Polynomially Improved Efficiency for Fast Parallel Single-Source Lexicographic Depth-First Search, Breadth-First Search, and Topological-First Search*

P. de la Torre1 and C. P. Kruskal2

1Department of Computer Science, University of New Hampshire, Durham, NH 03824, USA
dltrr@cs.unh.edu
2Department of Computer Science, University of Maryland, College Park, MD 20742, USA

Abstract. Although lexicographic (lex) variants of greedy algorithms are often \(\mathcal{P}\)-complete, \(\mathcal{NC}\)-algorithms are known for the following lex-search problems: lexicographic depth-first search (lex-dfs) for dags [12], [17], lexicographic breadth-first search (lex-bfs) for digraphs [12], [17], and lexicographic topological-first search (lex-tfs) for dags [12]. For the all-sources version of the problem for dense digraphs, the lex-dfs (lex-bfs, lex-tfs) in [12] is (within a log factor of) work-optimal with respect to the all-sources sequential solution that performs a dfs (bfs, tfs) from every vertex. By contrast, to solve the single-source lexicographic version on inputs of size \(n\), all known \(\mathcal{NC}\)-algorithms perform work that is at least an \(n\) factor away from the work performed by their sequential counterparts.

We present parallel algorithms that solve the single-source version of these lex-search problems in \(O(\log^2 n)\) time using \(M(n)\) processors on the EREW PRAM. \((M(n)\) denotes the number of processors required to multiply two \(n \times n\) integer matrices in \(O(\log n)\) time and has \(O(n^{2.376})\) as tightest currently known bound.) They all offer a polynomial improvement in work-efficiency over that of their corresponding best previously known and close the gap between the requirements of the best known parallel algorithms for the lex and the nonlex versions of the problems.

Key to the efficiency of these algorithms is the novel idea of a lex-splitting tree and lex-conquer subgraphs of a dag \(G\) from source \(s\). These structures provide a

* The research of Pilar de la Torre was supported in part by the NSF under Grants DC8600378, CCR-9010445, and CCR-9410592.
divide-and-conquer skeleton from which $\mathcal{NC}$-algorithms for several lexicographic search problems emerge, in particular, an algorithm that places in the class $\mathcal{NC}$ the $\text{lex-dfs}$ for reducible flow graphs—an interesting class of graphs which arise naturally in connection with code optimization and data flow analysis [4], [19]. A notable aspect of these algorithms is that they solve the lex-search problem instance at hand by efficiently transforming solutions of appropriate instances of (nonlex) path problems. This renders them potentially capable of transferring significant algorithmic advances—such as Driscoll et al.’s [14] single-source shortest paths algorithm and Ullman and Yannakakis’ [34] transitive closure algorithm—from fundamental (nonlex) path problems to lex-search problems.

1. Introduction

This paper considers several fundamental graph problems: Given a digraph $G$ and a single source vertex $s$, compute a depth-first search numbering, a breadth-first search numbering, and a topological sort numbering of $G$ from $s$. Optimal sequential solutions for these problems are known, but despite considerable research efforts (see, for example, [20], [22], [16], [1], [2], and [29]) the existence of $\mathcal{NC}$ work-efficient algorithms for these problems remains an open question.

Each of these problems can be solved by numbering the nodes in the order in which they are first visited by corresponding search algorithms, depth-first, breadth-first, and topological-first.

This paper addresses lexicographic (lex) variants of these search problems. The lex variant of a problem—such as dfs, bfs, or tfs—is to compute the output generated by a corresponding sequential algorithm, that selects the next arc to be examined from among the viable candidates in a greedy manner according to a given order on the arcs (usually specified by the adjacency list representation of the input graph). It has been shown (see, for example, [5], [6], and [18]) that lexicographic versions of these problems are often $\mathcal{P}$-complete. For example, $\text{lex-dfs}$ for general graphs was proved by Reif [27] to be $\mathcal{P}$-complete, and by Anderson [5] to remain so even for planar graphs.

In spite of their greedy character, $\mathcal{NC}$-algorithms have been discovered for the following lex-search problems: lexicographic depth-first search ($\text{lex-dfs}$) for dags [12], [17], lexicographic breadth-first search ($\text{lex-bfs}$) for digraphs [12], [17], and lexicographic topological-first search [12] for dags. The algorithms provided in [12] solve the all-sources version of these problem in $O(\log^2 n)$ time on an EREW PRAM for an $n$-vertex input graph. The $\text{lex-dfs}$ ($\text{lex-bfs}$, $\text{lex-tfs}$) algorithm on dense digraphs performs $O(n^3)$ work ($O(n^3 \log n)$ work), which is optimal (within a log $n$ factor) with respect to the all-sources sequential solution that performs a dfs (bfs, tfs) from every vertex. However, the $\text{lex-dfs}$ ($\text{lex-bfs}$, $\text{lex-tfs}$) in [12] and [17] ([12] and [17], [12]) solves the single-source version of the problem for dense digraphs performing $O(n^3)$ work ($O(n^3 \log n)$ work), which is at least an $n$ factor ($n \log n$ factor) away from optimal with respect to the sequential single-source algorithm. Given that $O(\log^2 n)$ time and $M(n)$ processors suffice to find a dfs-numbering [26], a bfs numbering [16], and some topological sort numbering [13], [15], [21] it is natural to ask ourselves: Are the lex variants under consideration “significantly harder” to solve efficiently?