ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

Memory efficient algorithms for cactus graphs and block graphs

Boris Brimkov *, Illya V. Hicks

Computational & Applied Mathematics, Rice University, Houston, TX 77005, USA

ARTICLE INFO

Article history:
Received 17 March 2015
Received in revised form 14 August 2015
Accepted 26 October 2015
Available online xxxx

Keywords:
Constant-work-space algorithm
In-place algorithm
Shortest path
Cactus graph
Block graph
Chromatic polynomial

ABSTRACT

We present constant-work-space polynomial time algorithms for solving the shortest path problem, finding the chromatic polynomial, clique number, number of components, circumference, and girth of cactus graphs and block graphs. For some of these problems we also present in-place algorithms, which perform faster than the corresponding constant-work-space algorithms. The problems considered are fundamental to many areas of graph theory, computational geometry, and combinatorial optimization, and have a wide range of applications including transportation, robotics, and image processing.

1. Introduction

Space efficiency is a central aspect of algorithm design and has been investigated in the framework of many important problems. The constant-work-space computational model, also known in complexity theory as the log-space model, entails algorithms which use a read-only input array and $O(1)$ cells of working memory; the in-place model uses a read–write input array and $O(1)$ working memory. In this paper, we present constant-work-space and in-place algorithms for solving the shortest path problem in block graphs and cactus graphs. These algorithms improve the time complexity of our earlier work [9] while preserving space efficiency. We also present constant-work-space and in-place algorithms for finding the clique number, circumference, girth, number of components, and chromatic polynomials of cactus graphs and block graphs. As a rule, due to time–space trade-off, improved space-efficiency is achieved on the account of higher time-complexity.

Cactus and block graphs are well-studied families with numerous applications. Different types of cactus graphs are classified and enumerated in [7,19]; see also the bibliography therein for problems in the theory of condensation in statistical mechanics where such graphs arise in a natural way. Cactus graphs can also be used to model loop feeder telecommunications networks, material handling networks for automated guided vehicles, and the combination of local area networks formed by ring and bus structures [13]. They have also recently been used to compare, analyze, and visualize sets of related genomes [6]. Block graphs are of interest in convex geometry, as they are the only graphs for which connected subsets of vertices form a convex geometry [15] and where every set of vertices has a unique minimal connected superset whose closure is in the convex geometry. Block graphs are also used to decompose general graphs into the intersection graphs of their biconnected components.

There are a number of constant-work-space algorithms for the shortest path problem and other graph problems, typically over special types of graphs. Das et al. [14] give constant-work-space algorithms for finding shortest and longest paths and

* Corresponding author.
E-mail addresses: boris.brimkov@rice.edu (B. Brimkov), ivhicks@rice.edu (I.V. Hicks).

http://dx.doi.org/10.1016/j.dam.2015.10.032
0166-218X/

Please cite this article in press as: B. Brimkov, I.V. Hicks, Memory efficient algorithms for cactus graphs and block graphs, Discrete Applied Mathematics (2015), http://dx.doi.org/10.1016/j.dam.2015.10.032

matchings in k-trees. Munro and Ramirez [25] give a space efficient algorithm for shortest paths in complete level networks (a family of k-partite graphs). Asano et al. [4] give constant-work-space algorithms for shortest paths in trees and simple polygons; some ideas and techniques developed in [4] are adapted in the present paper. Jakoby et al. [22] give constant-work-space algorithms for shortest and longest paths in series–parallel graphs, by reducing the problem to one that is known to be solvable with $O(\log n)$ space and $O(n^c)$ time "for some constant c significantly larger than 1" [21]. In contrast, we focus on developing space-efficient algorithms which have a low order time complexity as well.

For general graphs, the shortest path problem can be solved in linear space by the Bellman–Ford algorithm. However, there is no known constant-work-space algorithm for this problem, and its existence is unlikely. In fact, the shortest path problem for general weighted graphs is NL-complete, so the existence of a constant-work-space algorithm would imply that NL = L [22]. The Floyd–Warshall algorithm [18] and its various enhancements (cf. [29]) are in-place shortest path algorithms for general graphs, but only yield the length of a desired shortest path and cannot print the path itself without additional (non-constant) space. Moreover, a graph is optimally stored (with respect to space) using adjacency lists, which is not a suitable representation for the Floyd–Warshall algorithm.

The shortest path problem has applications in transportation, robotics, facility layout, VLSI design, and autonomous guidance for pilotless aircraft [12]. The need to solve the shortest path problem also arises in image processing, for tasks like segmenting medical images [32,35] and extracting the shape of a cognitive map [16] as well as in numerous binary image processing problems (see [20,26,27]). Several geometric variants of the shortest path problem are studied in [31] and the Voronoi diagram is introduced as an efficient solution method.

Aside from graph theoretic problems, space efficient algorithms have also been designed for problems in computational geometry like finding convex hulls, Delaunay triangulations, and nearest neighbors [8,10,11], and for problems in image processing like connected components labeling, intensity image thresholding, and rotated image restoration [1]. Asano [33,2,3] describes additional applications of space-efficient algorithms to image processing. Space efficient algorithms are often advantageous in embedded systems where storage space is very limited. Consumer electronics like GPS receivers, printers, and DVD players frequently have a small and inflexible working memory which cannot be easily increased. Furthermore, the firmware for these devices is usually stored on flash memory cells, which are much faster at reading than writing. Thus, constant-work-space algorithms are suitable for this setting, since their input is a read-only array.

An important result under the constant-work-space computational model (which is referred to in the sequel) is Reingold's deterministic polynomial time algorithm [30] for the undirected st-connectivity problem (USTCON) of determining whether two vertices in an undirected graph belong to the same connected component. There are also a number of randomized algorithms for USTCON which perform faster than Reingold's algorithm but whose output may be incorrect with a certain small probability. For instance, Barnes and Feige [5,17] and Kosowski [23] provide fast randomized algorithms for USTCON with improved time–space trade-offs; Kosowski's algorithm, based on weighted Metropolis–Hastings walks due to Nonaka et al. [28], uses $O(\log n)$ space and $O(n^2 \log n)$ time.

In addition to the applications of our algorithms to the aforementioned problems in computational geometry and image processing, they can be of practical use in other fields; below we outline two possible real-world examples where they may be useful. Suppose an autonomous surveillance drone is programmed to circle around a group of targets, and move to a particular location on its route when signaled. The flight pattern of this drone can be modeled by a cactus graph, and when signaled, the drone will solve the shortest path problem using a constant-work-space algorithm since its available memory is limited. As a second application, block graphs may be used to model networks of social or business groups where new members may join only one group but can later create groups of their own. Finding the shortest path in such a network corresponds to establishing contact between two people through the fewest interpersonal connections; the clique number corresponds to the largest group in the network, and the chromatic polynomial corresponds to the number of ways to assign jobs so that there are no duplicating jobs within each group. Since business and social networks can be very large, it may be necessary to use memory efficient algorithms to solve these problems.

This paper is organized as follows. In the next section, we recall some basic definitions and introduce a few concepts which will be used later on. In Section 3, we present a constant-work-space algorithm for finding the shortest path in block graphs. In Section 4, we adapt this algorithm to cactus graphs. In Section 5, we propose in-place algorithms for cactus and block graphs, which perform faster than the corresponding constant-work-space algorithms. In Section 6, we use the subroutines described in the previous sections to find the clique number, girth, circumference, and chromatic polynomials of cactus graphs and block graphs using constant work space. We conclude with some final remarks in Section 7.

2. Preliminaries

A *constant-work-space algorithm* is an algorithm that uses only a constant number of memory cells, each with size $O(\log n)$, where n is the size of the input. In addition, the input is given as a read-only array and may not be modified. An *in-place algorithm* also uses a constant number of memory cells, each with size $O(\log n)$; its input is a read–write array, the content of which may be modified or deleted. The output of constant-work-space and in-place algorithms is write-only and does not count toward the space used. We will assume that all graphs in the present paper are given by adjacency lists, where the vertices of a graph of order n are labeled with the first n natural numbers; the jth neighbor of vertex i can be accessed in $O(1)$ time with Adj(i, j).

Please cite this article in press as: B. Brimkov, I.V. Hicks, Memory efficient algorithms for cactus graphs and block graphs, Discrete Applied Mathematics (2015), http://dx.doi.org/10.1016/j.dam.2015.10.032