Discrete Applied Mathematics 158 (2010) 1660–1667

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph

Ken Takata *

Department of Mathematics, Hamline University, 1536 Hewitt Ave., Box 180, St. Paul, MN, 55104, United States

ARTICLE INFO

Article history:
Received 6 August 2008
Received in revised form 1 October 2009
Accepted 19 May 2010
Available online 2 July 2010

Keywords:
Graph algorithms
Minimal vertex separator
Listing algorithms
Polynomial space
Backtracking

ABSTRACT

For a graph G in read-only memory on n vertices and m edges and a write-only output buffer, we give two algorithms using only $O(n)$ rewritable space. The first algorithm lists all minimal $a-b$ separators of G with a polynomial delay of $O(nm)$. The second lists all minimal vertex separators of G with a cumulative polynomial delay of $O(n^3m)$.

One consequence is that the algorithms can list the minimal $a-b$ separators (and minimal vertex separators) spending $O(nm)$ time (respectively, $O(n^3m)$ time) per object output.

1. Introduction

Finding all minimal $a-b$ separators – that is, minimal sets of vertices whose deletion from a graph disconnects vertices a and b – is a problem relevant to the study of listing algorithms [7,8,11], network flow, formal concept analysis [4], and the treewidth of a graph [5,6].

We describe a simple backtracking algorithm for listing all the minimal $a-b$ separators in a graph $G=(V,E)$ with n vertices and m edges. This graph, of size $n+m$, is given in read-only memory, and the output is sent to the write-only buffer.

By creating a backtracking tree of height at most n, the algorithm works with a polynomial delay of $O(nm)$ and uses $O(n)$ rewritable space, using at most $O(n\log n)$ bits to perform its computations. Since the space necessary to store a graph (either as an adjacency list or matrix) is at least $\Omega(n\log n)$ bits, this algorithm is space-optimal within a multiplicative constant. We also provide a related backtracking algorithm that can list all minimal vertex separators in a graph with cumulative polynomial delay of $O(n^3m)$ and again using $O(n)$ rewritable space. To achieve this space complexity, these algorithms avoid using a data structure to store previously output objects, and thus all objects that the algorithm outputs can be sent to a write-only memory buffer. This is important given that a graph may have an exponential number of minimal $a-b$ separators (e.g., consider a graph with $\Theta(n)$ edge-disjoint paths of length 3 between vertices a and b).

One consequence of the algorithms' time complexity is that both algorithms work in polynomial total time. That is to say, the amount of time spent per object output is polynomial in the input size. The first and second algorithms list the minimal $a-b$ separators (minimal vertex separators) spending at most $O(nm)$ (respectively, $O(n^3m)$) time per object output. We describe these standards for efficient listing algorithms and how they are related further in Section 2.1. (For more on cumulative polynomial delay, see [7, p. 8]; for polynomial delay, see [8].)

Both algorithms differ from results by Shen and Liang [10] and Berry et al. [2] in that their algorithms have superior total polynomial time complexity and spend at most $O(n^2)$ time per object, but have greater space complexity, using $\Omega(n|\mathcal{S}|)$ space (where $|\mathcal{S}|$ is the total number of minimal $a-b$ separators to be output).

* Tel.: +1 651 523 2876; fax: +1 651 523 2620.
E-mail address: ktakata01@hamline.edu.

0166-218X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2010.05.013

For a specific class of graphs related to formal concept analysis – namely cobipartite graphs – Berry and Sigayret [4] provide an algorithm to list all minimal vertex separators using polynomial space. (Berry et al. [3] state that this algorithm uses at most $O(n \cdot \min(m, \binom{n}{2} - m))$ space and spends at most $O(\min(m, \binom{n}{2} - m))$ time per separator where $n = |V|$ and $m = |E|$.) In comparison, the algorithms presented in this paper work on general graphs and use less space and more time.

Before presenting the technical details, we first describe the backtracking approach in very general terms. The algorithm makes use of the relationship between (1) minimal $a - b$ separators and (2) connected induced subgraphs that contain the vertex a (see Lemma 1 for details). In specific, the algorithm constructs a simple backtracking tree to list these subgraphs and then uses propositions and lemmas described in Sections 2 and 3 to prune barren subtrees from the backtracking tree (i.e., subtrees that contain no new objects to be output). In doing so, the algorithm combines two techniques: (1) an approach Tsukiyama et al. [12] used to list the minimal cutsets of a graph by growing a connected component around a vertex a and (2) a generalization of the concept of a separator close to a vertex, introduced by Kloks and Kratsch [9].

The rest of the paper gives the details as follows: a brief discussion of standards for efficient listing and relevant graph theoretical terms, propositions, and lemmas are presented in Sections 2 and 3. Section 4 describes how one can create an efficient backtracking tree for which we can determine whether a node in the tree contains new objects to output or not. Sections 5 and 6 present the pseudocode and proofs for the two algorithms. Further remarks are in the last section.

2. Preliminaries

2.1. Standards for listing algorithms

Because a graph may have an exponential number of minimal $a - b$ separators, there is no algorithm that can always list these in $O(N^k)$ time where N is the input size and k is a constant. Below we discuss standards for efficient listing in which the running time depends on the size of both the input and the output and describe how these standards are relevant to the algorithms in this paper. We also discuss standards for space complexity.

Definition 1. A listing algorithm is said to run with polynomial delay if it spends at most $O(N^k)$ time

(1) between the start of the program and either producing the first object to be output or halting with no output, and
(2) between outputting one object and either producing the next object or halting.

Our algorithm to list the minimal $a - b$ separators works with polynomial delay. Thus, with a delay of at most $O(nm)$ time, our algorithm is guaranteed to output a new object or to halt. A weaker standard of listing complexity is the following:

Definition 2. A listing algorithm runs in polynomial total time if it outputs at least one object and spends at most $O(N^k)$ time per object output or halts within $O(N^k)$ time with no output.

With such an algorithm, one may have to wait a superpolynomial amount of time to get the first or any other output. For an example, consider an algorithm that ultimately outputs 2^N objects in rapid succession, but does so after a time delay of 2^N. While such a delay indicates that the number of objects that the algorithm outputs eventually is superpolynomial, one cannot necessarily terminate the algorithm after running the algorithm for a superpolynomial time and outputting only a polynomial number of objects. Given this limitation, an intermediate standard was proposed in [7] that was weaker than polynomial delay but stronger than polynomial total time.

Definition 3. A listing algorithm runs with cumulative polynomial delay if it spends at most

(1) $O(N^k)$ time between the start of the program and halting with no output, or
(2) $(i + 1) \cdot O(N^k)$ time in outputting the first i objects.

Such an algorithm may have a superpolynomial delay, but this can occur only after a superpolynomial number of objects have been output. Unlike polynomial total time, one avoids the possibility of superpolynomial delays near the beginning of an algorithm's run. Our algorithm that lists all minimal vertex separators works with a cumulative polynomial delay of $O(n^3m)$ and illustrates how algorithms of this listing complexity can arise naturally when modifying algorithms that work with polynomial delay.

These standards are related as follows: *poly. delay* $\subseteq$ *cumulative poly. delay* $\subseteq$ *poly. total time*. We also note the following standard for space complexity:

Definition 4. A listing algorithm runs in polynomial space if it uses at most $O(N^k)$ space.

Our algorithms are polynomial space. In addition, they are also space-optimal in the sense that the rewritable memory that they require is no greater than a multiplicative constant of the amount of memory needed to store the input in read-only memory.