ARTICLE IN PRESS

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

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

Solving all-pairs shortest path by single-source computations: Theory and practice☆

Andrej Brodnik[a,b], Marko Grgurovič[a,*]

[a] Department of Information Science and Technology, University of Primorska, Glagoljaška 8, 6000 Koper, Slovenia
[b] Faculty of Computer and Information Science, University of Ljubljana, Tržaška cesta 25, 1000 Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 19 November 2015
Received in revised form 31 January 2017
Accepted 20 March 2017
Available online xxxx

Keywords:
All pairs shortest path
Single source shortest path

ABSTRACT

Given an arbitrary directed graph $G = (V, E)$ with non-negative edge lengths, we present an algorithm that computes all pairs shortest paths in time $\mathcal{O}(m^*n + m \lg n + nT_\psi(m^*, n))$, where m^* is the number of different edges contained in shortest paths and $T_\psi(m^*, n)$ is the running time of an algorithm ψ solving the single-source shortest path problem (SSSP). This is a substantial improvement over a trivial n times application of ψ that runs in $\mathcal{O}(nT_\psi(m, n))$. In our algorithm we use ψ as a black box and hence any improvement on ψ results also in improvement of our algorithm. A combination of our method, Johnson's reweighting technique and topological sorting results in an $\mathcal{O}(m^*n + m \lg n)$ all-pairs shortest path algorithm for directed acyclic graphs with arbitrary edge lengths. We also point out a connection between the complexity of a certain sorting problem defined on shortest paths and SSSP. Finally, we show how to improve the performance of the proposed algorithm in practice. We then empirically measure the running times of various all-pairs shortest path algorithms on randomly generated graph instances and obtain very promising results.

1. Introduction

Let $G = (V, E)$ denote a directed graph where E is the set of edges and V is the set of vertices of the graph and let $\ell(\cdot)$ be a function mapping each edge to its length. Without loss of generality, we assume G is strongly connected. To simplify notation, we define $m = |E|$ and $n = |V|$. Furthermore, we define $d(u, v)$ for two vertices $u, v \in V$ as the length of the shortest path from u to v. A classic problem in algorithmic graph theory is to find shortest paths. Two of the most common variants of the problem are the single-source shortest path (SSSP) problem and the all-pairs shortest path problem (APSP). In the SSSP variant, we are asked to find paths with the least total length from a fixed vertex $s \in V$ to every other vertex in the graph. Similarly, the APSP problem asks for the shortest path between every pair of vertices $u, v \in V$. A common simplification of the problem constrains the edge length function to be non-negative, i.e. $\ell : E \to \mathbb{R}^+$, which we assume throughout the rest of the paper, except where explicitly stated otherwise. Additionally, we define $\forall(u, v) \notin E : \ell(u, v) = \infty$.

It is obvious that the APSP problem can be solved by n calls to an SSSP algorithm. Let us denote the SSSP algorithm as ψ. We can quantify the asymptotic time bound of such an APSP algorithm as $\mathcal{O}(nT_\psi(m, n))$ and the asymptotic space bound

☆ A preliminary version of this work has been published in the Proceedings of 23rd International Symposium on Algorithms and Computation (ISAAC 2012).

* Corresponding author.
E-mail addresses: andrej.brodnik@upr.si (A. Brodnik), marko.grgurovic@famnit.upr.si (M. Grgurovič).

http://dx.doi.org/10.1016/j.dam.2017.03.008
0166-218X/

as $\mathcal{O}(S_\psi(m, n))$, where $T_\psi(m, n)$ is the time required by algorithm ψ and $S_\psi(m, n)$ is the space requirement of the same algorithm. We assume that the time and space bounds can be written as functions of m and n only, even though this is not necessarily the case in more "exotic" algorithms that depend on other parameters of G. Note, that if we are required to store the computed distance matrix, then we will need at least $\Theta(n^2)$ additional space. If we account for this, then the space bound becomes $\mathcal{O}(S_\psi(m, n) + n^2)$.

In this paper we are interested in the following problem: what is the best way to make use of an SSSP algorithm ψ when solving APSP? There exists some prior work on a very similar subject in the form of an algorithm named the Hidden Paths Algorithm [10]. The Hidden Paths Algorithm is essentially a modification of Dijkstra's algorithm [3] to make it more efficient when solving APSP. Solving the APSP problem by repeated calls to Dijkstra's algorithm requires $\mathcal{O}(mn + n^2 \lg n)$ time using Fibonacci heaps [5]. The Hidden Paths Algorithm then reduces the running time to $\mathcal{O}(m^*n+n^2 \lg n)$. The quantity m^* represents the number of edges $(u, v) \in E$ such that (u, v) is included in at least one shortest path. In the Hidden Paths Algorithm this is accomplished by modifying Dijkstra's algorithm, so that it essentially runs in parallel from all vertex sources in G, and then reusing the computations performed by other vertices. The idea is simple: we can delay the inclusion of an edge (u, v) as a candidate for forming shortest paths until vertex u has found (u, v) to be the shortest path to v. However, the speedup technique employed by the Hidden Paths Algorithm is only applicable to Dijkstra's algorithm, since it explicitly sorts the shortest path lists by path lengths, through the use of a priority queue. As a related algorithm, we also point out that a different measure denoted as $|UP|$, related to the number of so-called uniform paths, has also been exploited to yield faster algorithms [2], including an $\mathcal{O}(n^2)$ average-case algorithm for random complete graphs [11].

In Sections 3 and 4 we show that there is a method for solving APSP which produces the shortest path lists of individual vertices in sorted order according to the path lengths. The interesting part is that it can accomplish this without the use of priority queues of any form, and requires only an SSSP algorithm to be provided. This avoidance of priority queues permits us to state a time complexity relationship between a sorted variant of APSP and SSSP. Since it is very difficult to prove meaningful lower bounds for SSSP, we believe this connection might prove useful.

As a direct application of our approach, we show that an algorithm with a similar time bound to the Hidden Paths Algorithm can be obtained. Unlike the Hidden Paths Algorithm, the resulting method is general in that it works for any SSSP algorithm, effectively providing a speed-up for arbitrary SSSP algorithms. The proposed method, given an SSSP algorithm ψ, has an asymptotic worst-case running time of $\mathcal{O}(m^*n + m \lg n + nT_\psi(m^*, n))$ and space $\mathcal{O}(S_\psi(m, n) + n^2)$. We point out that the m^*n term is dominated by the $nT_\psi(m^*, n)$ term, but we feel that stating the complexity in this (redundant) form makes the result clearer to the reader. For the case of ψ being Dijkstra's algorithm, this is asymptotically equivalent to the Hidden Paths Algorithm. However, since the algorithm ψ is arbitrary, we show that the combination of our method, Johnson's reweighting technique [9] and topological sorting gives an $\mathcal{O}(m^*n + m \lg n)$ APSP algorithm for directed acyclic graphs with arbitrary edge lengths.

In Section 5 we detail optimizations that can be used to speed up the proposed algorithm in practice. In Section 6 we perform an experimental comparison between current algorithms for the all-pairs shortest path problem.

2. Preliminaries

As before, let $G = (V, E)$ denote a directed graph where E is the set of edges and V is the set of vertices of the graph. Throughout the paper and without loss of generality, we assume that we are not interested in paths beginning in v and returning to v. We have previously defined the edge length function $\ell(\cdot)$, which we now extend to the case of paths. Thus, for a path π, we write $\ell(\pi)$ to denote its length, which corresponds to the sum of the lengths of its edges.

Similar to the way shortest paths are discovered in Dijkstra's algorithm, we rank shortest paths in nondecreasing order of their lengths. Thus, we call a path π the kth shortest path if it is at position k in the length-sorted shortest path list. The list of paths is typically taken to be from a single source to variable target vertices. In contrast, we store paths from variable sources to a single target. By reversing the edge directions we obtain the same lists, but it is conceptually simpler to consider the modified case. Thus, the kth shortest path of vertex v actually represents the kth shortest incoming path into v. We will now prove a theorem on the structure of shortest paths, which is the cornerstone of the proposed algorithm.

Definition 2.1 (*Ordered Shortest Path List P_v*)**.** Let $P_v = (\pi_1, \pi_2, \ldots, \pi_{n-1})$ denote the shortest path list for the vertex $v \in V$. Then, let $P_{v,k}$ denote the kth element in the list P_v. The shortest path lists are ordered according to path lengths, thus we have $\forall i, j : 0 < i < j < n \Rightarrow \ell(\pi_i) \leq \ell(\pi_j)$.

Theorem 2.2. *To determine $P_{v,k}$ we only need to know the first k elements of each list P_u, such that $(u, v) \in E$.*

Proof. We assume that we have found the first k shortest paths for all neighbors of v, and are now looking for the kth shortest path into v, which we denote as π_k. There are two possibilities: either π_k is simply an edge (u, v), in which case we already have the relevant information, or it is the concatenation of some path π and an edge (u, v). The next step is to show that π is already contained in $P_{u,i}$ where $i \leq k$.

We will prove this by contradiction. Assume the contrary, that π is either not included in P_u, or is included at position $i > k$. This would imply the existence of some path π' for which $\ell(\pi') \leq \ell(\pi)$ and which is contained in P_u at position $i \leq k$. Then we could simply take π_k to be the concatenation of (u, v) and π', thereby obtaining a shorter path than the

Please cite this article in press as: A. Brodnik, M. Grgurovič, Solving all-pairs shortest path by single-source computations: Theory and practice, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.03.008