Discrete Mathematics 314 (2014) 1–5

Contents lists available at ScienceDirect

Discrete Mathematics

journal homepage: www.elsevier.com/locate/disc

An evasion game on a graph

John Haslegrave

School of Mathematics and Statistics, University of Sheffield, Hounsfield Road, Sheffield, S3 7RH, UK

ARTICLE INFO

Article history:
Received 9 December 2011
Received in revised form 2 September 2013
Accepted 4 September 2013
Available online 8 October 2013

Keywords:
Pursuit game
Evasion game
Tree
Cops and robbers
Graph searching

ABSTRACT

This paper introduces a pursuit and evasion game to be played on a connected graph. One player moves invisibly around the graph, and the other player must guess his position. At each time step the second player guesses a vertex, winning if it is the current location of the first player; if not the first player must move along an edge. It is shown that the graphs on which the second player can guarantee to win are precisely the trees that do not contain a particular forbidden subgraph, and best possible capture times on such graphs are obtained.

1. Introduction

Pursuit and evasion games on graphs have been widely studied. Perhaps the most significant is the Cops and Robbers game, an instance of which is a graph G together with a fixed number of cops. The cops take up positions on vertices of G and a robber then starts on any unoccupied vertex. The cops and the robber take turns: the robber chooses either to remain at his current vertex or to move to any adjacent vertex, and then the cops simultaneously make moves of the same form. The game is played with perfect information, so that at any time each of the players knows the location of all others. The cops win if at any point one of them is at the same position as the robber. Early results on this game include those obtained by Nowakowski and Winkler [5] and Aigner and Fromme [1]. An important open problem is Meyniel's conjecture, published by Frankl [4], that $O(\sqrt{n})$ cops are enough to win on any n-vertex connected graph. More recently, several variations on the game have been analysed by Clarke and Nowakowski (e.g. [3]).

In this paper we will consider a novel form of pursuit game, which bears some similarity to the Cops and Robbers game but differs in that the movement of the pursuer (the cat) is not constrained by the edges of the graph, and also in that the pursuer is disadvantaged by not knowing where the pursued (the mouse) is. This imperfect information will naturally lead to a different emphasis: we ask whether there is a strategy for the cat that is successful against any possible strategy for the mouse, and if so how long it takes. Another recent variation of a similar type is the Robber Locating game, introduced by Seager [6] and further studied by Carraher, Choi, Delcourt, Erickson, and West [2], in which a cop probes a vertex at each turn and is told the current distance to the robber.

Descriptively, we consider a connected graph to represent a network of mouse-holes connected by passageways. The cat tries to catch the mouse by inserting a paw into one of the holes; if the cat has chosen the correct hole, then the mouse is caught. After each unsuccessful attempt, the mouse moves from the hole he is currently in to any adjacent hole. A rudimentary form of this problem, asking how the cat may win on a path, appeared on an internet puzzle forum [7].

We consider the active version, in which the mouse is required to move. The cat cannot guarantee to win without this restriction (on at least two vertices), since the mouse would have at least two options at each time step, one of which

E-mail addresses: j.haslegrave@shef.ac.uk, j.haslegrave@cantab.net.

0012-365X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.disc.2013.09.004

avoids the cat. The active game is not feasible if there is only one vertex; clearly the cat can guarantee to catch the mouse in two attempts (by choosing the same vertex twice) on the two-vertex connected graph, and cannot do better, so we shall subsequently assume that our graph has at least three vertices.

2. Strategies for the mouse

We say that the mouse can survive to time t on a graph G if, for any sequence $c_1, \dots, c_t$ of vertices of G, there exists a sequence $m_1, \dots, m_t$ of vertices such that $m_i \neq c_i$ for every i, and $m_i m_{i+1}$ is an edge of G for $1 \leq i \leq t-1$. We will refer to the sequence (c_i) as a *cat sequence* and (m_i) as a *mouse sequence* that beats it. For each G we wish to determine whether there is a t such that the mouse cannot survive to time t, and to determine the least such t if so. Write $m(G)$ for the least such t, if it exists, and $m(G) = \infty$ otherwise, so that the mouse can survive to time t on G if and only if $m(G) > t$. The main aim of this paper is to find a necessary and sufficient condition on G for $m(G)$ to be finite, and a simple formula for $m(G)$ if G is such a graph.

Consider first the case where G is a cycle. In this case the mouse may always survive, because at every stage he has a choice of two moves and at least one of them must be safe. Formally, given a cat sequence $c_1, \dots, c_t$ we may inductively find a mouse sequence that beats it: take a mouse sequence $m_1, \dots, m_{t-1}$ to beat $c_1, \dots, c_{t-1}$ and choose for m_t a neighbour of m_{t-1} that is not equal to c_t; this is possible since there are two neighbours to choose from.

Trivially, if H is a subgraph of G and the mouse can survive to time t on H then he can survive to time t on G by restricting himself to making moves on H. Consequently the mouse can always survive if G contains a cycle. If $m(G) < \infty$, then G must be a tree.

Next we shall show that there are some trees on which the mouse can always survive. Let T^* be the tree consisting of three paths of length 3 with one common endpoint, with the jth path having vertices u_j, v_j, w_j and x in that order.

Lemma 1. *The mouse can survive to time t on T^* for any t.*

Proof. Given a cat sequence $c_1, \dots, c_t$ on T^*, we shall construct a mouse sequence $m_1, \dots, m_t$ that beats it. The key idea is that at every odd time the mouse will be at x or at distance 2 from x, and it will be x whenever possible. We shall show that when the mouse is forced away from x he may choose one of the three arms of T^* to move down in such a way that he will be able to return to x once it is safe to go back there (though he may need to know the cat sequence arbitrarily far in advance in order to make the correct choice).

It suffices to prove the assertion for odd t, since when s is even we shall then have proved that the mouse can survive to time $s+1$, and consequently to time s. The argument does not depend on t being odd, but we construct even terms of the mouse sequence from the neighbouring odd terms, so this will avoid having a separate case for the final term.

Set $m_i = x$ for every odd i such that $c_i \neq x$. We must now choose a suitable value of m_i for every odd i with $c_i = x$. For each such i, we must have $m_i = u_j$, $m_i = v_j$, or $m_i = w_j$ for some j. We divide such i into maximal subsequences of consecutive odd terms of the cat sequence taking the value x. Within each such group we ensure that we consistently choose the same value of j, since the mouse must go down one arm and may not return to x for the duration of this group. For each i and k, with i odd, such that $c_i = c_{i+2} = \cdots = c_{i+2k} = x$ but $c_{i-2} \neq x$ (or $i = 1$) and $c_{i+2k+2} \neq x$ (or $i + 2k = t$) we choose $j \in \{1, 2, 3\}$ such that $c_{i-1} \neq w_j$ and $c_{i+2k+1} \neq w_j$. This is possible since at most two out of the three values are not permitted. Now set $m_i = m_{i+2} = \cdots = m_{i+2k} = v_j$. This choice ensures that the mouse can safely move from x to v_j and back again when required.

We have now defined m_i for all odd i. If i is even and $m_{i-1} = m_{i+1} = x$, then choose any j with $c_i \neq w_j$ and set $m_i = w_j$. If i is even and $m_{i-1} = m_{i+1} = v_j$ for some j, then set $m_i = u_j$ or $m_i = w_j$, whichever is not equal to c_i. By our construction of m_i for odd i, the only other possibility for even i is that one of m_{i-1} and m_{i+1} is x but the other is v_j for some j; in that case either $c_{i-1} = x \neq c_{i+1}$ or $c_{i+1} = x \neq c_{i-1}$. Our choice of odd mouse values then implies that $c_i \neq w_j$ and we can take $m_i = w_j$. In every case we have chosen m_i for even i to be adjacent to m_{i-1} and m_{i+1}, and $m_i \neq c_i$ for all i, as required. □

In fact T^* is essentially the only example of a tree on which the mouse can survive: we shall show that the cat can always catch the mouse on any tree that does not have T^* as a subgraph. We refer to such trees as *T^*-free*.

Before giving a strategy for the cat to win on any T^*-free tree T with at least three vertices, we prove a lower bound on $m(T)$. We will show later that this bound is equal to $m(T)$ when T is T^*-free. The key idea in defining the lower bound is to consider how often the cat must visit each vertex.

Let T be a tree with at least three vertices, and let v be a vertex of T. Define $\tilde{d}(v)$ as the number of neighbours of v that are not leaves. Define $a(v)$ for $v \in V(T)$ as follows:

$$a(v) = \begin{cases} 2\tilde{d}(v) - 2 & \text{if } \tilde{d}(v) \geq 2; \\ 2 & \text{if } d(v) \geq 2 \text{ but } \tilde{d}(v) < 2; \\ 0 & \text{if } d(v) = \tilde{d}(v) = 1. \end{cases}$$

Let $A(T) = \sum_v a(v)$. If $d(v) = 1$ but $\tilde{d}(v) = 0$, then T is the two-vertex tree, which we have already excluded.

Lemma 2. *If T is a tree with at least three vertices, then $m(T) \geq A(T)$.*