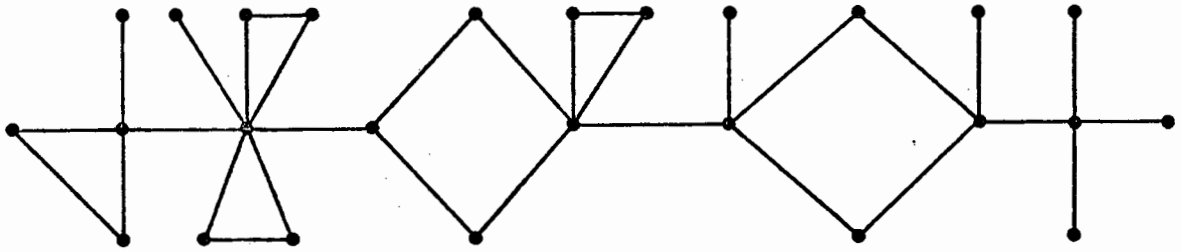

Fig. 5

Corollary. For a graph G, $s(G) \leq 2$ if and only if (i) There is a simple path covering all vertices of G of degree greater than 2. (ii) No simple cycle in G contains more than two vertices of degree greater than 2. (iii) No two simple cycles in G have a common edge.

Remark. Our theorem on graphs with $s(G) \leq 2$ implies an efficient algorithm for deciding whether or not $s(G) \leq 2$. First, construct the graph G^* and check whether or not there is a vertex of reduced degree (i.e. the number of neighbors) greater than 2 in G^*, and whether or not there are two distinct vertices in G^* linked by more than two edges. If the answer to either one of these questions is "yes," then $s(G) \geq 3$. If the answer to both questions is "no," then a Hamiltonian path in G^* can be easily found. Now, check whether or not there is an edge of G^*, not on the Hamiltonian path, that is neither a loop nor linking two consecutive vertices on the path. Again, if the answer is "yes" then $s(G) \geq 3$, and if "no" then $s(G) \leq 2$. (Note that the case $s(G) = 1$ is trivial, namely, G must be a simple path.

5. Graphs with search number 3

In section 4 we proved that there were only three minimal graphs which could not be cleared by two pursuers. The situation with three pursuers is much more complicated. We have been able

to identify thirteen minimal graphs (see Figure 6) which cannot be cleared by three pursuers.

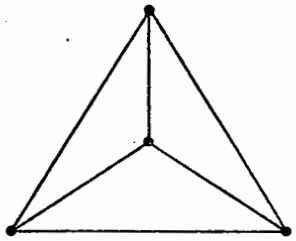

Fig. 6(a)

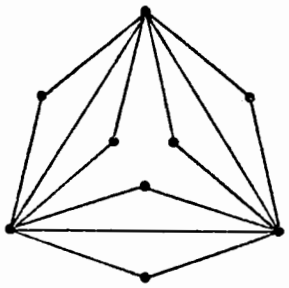

Fig. 6(b)

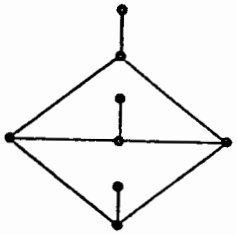

Fig. 6(c)

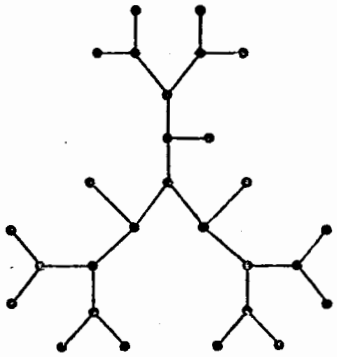

Fig. 6(d)

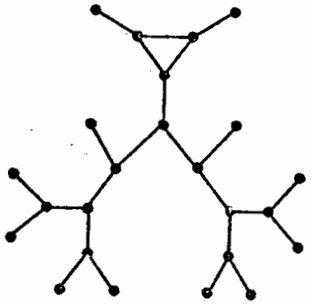

Fig. 6(e)

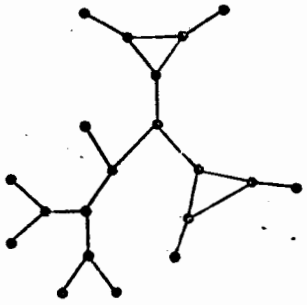

Fig. 6(f)

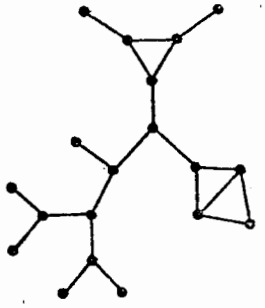

Fig. 6(g)

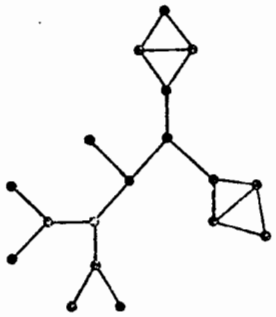

Fig. 6(h)

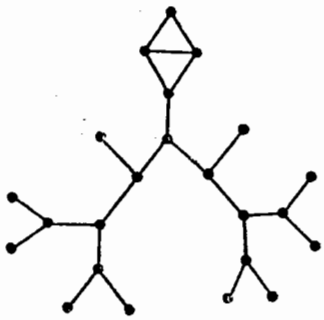

Fig. 6(i)

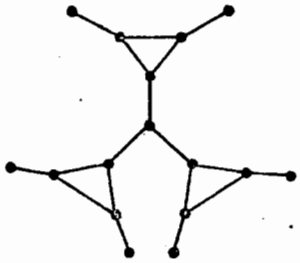

Fig. 6(j)

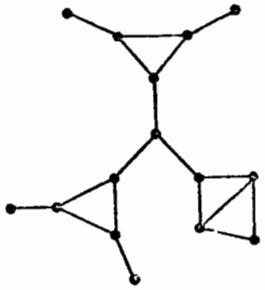

Fig. 6(k)

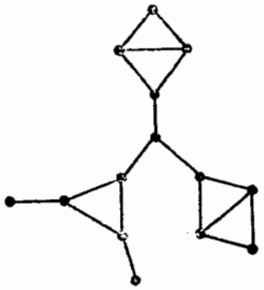

Fig. 6(l)

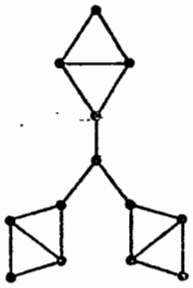

Fig. 6(m)

Thus, a characterization of graphs whose search number is 3, based on these minimal graphs, although seems to be possible, may not be particularly enlightening.

Appendix

Number of pursuers required to clear a graph in a predetermined order

Given a subset F of edges in $G = (V,F)$, let $V(F)$ denote the set of all vertices v such that at least one edge in F and at least one edge in $E\backslash F$ are incident upon v. Obviously, if at some time t, F is the set of all clear edges, then at that time guards must be placed at each $v \in V(F)$. Suppose that we have to clear one more edge $e \in E\backslash F$ while keeping all edges in F clear. We distinguish the following cases:

Case 1: The edge e has one of its vertices v in $V(F)$ and e is the only edge in $E\backslash F$ that is incident upon v. In this case the guard placed at v should proceed into e, clear it up, and occupy the other vertex of e.

Case 2: One vertex v of e is either of degree one or v belongs to $V(F)$. We also assume in this case that if $v \in V(F)$ then another edge $e' \notin E\backslash F$ is incident upon v. In this case one additional pursuer should be employed for clearing the edge e, starting from v and then occupying the other vertex of e.

Case 3: Both vertices of e have degree greater than one and neither belongs to $V(F)$. In this case we need two additional pursuers for clearing e and then occupying both of its vertices.

The above case analysis implies that the number k of pursuers required for clearing the edges in the order $e_1,\ldots,e_m$ can be calculated as follows. Initialize with $F = \emptyset$ and one pursuer in "reserve." Note that since $V(\emptyset) = \emptyset$ no pursuer is required to occupy any vertex at the beginning. Next, suppose that $F = \{e_1,\ldots,e_{j-1}\}$ and every vertex in V(F) is occupied by a pursuer and that there are r pursuers in "reserve." Let x be the number of additional searchers required for clearing e_j according to the previous case analysis ($0 \leq x \leq 2$).

Update: $r = \text{Max}\{r,x\} + |V(F)| - |V(F \cup \{e_j\}|$

$F = (F \cup \{e_j\})$.

Thus, after clearing e_j, we have r pursuers in reserve and $|V(F)|$ pursuers occupying critical vertices. This algorithm terminates when $F = E$ and then, since $V(E) = \emptyset$, the number r equals the minimum of pursuers required for clearing in the order $e_1,\ldots,e_m$.

References

1. S. Alpern, "The Search Game with Mobile Hider on the Circle," in Differential Games and Control Theory, E. O. Roxin, P. T. Liu and R. L. Sternberg, eds., Marcel Dekker, Inc., New York, 1974, pp. 181-200.

2. I. M. Asel'derova, "On a Certain Discrete Pursuit Game on Graphs," Cybernetics, 10 (1974), pp. 859-863.

3. R. Isaacs, Differential Games, John Wiley & Sons, New York, 1965.

4. T. D. Parsons, "Pursuit-Evasion in a Graph," in Theory and Applications of Graphs, Y. Alavi and P. R. Lick, eds., Springer-Verlag, 1978, pp. 426-441.

4. T. D. Parsons, "The Search Number of a Connected Graph," Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing at Boca-Raton, January-February 1978 (to appear).

4. Graphs with search number 2

Consider the three graphs shown in Figure 3:

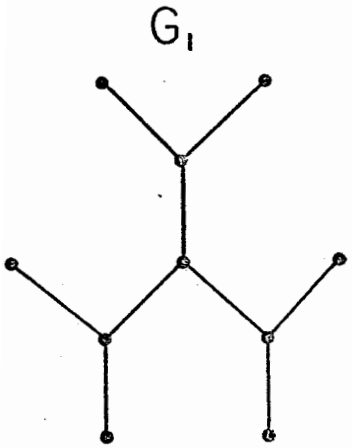

Fig. 3(a)

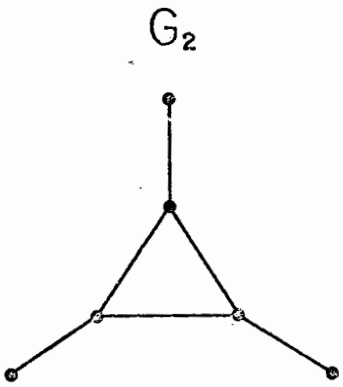

Fig. 3(b)

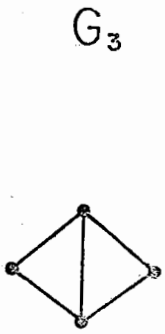

Fig. 3(c)

It can be easily verified that three pursuers suffice for each one of these graphs. It can also be shown (by techniques similar to the one we used in Section 3 for showing $s(K_n) = n$) that none of these graphs can be cleared by two pursuers. Furthermore, we claim that these graphs are minimal in the following sense. Recall that our graphs are embedded in a euclidean space and that a subgraph is a subset of a graph which itself constitutes a graph. Thus, the graph shown in Figure 4, for example, is a subgraph of graph G_2 (shown in Figure 3(b)) according to our definition.

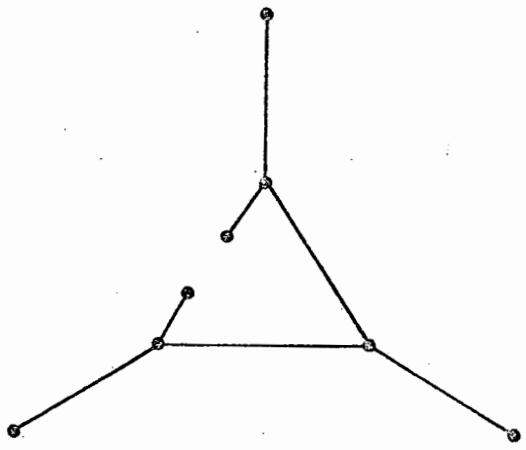

Fig. 4

to identify thirteen minimal graphs (see Figure 6) which cannot be cleared by three pursuers.

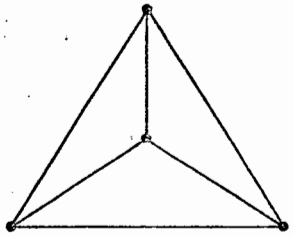

Fig. 6(a)

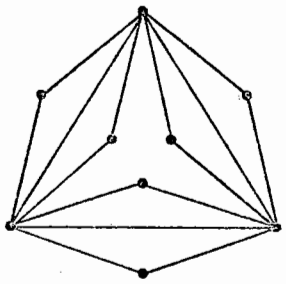

Fig. 6(b)

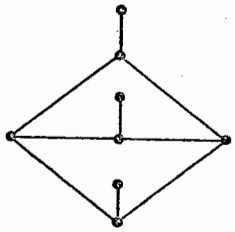

Fig. 6(c)

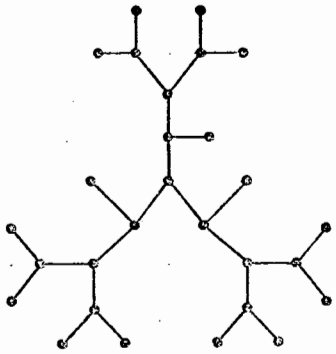

Fig. 6(d)

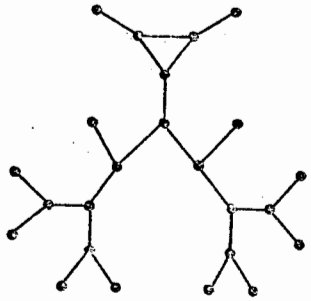

Fig. 6(e)

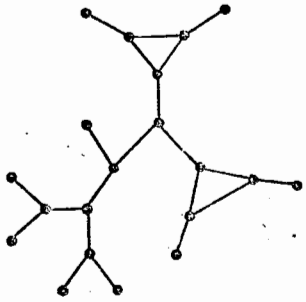

Fig. 6(f)

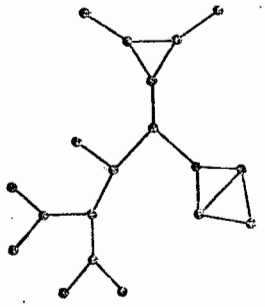

Fig. 6(g)

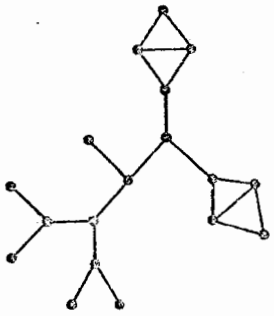

Fig. 6(h)

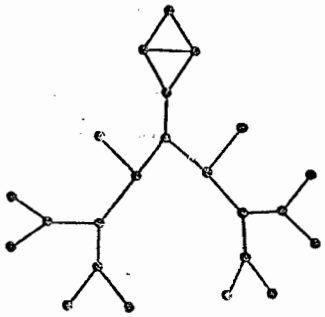

Fig. 6(i)

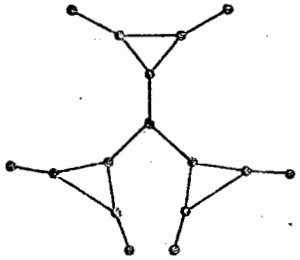

Fig. 6(j)

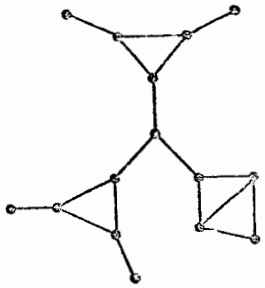

Fig. 6(k)

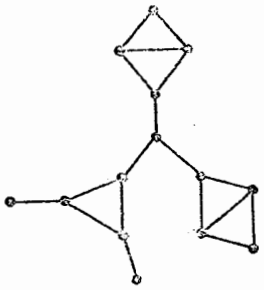

Fig. 6(l)

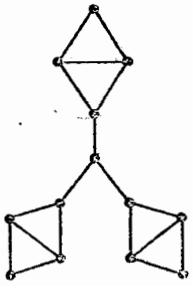

Fig. 6(m)

Thus, a characterization of graphs whose search number is 3, based on these minimal graphs, although seems to be possible, may not be particularly enlightening.

than two neighbors in G^*. It follows that G^* must contain a simple path such that each edge of G^* which is not on this path is either a loop or linking two consecutive vertices on this path.

All these properties of G^* imply the following valid search plan for G with two pursuers. Let $v_1, v_2, \ldots, v_p$ be the order in which the vertices of G^* appear on the path mentioned above. Note that these are also vertices of G. The two pursuers start at v_1. One of them waits at v_1, while the other one clears all the edges that are reachable from v_1 without passing through the edges that lead towards v_2. Then both pursuers proceed from v_1 to v_2, either along different paths (there can be two different paths at most) or along the same path, if there is only one path from v_1 to v_2. Next, one waits at v_2 while the other is clearing all unclear edges, that are reachable from v_2 without passing through the edges that lead towards v_3. Then both proceed to v_3, and so on. □

A typical graph with search number 2 is shown in Figure 5.

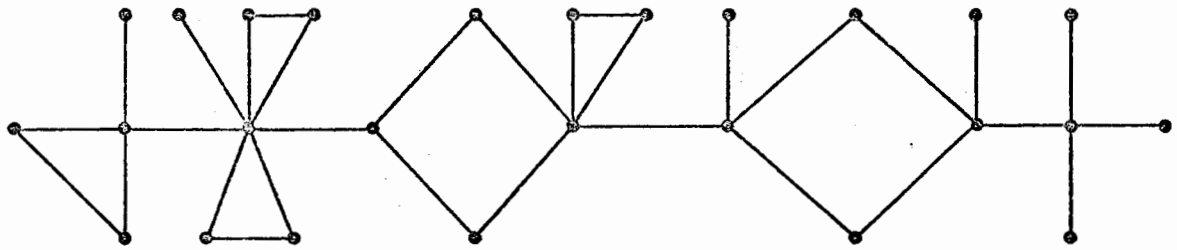

Fig. 5

In Section 2 we describe a linear time algorithm for finding the minimum number of pursuers (the search number) required for a tree graph. The algorithm computes a search plan as well. In Section 3 we describe a method to compute the search number of a general graph. This method is then applied to derive some bounds for the search number. In Section 4 we characterize graphs with search number 2 (via minimal graphs of search number 3). Section 5 provides examples of minimal graphs with search number 4.

2. Pursuit-evasion on trees

In this section our graph is a tree (i.e. it is connected and has no cycles) and is denoted by T. The number of pursuers required to guarantee capture is denoted by s(T) and is called the "search number" of T. We develop here a linear-time algorithm for finding s(T) as well as a supporting search plan.

There does not seem to exist a simple rule that states how essentially should a tree be searched. The tree shown in Figure 1 may be a good example.

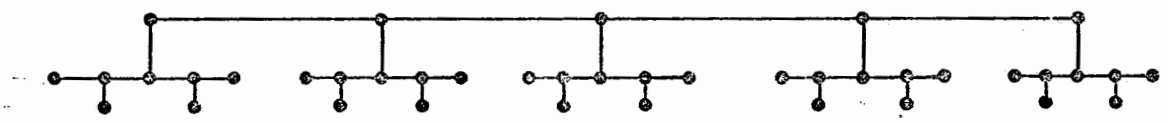

Fig. 1