Information Processing Letters 117 (2017) 35–39

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Simple DFS on the complement of a graph and on partially complemented digraphs

Benson Joeris [a], Nathan Lindzey [b,*], Ross M. McConnell [c], Nissa Osheim [c]

[a] Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON N2L 3G1, Canada
[b] Department of Mathematics Colorado State University, Fort Collins, CO 80521, United States
[c] Department of Computer Science, Colorado State University, Fort Collins, CO 80521, United States

ARTICLE INFO

Article history:
Received 31 August 2015
Received in revised form 17 August 2016
Accepted 22 August 2016
Available online 30 August 2016
Communicated by R. Uehara

Keywords:
Depth-first search
Graph algorithms
Graph traversal
Partially complemented digraphs

ABSTRACT

A *complementation operation* on a vertex of a digraph changes all outgoing arcs into non-arcs, and outgoing non-arcs into arcs. Given a digraph G, a *partially complemented digraph* $\tilde{G}$ is a digraph obtained from G by performing a sequence of vertex complement operations on G. Dahlhaus et al. showed that, given an adjacency-list representation of $\tilde{G}$, depth-first search (DFS) on G can be performed in $O(n+\tilde{m})$ time, where n is the number of vertices and $\tilde{m}$ is the number of edges in $\tilde{G}$. This can be used for finding a depth-first spanning forest and the strongly connected components of the complement of G in time that is linear in the size of G, and Dahlhaus et al. give applications to finding the modular decomposition of an undirected graph that require that some adjacency lists be complemented and others not. To achieve this bound, their algorithm makes use of a somewhat complicated stack-like data structure to simulate the recursion stack, instead of implementing it directly as a recursive algorithm. We give a recursive $O(n+\tilde{m})$ algorithm that requires no such data-structures.

1. Introduction

A *complementation operation* on a vertex of a digraph changes all outgoing arcs into non-arcs, and outgoing non-arcs into arcs. A *partially complemented representation* $\tilde{G}$ of G is a digraph obtained from a sequence of vertex complement operations, starting with a digraph G, and marking the vertices that have been complemented. Let n denote the number of vertices and m denote the number of edges of G, and let $\tilde{m}$ denote the number of edges of $\tilde{G}$. In [2], it is shown how to run several algorithms on G, given $\tilde{G}$, in $O(n+\tilde{m})$ time. This can be sublinear in the size of G.

Their algorithm for DFS on partially complemented digraphs was notably more complicated than their algorithm for BFS despite the comparable simplicity of DFS and BFS in the usual context.

Their algorithm for DFS is not recursive and is complicated by the use of so-called *complement stacks*, a stack-like data structure used to simultaneously simulate the recursion stack and keep track of which undiscovered vertices will not be called from which vertices on the recursion stack. This raised the question of whether there exists a more natural recursive DFS algorithm for partially complemented digraphs. To this end, we give an elementary recursive $O(n+\tilde{m})$ algorithm for performing depth-first search on G, given $\tilde{G}$.

A notable special case is when every vertex is complemented, that is, $\tilde{G}=\overline{G}$ where $\overline{G}$ denotes the complement of G. Algorithms for performing DFS on G, given $\overline{G}$, have also been developed [6,7], the most efficient of which runs in $O(n+\overline{m})$ time where n and $\overline{m}$ is the number of vertices and number edges of $\overline{G}$ respectively. To achieve this bound,

* Corresponding author.
E-mail address: nlindzey@uwaterloo.ca (N. Lindzey).

http://dx.doi.org/10.1016/j.ipl.2016.08.006
0020-0190/

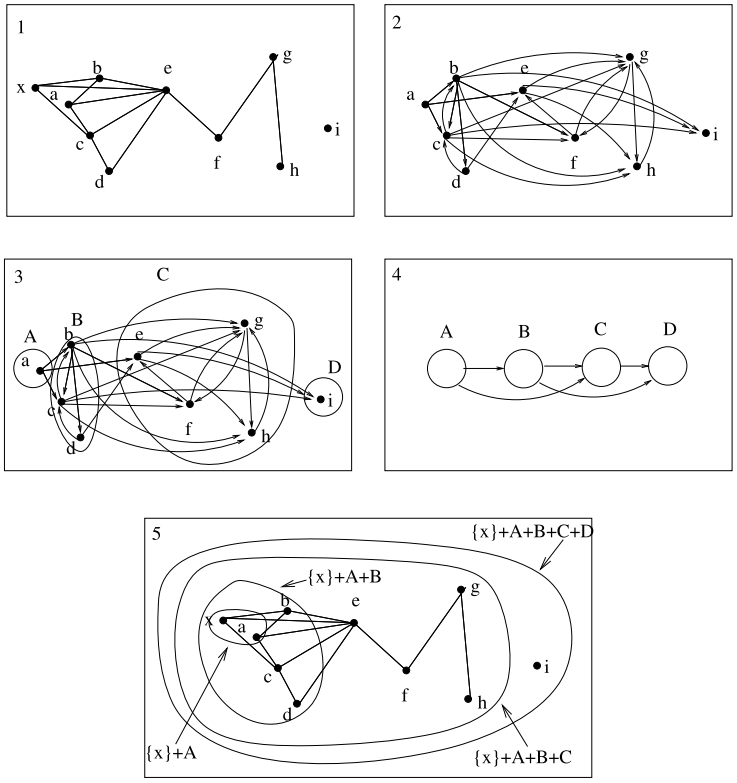

Fig. 1. Finding the modules that contain x in a graph where these modules are ordered by containment.

the algorithm in [6] makes use of the Gabow–Tarjan disjoint set data structure [4]. Our algorithm gives a more direct approach.

2. An application

A *module* of an undirected graph is a set Y of vertices such that every vertex not in Y is either a neighbor of all vertices in Y or a neighbor of none of them. Examples are $\{a, x\}$ and $\{a, b, c, d, x\}$ in the graph in part 1 of Fig. 1. There is a canonical recursive decomposition of every undirected graph, called the *modular decomposition*, which implicitly represents all modules of a graph. This decomposition was first discovered by Gallai [5], and has many applications to combinatorial algorithms on graphs. An $O(n^2)$ algorithm for computing it is given in [3]. A much more complicated $O(n + m)$ algorithm was later given in [8].

The algorithm of [3] chooses a vertex x, finds the set $\mathcal{M}$ of maximal modules that do not contain x, selects x and one vertex from each member of $\mathcal{M}$, forming an induced subgraph $G' = (V', E')$. It is not hard to show that all modules of G' of size greater than one contain x; they correspond to the ancestors of x in the modular decomposition tree. It then recursively finds the modular decomposition of the subgraph induced by M, for each $M \in \mathcal{M}$, to find the subtrees rooted at siblings of ancestors in the modular decomposition, completing the modular decomposition of G.

The only bottleneck to an $O(m \log n)$ bound for this simple approach is the problem of finding the modules of G' that properly contain x. Since they correspond to ancestors of x, we know that they are ordered by containment. Let $(Y_1, Y_2, \ldots, Y_k)$ be this ordering, that is, $\{Y_1, Y_2, \ldots, Y_k\}$ are the modules of G' that contain x, and for each $i \in \{1, 2, \ldots, k-1\}$, $Y_i \subset Y_{i+1}$. An example of such a G' is the one depicted in part 1 of Fig. 1, and $Y_1 = \{x, a\}$, $Y_2 = \{x, a, b, c, d\}$, $Y_3 = \{x, a, b, c, d, e, f, g, h\}$ and Y_4 is the set of vertices of G'. These are depicted in Part 5 of the figure.

To find these, the algorithm of [3] constructs a directed a graph $D(G') = (V' \setminus \{x\}, A)$, whose vertices are the vertices of G' other than x, and where there is a directed edge from vertex y to vertex z if and only if y is adjacent to exactly one of x and z in G'. Part 2 of Fig. 1 depicts $D(G')$ for the example graph G'. The following observation follows from the definition of $D(G')$: *A set Y of vertices of G' is a module if and only if $Y \setminus \{x\}$ has no incoming directed edge from $V' \setminus Y$ in $D(G')$.* A set Y is a module if and only if no vertex outside of Y fails to have a uniform relationship to members of Y, and this happens if and only if it has the same relationship to x as it does to all other members of Y, hence if and only if the condition of the observation applies.

From this observation, it is immediate that for each $i \in \{1, 2, \ldots, k-1\}$, $Y_{i+1} \setminus Y_i$ is a strongly connected component of $D(G')$. This is depicted in part 3 of the figure. Also, because these modules are ordered by containment, it is immediate from the observation that there must be a