Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

The fast search number of a Cartesian product of graphs

Yuan Xue, Boting Yang *

Department of Computer Science, University of Regina, Canada

ARTICLE INFO

Article history:
Received 11 April 2016
Received in revised form 30 October 2016
Accepted 8 March 2017
Available online xxxx

Keywords:
Graph searching
Fast searching
Cops and robber game

ABSTRACT

Given a graph that contains an invisible fugitive, the fast searching problem is to find the fast search number, i.e., the minimum number of searchers to capture the fugitive in the fast search model. In this paper, we give a new lower bound on the fast search number. Using the new lower bound, we prove an explicit formula for the fast search number of the Cartesian product of an Eulerian graph and a path. We also give formulas for the fast search number of variants of the Cartesian product. We present an upper bound on the fast search number of hypercubes, and extend the results to a broader class of graphs including toroidal grids.

1. Introduction

Motivated by applied problems in the real world and theoretical issues in computer science and mathematics, graph searching has become a hot topic. It has many models, such as edge searching, node searching, mixed searching, fast searching, etc. These models are basically defined by the class of graphs, the actions of searchers and fugitives, visibility of fugitives, and conditions of captures [1,2,4,5,9,10].

Parsons [14] first introduced the graph search problem in which both searchers and fugitive move continuously along edges of a graph. Megiddo et al. [13] studied the discrete version of Parsons' model, called edge search problem. In the edge search model, there are three actions for searchers: placing a searcher on a vertex, removing a searcher from a vertex and sliding a searcher along an edge from one endpoint to the other. An edge is cleared by a sliding action. Kirousis and Papadimitriou [11] introduced the node search problem, in which there are two actions for searchers: placing and removing. An edge becomes cleared if both endpoints are occupied by searchers. Bienstock and Seymour [3] introduced the mixed search problem, which is a combination of the edge searching and node searching. In the mixed search problem, searchers have the same actions as those in the edge search model, and an edge is cleared if both of its endpoints are occupied by searchers or cleared by a sliding action.

Dyer et al. [8] introduced the fast search problem. They proposed a linear time algorithm for computing the fast search number of trees. In the fast search model, there are two actions for searchers: placing and sliding. An edge is cleared by a sliding action and every edge is traversed exactly once. Details of this model will be given in Section 2. Stanley and Yang [15] gave a linear time algorithm for computing the fast search number of Halin graphs and their extensions. They also presented a quadratic time algorithm for computing the fast search number of cubic graphs, while the problem of finding the node search number of cubic graphs is NP-complete [12]. Yang [17] proved that the problem of finding the fast search number of a graph is NP-complete; and it remains NP-complete for Eulerian graphs. He also proved that the problem of determining whether the fast search number of G is a half of the number of odd vertices in G is NP-complete; and it remains NP-complete for planar graphs with maximum degree 4. Dereniowski et al. [7] characterized graphs for which 2 or 3 searchers are sufficient in the

* Corresponding author.
E-mail addresses: xue228@uregina.ca (Y. Xue), boting.yang@uregina.ca (B. Yang).

http://dx.doi.org/10.1016/j.dam.2017.03.003
0166-218X/© 2017 Elsevier B.V. All rights reserved.

fast search model. They proved that the fast searching problem is NP-hard for multigraphs and for graphs. Very recently, Xue et al. [16] provided lower bounds and upper bounds on the fast search number of complete k-partite graphs. They solved the open problem of determining the fast search number of complete bipartite graphs. They also presented upper and lower bounds on the fast search number of complete split graphs.

This paper is organized as follows. In Section 2, we give some definitions and notation. In Section 3, we first consider the trail covering problem, and show its relations to the fast searching problem. We then give a new lower bound on the fast search number. Using the new lower bound, we give an explicit formula for the fast search number of the Cartesian product of an Eulerian graph and a path. In Section 4, we investigate the fast search number of hypercubes. We prove an upper bound and a lower bound respectively on the fast search number of hypercubes. Section 5 concludes the paper with some open problems.

2. Preliminaries

Throughout this paper, we only consider finite undirected graphs with no loops or multiple edges. Let $G = (V, E)$ be a graph with vertex set V and edge set E. We also use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G respectively. We use uv to denote an edge with endpoints u and v. For a vertex $v \in V$, the *degree* of v is the number of edges incident on v, denoted $\deg_G(v)$. A *leaf* is a vertex that has degree one. A vertex is *odd* when its degree is odd. An *odd graph* is a graph with vertex degrees all odd. Similarly, a vertex is *even* when its degree is even; and an *even graph* is a graph with vertex degrees all even. Define $V_{\text{odd}}(G) = \{v \in V : v \text{ is odd}\}$.

For a subset $V' \subseteq V$, we use $G[V']$ to denote the subgraph induced by V', which consists of all vertices of V' and all the edges of G between vertices in V'. We use $G - V'$ to denote the induced subgraph $G[V \setminus V']$. For a subset $E' \subseteq E$, we use $G - E'$ to denote the subgraph $(V, E \setminus E')$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two subgraphs of G (the intersection of V_1 and V_2 may not be empty). The *union* of two graphs G_1 and G_2 is the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. We use $G_1 + V_2$ to denote the induced subgraph $G[V_1 \cup V_2]$ and we also use $G_1 + E_2$ to denote the subgraph $(V_1 \cup V(E_2), E_1 \cup E_2)$, where $V(E_2)$ is the vertex set of all edges in E_2.

Given two graphs H_1 and H_2, the *Cartesian product* of H_1 and H_2, denoted by $H_1 \Box H_2$, is the graph whose vertex set is the Cartesian product $V(H_1) \times V(H_2)$ and in which two vertices $(u, v), (u', v') \in V(H_1) \times V(H_2)$ are adjacent in $H_1 \Box H_2$ if and only if $u = u'$ and v is adjacent to v' in H_2, or $v = v'$ and u is adjacent to u' in H_1.

A *walk* is a sequence $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges such that each edge e_i, $1 \le i \le k$, has endpoints v_{i-1} and v_i. A *path* is a walk in which every vertex appears once, except that its first vertex might be the same as its last. We use $v_0 v_1 \ldots v_k$ to denote a path with ends v_0 and v_k. A *cycle* is a path in which its first vertex is the same as its last vertex. We use $v_0 v_1 \ldots v_k v_0$ to denote a cycle with $k+1$ vertices. We will also use P_n to denote a path with n vertices and C_n to denote a cycle with n vertices, respectively. A *trail* is a walk that does not contain the same edge twice. For a connected subgraph G' with at least one edge, an *Eulerian trail* of G' is a trail that traverses every edge of G' exactly once. A *circuit* is a trail that begins and ends on the same vertex. An *Eulerian circuit* of G' is an Eulerian trail of G' that begins and ends on the same vertex. A graph is called *Eulerian* if it contains an Eulerian circuit that traverses all its edges. We will use B_m to denote an Eulerian graph with m vertices. Note that we only consider finite graphs with no loops or multiple edges. So an Eulerian circuit or Eulerian subgraph contains at least three edges throughout this paper. A *trail cover* of a graph G is a family of edge-disjoint trails in G that contain every edge of G. The minimum number of such trails is called the *trail cover number* of G and is denoted by $\tau(G)$.

In the fast search model introduced by Dyer et al. [8], an invisible fugitive hides either on a vertex or along an edge, and he can move at a high speed at any moment from a vertex to another vertex along a path that contains no searchers. We call an edge uv *contaminated* if uv may contain the fugitive. An edge uv that does not contain the fugitive is called *cleared*. One of the two actions can happen in each step of the fast search model:

- *placing* a searcher on a vertex; or
- *sliding* a searcher along a contaminated edge from one endpoint to the other.

Note that the above sliding action is slightly different from the one used in the edge search model, in which a searcher is allowed to slide along a cleared edge. An edge uv can be cleared in one of the following two ways:

- if u is occupied by at least two searchers, one of them slides along uv from u to v; or
- if u is occupied by only one searcher and uv is the only contaminated edge incident on u, the searcher on u slides to v along uv.

In the fast search problem, we always suppose that all edges are contaminated initially and each edge will be cleared by a sliding action, that is, the fugitive is captured at the moment when the last contaminated edge is cleared; we also suppose that for each contaminated edge, after it is cleared, it will not get recontaminated in the remaining steps. Since searchers are allowed to slide only on contaminated edges, every edge will be traversed exactly once when all edges are cleared. A *fast search strategy* of a graph is a sequence of placing and sliding actions that clear all edges of the graph. Since searchers cannot be removed from the graph or "jump" from a vertex to another vertex, we can assume, without loss of generality, that all placing actions in a fast search strategy take place before all sliding actions. The *fast search number* of a graph G, denoted by $\text{fs}(G)$, is the smallest number of searchers needed to capture the fugitive in G. We say that a fast search strategy of G is