Journal of Discrete Algorithms 8 (2010) 189–198

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A faster algorithm for the single source shortest path problem with few distinct positive lengths

James B. Orlin [a], Kamesh Madduri [b,1], K. Subramani [c,*,2], M. Williamson [c]

[a] Sloan School of Management, MIT, Cambridge, MA, USA
[b] Computational Research Division, Lawrence Berkeley Laboratory, Berkeley, CA, USA
[c] LDCSEE, West Virginia University, Morgantown, WV, USA

ARTICLE INFO

Article history:
Received 2 November 2008
Accepted 3 March 2009
Available online 20 March 2009

Keywords:
Shortest path problem
Dijkstra's algorithm
Linear time
Red–blue graphs

ABSTRACT

In this paper, we propose an efficient method for implementing Dijkstra's algorithm for the Single Source Shortest Path Problem (SSSPP) in a graph whose edges have positive length, and where there are few distinct edge lengths. The SSSPP is one of the most widely studied problems in theoretical computer science and operations research. On a graph with n vertices, m edges and K distinct edge lengths, our algorithm runs in $O(m)$ time if $nK \leqslant 2m$, and $O(m \log \frac{nK}{m})$ time, otherwise. We tested our algorithm against some of the fastest algorithms for SSSPP on graphs with arbitrary but positive lengths. Our experiments on graphs with few edge lengths confirmed our theoretical results, as the proposed algorithm consistently dominated the other SSSPP algorithms, which did not exploit the special structure of having few distinct edge lengths.

1. Introduction

In this paper, we provide an algorithm for solving the Single Source Shortest Path Problem (SSSPP) on a graph whose edges have positive length. The SSSPP is an extremely well-studied problem in both the operations research and the theoretical computer science communities because of its applicability in a wide range of domains. Ahuja et al. [2] describe a number of applications of the SSSPP, as well as efficient algorithms for the same. This paper provides an efficient algorithm for the SSSPP in the case where the number of distinct edge lengths is small. Our motivation for focusing on problems with few distinct edge lengths comes from a problem that arises in social networks (see Section 3).

We consider a graph with n vertices, m edges, and K distinct edge lengths. We provide two algorithms: The first algorithm is a simple implementation of Dijkstra's algorithm that runs in time $O(m + nK)$. The second algorithm modifies the first algorithm by using binary heaps to speed up the FindMin() operation. Assuming that $nK > 2m$, its running time is $O(m \log \frac{nK}{m})$.

For various ranges of the parameters n, m, and K, the running time of our algorithm is less than the running time of Fredman and Tarjan's Fibonacci Heap implementation [8], which runs in $O(m + n \log n)$ time. In fact, it improves upon the Atomic Heap implementation of Fredman and Willard [9], which runs in $O(m + \frac{n \log n}{\log \log n})$ time. (This latter paper relies on a slightly different model of computation than is normally assumed in papers on algorithms.) In particular, our algorithm

* Corresponding author.
E-mail addresses: jorlin@mit.edu (J.B. Orlin), kmadduri@lbl.gov (K. Madduri), ksmani@csee.wvu.edu (K. Subramani), mwilli65@mix.wvu.edu (M. Williamson).

[1] This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
[2] This research has been supported in part by the Air Force Office of Scientific Research under grant FA9550-06-1-0050 and in part by the National Science Foundation through Award CCF-0827397.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2009.03.001

runs in $O(m)$ time whenever $nK = O(m)$. We also note that even if all edge lengths are distinct, the running time of our algorithm is $O(m \log m)$, which is the same time as the binary heap implementation of Dijkstra's algorithm.

The main contributions of this paper are as follows.

(i) A new algorithm for the SSSPP problem that is parameterized by the number of distinct edge lengths.
(ii) An empirical analysis of our algorithm that demonstrates the superiority of our approach when the number of distinct edge lengths is small.

The rest of this paper is organized as follows. Section 2 formally specifies the problem under consideration. Section 3 describes the motivation for our work. Related work in the literature is discussed in Section 4. Section 5 describes and analyzes the $O(m + nK)$ implementation of Dijkstra's algorithm. Section 6 describes the $O(m \log \frac{nK}{m})$ implementation and also provides a proof of its running time. In Section 7, we provide an empirical analysis, confirming the improved performance of our algorithm on graphs with few distinct edge lengths. We offer brief conclusions in Section 8.

2. Statement of problem

We consider a directed graph $G = (V, E)$, with a vertex set V with n vertices, and an edge set E with m edges. For each vertex $v \in V$, we let $E(v)$ denote the set of edges directed out of v. Let $L = \{l_1, \ldots, l_K\}$ be the set of distinct nonnegative edge lengths given in increasing order. We assume that L is provided as part of the input and stored as an array. Each edge $(i, j) \in E$ has an edge length $c_{ij} \in L$.

Rather than store the edge length c_{ij} explicitly, we assume that associated with each edge (i, j) is an index t_{ij} such that $c_{ij} = l_{t_{ij}}$. We note that in practice, one can determine L and all of the indices in $O(m + K \log K)$ expected time; we first use perfect hashing to identify the K distinct edge lengths, and then sort the edge lengths.

There is a special vertex $s \in V$ called the *source*. We let $\delta(v)$ denote the length of the shortest path in G from vertex s to vertex v. If there is no path from s to v, then $\delta(v) = \infty$. The goal of the SSSPP is to identify a shortest path from vertex s to each other vertex that is reachable from vertex s.

3. Motivation

Our work was motivated by the "gossip" problem for social networks. Consider a social network, which is composed of clusters of participants. We model the intra-cluster distance by the value 1 and the inter-cluster distances by a real number l, where $l > 1$. The goal is then to determine the fastest manner in which gossip originating in a cluster can reach all the participants in the social network. This is a special case of SSSPP in which $K = 2$.

Although the motivating example has $K = 2$, we have extended our results to values of K that can grow with the input size.

Given that the SSSPP problem arises in so many domains, researchers have called for a "toolbox" [5,10] of different implementations that are efficient for different types of input. Possibly, the algorithm presented here would be appropriate for such a toolbox.

4. Related work

The literature on the SSSPP problem is large; interested readers are referred to Ahuja et al. [1]. In what follows, we briefly outline various paradigms in algorithmic advancement and provide a context for our work.

The first polynomial time algorithm for the SSSPP problem was devised by Dijkstra [7], the running time of the algorithm depends upon the data structure used to implement the priority queue, which is part of the algorithm. Since then, advances in algorithmic techniques have been along the following fronts:

(i) New design paradigms – Thorup provided a linear-time algorithm for the SSSPP when the graph edges are undirected [14]. He exploited the connection between the Minimum Spanning Tree of an undirected graph and the Single Source Shortest paths tree.
(ii) Data Structure improvements – Algorithms based on Dijkstra's approach perform a series of Extract-Min() and Decrease-Key() operations. Inasmuch as each vertex is extracted only once and each edge is processed at most once, the running time of any such algorithm can be represented as: $T(n, m) = n * \text{Extract-Min}() + m * \text{Decrease-Key}()$.
An optimized priority queue design balances the costs between the two operations; in the comparison based-model, the most efficient priority queue for this pair of operations is the Fibonacci Heap. Dijkstra's algorithm with Fibonacci Heaps has a running time of $O(m + n \log n)$. Other heaps proposed for this problem include the d-heap [3] and the R-heap [6].
(iii) Parameterization – In this approach, the design focuses on a certain parameter (or parameters) that may be small in magnitude for an interesting subset of problems. One such parameter is the largest edge length (say C); Ref. [2] describes how Dijkstra's approach can be made to run in $O(m + n\sqrt{\log C})$ time.