Theoretical Computer Science 520 (2014) 43–50

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Sharing information for the all pairs shortest path problem ☆

Tadao Takaoka

Department of Computer Science, University of Canterbury, Christchurch, New Zealand

ARTICLE INFO

Article history:
Received 17 July 2012
Received in revised form 2 June 2013
Accepted 3 September 2013
Communicated by G.F. Italiano

Keywords:
Information sharing
All pairs shortest path problem
Single sink shortest path problem
Priority queue
Nearly acyclic graph
Limited edge cost

ABSTRACT

We improve the time complexity of the all pairs shortest path (APSP) problem for special classes of directed graphs. One is a nearly acyclic directed graph and the other is a directed graph with limited edge costs. The common idea for speed-up is information sharing by n single source shortest path (SSSP) problems that are solved for APSP. We measure the degree of acyclicity by the size, r, of a given set of feedback vertices. If r is small, the given graph can be considered to be nearly acyclic. We consider this parameter, r, in addition to the traditional parameters of the number of vertices, n, and that of edges, m. In the first part we improve the existing time complexity of $O(mn+r^3)$ for the all pairs shortest path problem to $O(mn+rn\log n)$. This complexity is equal to or better than the previous one for all values of r under a reasonable assumption of $m \geqslant n$. In the second part, we deal with a directed graph with non-negative integer edge costs bounded by c. We show the all pairs shortest path (APSP) problem can be solved in $O(mn+n^2\log(c/n))$ time with the data structure of cascading bucket system. We use the traditional computational model such that comparison–addition operations on distance data and random access with $O(\log n)$ bits can be done in $O(1)$ time. Also the graph is not separated, meaning $m \geqslant n-1$.

1. Introduction

We consider the all pairs shortest path (APSP) problem for a directed graph with non-negative edge costs under the classical computational model of addition–comparison on distances and random accessibility by an $O(\log n)$ bit address. The complexity for this problem under the classical computational model is $O(mn+n^2\log n)$ by running the single source shortest path (SSSP) algorithm n times, as the SSSP problem can be solved in $O(m+n\log n)$ time by Dijkstra's algorithm with a priority queue such as a Fibonacci heap [7] or 2–3 heap [12].

We improve the second term of the time complexity of the APSP problem for special classes of directed graphs; nearly acyclic directed graph and directed graph with limited edge costs of non-negative integers. The common idea for those two problems is resource sharing, or information sharing. When we solve n SSSP's, we let them share some resource so that we can spend less time than we would need to solve n SSSP's independently.

We start from nearly acyclic graphs. There can be many definitions for near-acyclicity [10,11,13]. Here we define it by the size of the feedback vertex set, denoted by T. Every cycle goes through at least one vertex in T, meaning that removal of T from the graph cuts all cycles. If the size is small, we say the given graph is nearly acyclic. We traverse edges forward and backward. When we traverse backward, we can define the single sink problem; we compute shortest paths from all vertices to a specified vertex, called a sink. If we solve r single sink problems for all vertices in T as sinks with $|T|=r$, and share the result when we solve n single source problems, we can achieve $O(mn+rn\log n)$ time for the APSP problem, which is equal to or better than the existing complexity of $O(mn+r^3)$ [11] for all values of r when $m \geqslant n$. For the priority

☆ A preliminary version of this paper was presented at [15].

E-mail address: tad@cosc.canterbury.ac.nz.

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2013.09.006

queue we use a Fibonacci heap or 2–3 heap to choose minimum distance vertices one by one and modify the distances of other candidate vertices, which are to be included in the solution set in the future.

In the second part of the paper, we consider the all pairs shortest path (APSP) problem for a directed graph with non-negative edge costs bounded by a positive integer under the classical computational model.

Again we improve the second term of the time complexity of $O(mn + n^2 \log n)$. To deal with such a graph with edge costs bounded by c, Ahuja, Melhorn, Orlin, and Tarjan [1] invented the radix heap and achieved $O(m + n\sqrt{\log c})$ time for SSSP. Thorup [16] improved this complexity to $O(m + n \log\log c)$. If we apply these methods n times for the APSP problem, the time complexity becomes $O(mn + n^2\sqrt{\log c})$ and $O(mn + n^2 \log\log c)$ respectively. We first improve the time complexity for this problem to $O(mn + cn)$. When $c \leqslant m$, that is, for moderately large c, this complexity is $O(mn)$. We finally reduce this to $O(mn + n^2 \log(c/n))$.[1] For the priority queue we start from a simple bucket system with c buckets, which gives nc find-min operations for both SSSP and APSP problems. If edge costs are between 0 and c for SSSP, the tentative distances from which we choose the minimum distance for a vertex to be included into the solution set take values ranging over a band of length c. If $c < n$, we can reduce the time for delete-min by looking at the fixed range of values, as observed by Dial [5]. We observe that the same idea works for the APSP problem in a better way, as shown in [14]. To the author's knowledge, this simple idea is nowhere else in print.

For the more sophisticated priority queue we use a cascading bucket system of k levels (k will be specified later) to choose minimum distance vertices one by one and modify the distances of other candidate vertices. This data structure is essentially the same as in Denardo and Wegman [4] and in Ahuja et al. [1]. Our contribution is to show how to use the data structure, which was used for the SSSP problem, in the APSP problem without increasing the complexity by a factor of n. We mainly follow the style of Cherkassky, Goldberg and Radzik [3]. Taking an analogy from agriculture, the data structure is similar to a combine-harvester machine with k wheels. The first $k-1$ wheels work inside the machine for threshing crops and refining them to finer grains from wheel to wheel, whereas the last wheel goes the distance of $c(n-1)$ and harvests the crops. For SSSP and APSP, the amount of crops is $O(n)$ and $O(n^2)$ respectively.

It is still open whether the APSP problem can be solved in $O(mn)$ time. The best bounds for a general directed graph are $O(mn + n^2 \log\log n)$ with an extended computational model by Pettie [9] and $o(mn)$ for an unweighted undirected graph by Chan [2]. In [9], extended instructions are simulated by the conventional RAM model. Our complexity of $O(mn + n^2 \log(c/n))$ is better than existing $O(mn + n^2 \log\log n)$ and $O(mn + n^2 \log\log c)$ when $c = n(\log n)^{o(1)}$.

The rest of the paper is as follows: In Section 2, we define shortest path problems: single source, single sink and all pairs. In Section 3, we define the sweeping algorithm on an acyclic graph. In Section 4, we define the generalized single source shortest path problem, where the concept of source is distributed over the entire set of vertices. In Section 5, we define nearly acyclic graph with a set of feedback vertices, and show the APSP problem can be solved in $O(mn + rn \log n)$ time, where r is the size of the feedback vertex set. In Section 6, we introduce a simple data structure of the linear bucket system. In Section 7, we discuss some properties of the shortest path problem with limited edge costs. In Section 8, we introduce the concept of pseudo-edges and show that the bucket system works well for the APSP problem by determining shortest distances directly on pseudo-edges. In Section 9 we describe the k-level cascading bucket system, which is a generalization of the linear bucket system. In Section 10, we show that the hidden path algorithm [8] can fit into our framework of bucket systems, whereby we can have a further speed-up. Specifically m in the complexity bounds can be replaced by m^* in the complexity, where m^* is the number of optimal edges, that is, the edges which are part of shortest paths. Section 11 is the conclusion.

2. Shortest path problems

To prepare for the later development, we describe the single source shortest path algorithm in the following. Let $G = (V, E)$ be a directed graph where $V = \{v_1, \ldots, v_n\}$ and $E \subseteq V \times V$. The cost of edge (u, v) is a non-negative real number denoted by $cost(u, v)$. We specify a vertex, s, as the source. A shortest path from s to vertex v is a path such that the sum of edge costs of this path is minimum among all paths from s to v. The minimum cost is also called the shortest distance. In Dijkstra's algorithm [6] given below, we maintain two sets of vertices, S and F. The set S is the set of vertices to which the shortest distances have been finalized by the algorithm. The set F is the set of vertices which can be reached from S by a single edge. We maintain $d[v]$ for vertices v in S and F. If v is in S, $d[v]$ is the (final) shortest distance to v. If v is in F, $d[v]$ is the distance of the shortest path that lies in S except for the end point v. Let $OUT(v) = \{w \mid (v, w) \in E\}$, and $IN(v) = \{u \mid (u, v) \in E\}$. The solution (the shortest distances from s) is in the array d at the end of the computation. To simplify presentation, only the shortest path distances are calculated, not the shortest paths. We assume all vertices are reachable from the source.

Set F is organized as a heap, such as a Fibonacci heap, that supports delete-min in $O(\log n)$ time, insert/decrease-key in $O(1)$ time. These times are amortized. We could use other heaps that support those operations with the above times in the worst case. Line 6 is delete-min, line 11 is decrease-key, and line 12 is insert.

If we use $IN(v)$ at line 8, u in place of w in lines 8–13 and $cost(u, v)$ in lines 11 and 12, then we name the resulting algorithm Algorithm 2, which solves the single sink shortest path problem. We assume that the sink s is reachable from all vertices.

[1] Precisely speaking, this should be $O(mn + n^2 \log(c/n + 1))$.