On the Comparison-Addition Complexity of All-Pairs Shortest Paths*

Seth Pettie
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
seth@cs.utexas.edu

TR-02-21

May 28, 2002

Abstract

We present an all-pairs shortest path algorithm for arbitrary graphs that performs $O(mn \log \alpha(m,n))$ comparison and addition operations, where m and n are the number of edges and vertices, resp., and α is Tarjan's inverse-Ackermann function. Our algorithm eliminates the sorting bottleneck inherent in approaches based on Dijkstra's algorithm, and for graphs with $O(n)$ edges our algorithm is within a tiny $O(\log \alpha(n,n))$ factor of optimal. Our algorithm can be implemented to run in polynomial time (granted, a large polynomial). We leave open the problem of providing an efficient implementation.

1 Introduction

In 1975 Fredman [F76] presented a simple and elegant algorithm for the all-pairs shortest paths problem that performs only $O(n^{2.5})$ comparison and addition operations, rather than the $O(n^3)$ bound of Floyd's algorithm or Dijkstra's algorithm (see [CLRS01]). However, Fredman gave no polynomial-time implementation of this algorithm,[1] illustrating that the notion of *comparison-addition complexity* in shortest paths problems can be studied apart from the usual notion of *algorithmic complexity*, that is, the actual running times of shortest path programs. We present, in the same vein, an APSP algorithm that makes $O(mn \log \alpha(m,n))$ comparisons and additions, where m and n are the number of edges and vertices, resp., and α is the mind-bogglingly slow-growing inverse-Ackermann function. For sparse graphs, the best comparison-addition-based algorithm to date was established very recently [Pet02]; it runs in $O(mn + n^2 \log\log n)$ time, improving on the long-standing bound of $O(mn + n^2 \log n)$ [Dij59, FT87, J77]. A trivial lower bound on the APSP problem is $\Omega(n^2)$, implying that our algorithm is tantalizingly close to optimal for edge-density $m/n = O(1)$. For dense graphs, the best implementable algorithm to date is due to Takaoka [Tak92], running in time $O(n^3 \sqrt{\frac{\log\log n}{\log n}})$.

It is still an open question whether there are $O(n^2) + o(mn)$ algorithms for APSP when $m = O(n^{1.5})$. Karger et al. [KKP93] have shown that $\Omega(mn)$ is a lower bound among algorithms that only compare path-lengths. Fredman's algorithm obviously does not fit into this class, and neither does our algorithm.

*This work was supported by Texas Advanced Research Program Grant 003658-0029-1999, NSF Grant CCR-9988160, and an MCD Graduate Fellowship.

[1] A slightly worse algorithm making $O(n^{2.5}\sqrt{\log n})$ comparisons/additions can be implemented in $O(n^3)$ time.

This raises the interesting possibility that our techniques could be used to obtain $O(n^2) + o(mn)$ APSP algorithms for sparse graphs.

Our APSP algorithm is based on the divide-and-conquer approach to single source shortest paths invented by Thorup [Tho99] for the special case of undirected graphs, and generalized by Hagerup [Hag00] to directed graphs. The [Tho99, Hag00] algorithms were designed for *integer*-weighted graphs in the RAM model of computation. Their improved running times (over say, Dijkstra's algorithm) depended crucially on the ability of RAMs to sort n numbers in $o(n \log n)$ time. It was, therefore, not obvious whether these algorithms could be translated into good algorithms for real-weighted graphs in the comparison-addition model. Pettie & Ramachandran [PR02b] gave an adaptation of Thorup's algorithm to real-weighted undirected graphs; it solved the s-sources shortest paths problem in $O(sm\alpha(m,n) + \min\{n \log n, n \log\log r\})$ time, where r is the ratio of the maximum-to-minimum edge length. In particular, for $s \geq \log n$ the running time is $O(sm\alpha(m,n))$ and for $s = 1$ and $r = poly(n)$ the running time is $O(m + n \log\log n)$.[2] The techniques used in [PR02b] are specific to undirected graphs and simply have no analogues in directed graphs. Pettie [Pet02], using a different set of techniques, gave a version of Hagerup's algorithm for real-weighted directed graphs. It solves the s-sources shortest paths problem in $O(mn + sn \log\log n)$ time; it is only an improvement over approaches based on Dijkstra's algorithm if $s = \omega(m/\log n)$.

In this paper we attempt to generalize and refine the techniques introduced in [Pet02], in particular the use of *relative distances* and the technique of discretely approximating reals by small integers.

The next Section gives a technical introduction to the commonalities present in the "component hierarchy" based shortest path algorithms [Tho99, Hag00, PR02b, Pet02], and an outline of the approach of this paper.

1.1 Technical Introduction

One way to characterize Dijkstra's SSSP algorithm [Dij59] — without actually specifying it — is to say that it finds a permutation π_s of the vertices such that

$$\pi_s(u) < \pi_s(v) \quad \Rightarrow \quad d(s,u) \leq d(s,v)$$

where $d(\cdot,\cdot)$ is the distance function and s is the source. We give a similar characterization of the shortest path algorithms based on component hierarchies [Tho99, Hag00, PR02b, Pet02].

Suppose for this discussion that the graph is strongly connected. Let $circ(u,v)$ be the set of all cycles containing vertices u and v and let $sep(u,v)$ be defined as

$$sep(u,v) \quad = \quad \min_{\mathcal{C} \in circ(u,v)} \quad \max_{e \in \mathcal{C}} \quad length(e)$$

Notice that if the graph is undirected, $sep(u,v)$ corresponds to the longest edge in the minimum spanning tree path connecting u and v. Regardless of whether the graph is undirected or directed, all component hierarchy-based algorithms [Tho99, Hag00, PR02b, Pet02] generate a permutation π_s satisfying Property 1.

Property 1 *Let s be the source. For vertices u, v, π_s satisfies*

$$d(s,v) \geq d(s,u) + sep(u,v) \quad \Rightarrow \quad \pi_s(u) < \pi_s(v)$$

When characterized in this way we can lower-bound the complexity of specific algorithms in the comparison-addition model via counting arguments. Assume that the graph topology and source are fixed. We will say a set of permutations satisfies Property 1 if for any length function, some permutation in the set satisfies Property 1. The first thing to notice is that $sep(u,v)$ does not depend on the source, whereas $d(s,u)$ and $d(s,v)$ obviously do. From the perspective of a multi-source shortest path algorithm, computing *sep* is a *one-time charge* and computing π_s, given *sep*, relates to the *marginal cost* of computing SSSP from s. Lower bounding both the first SSSP cost and the marginal SSSP cost are of great interest. Let Π_{sep} and Π_{nosep} be the smallest sets of permutations satisfying Property 1, when the *sep* function is known and unknown, respectively. (Computing a component hierarchy gives an implicit approximation to the *sep* function — see

[2] Pettie et al. implemented a simplified version of [PR02b]; in their experiments with real-weighted graphs it consistently outperformed Dijkstra's algorithm.

Section 4.) It is not difficult to show that $\Omega(m + \log|\Pi_{sep}|)$ and $\Omega(m + \log|\Pi_{nosep}|)$ are lower bounds on the number of comparisons/additions performed by any algorithm characterized by Property 1, when *sep* is given a priori, and when no information is provided, respectively.

It can be shown that even for undirected graphs, $|\Pi_{nosep}| = n^{\Theta(n)}$ in the worst case, and $|\Pi_{nosep}| = 2^{\Omega(n \log\log r)}$ if the ratio of any two edge lengths is bounded by $r < 2^n$ – see [PR02b, Pet02]. If *sep* is known the situation is a little different. The shortest path algorithm from [PR02b] demonstrates implicitly that for undirected graphs $|\Pi_{sep}| = 2^{O(m)}$. It is shown in [Pet02] that for directed graphs, even in the special case when $r < n$ and $sep(u, v) = sep(w, z)$ for all u, v, w, z, $|\Pi_{sep}|$ can be as large as $n^{\Omega(n)}$.

What conclusions can be drawn from these bounds? First, any undirected SSSP algorithm satisfying Property 1 must make $\Omega(m+\min\{n \log\log r, n \log n\})$ comparisons, making the [PR02b] algorithm essentially optimal. For directed graphs, even if the *sep* function is known and the component hierarchy given explicitly, any SSSP algorithm satisfying Property 1 makes $\Omega(m + \min\{n \log r, n \log n\})$ comparisons. Clearly, at least one more idea is required to improve the $O(m + n \log n)$ upper bound [Dij59, FT87] on SSSP.

For the all-pairs shortest path problem on directed graphs, the above bounds become weaker. Any APSP algorithm that first computes a component hierarchy (read: computes the *sep* function), then performs n *independent* SSSP computations must make $\Omega(mn + n^2 \log n)$ comparisons. The key technique to improving this bound, which was used to a lesser extent in [Pet02], is to make the SSSP computations *dependent*. In the algorithm presented here, we perform a sequence of n SSSP computations in such a way that later SSSP computations can learn from the time-consuming mistakes of earlier ones.

1.2 Organization

In Section 2 we define the SSSP and APSP problems and the comparison-addition model. In Sections 3 and 4 we review Dijkstra's algorithm and a generic form of the component hierarchy algorithm, which encompasses [Tho99, Hag00, PR02b, Pet02]. In Section 5 we give a method to simulate the generic algorithm in the comparison-addition model. A little familiarity with component hierarchy-based algorithms [Tho99, Hag00, PR02b, Pet02] will be invaluable, though it is not assumed. A brief introduction to the CH approach is given in Appendix B.

2 Preliminaries

The input is a weighted, directed graph $G = (V, E, \ell)$ where $|V| = n, |E| = m$, and $\ell : E \to \mathbb{R}$ assigns a real *length* to every edge. It is well-known [J77] that the shortest path problem is reducible in $O(mn)$ time to one of the same size but having only non-negative edge lengths. We therefore assume that $\ell : E \to \mathbb{R}^+$ assigns only non-negative lengths. We let $d(u, v)$ denote the length of the shortest path from u to v, or ∞ if none exists. The all-pairs shortest path problem is to compute $d(\cdot, \cdot)$ and the single-source shortest paths problem is to compute $d(s, \cdot)$ where the first argument, the *source*, is fixed. Generalizing the d notation, let $d(u, H)$ be the shortest distance from u to any vertex in the subgraph H. H may also be an object that is associated with a subgraph, not necessarily the subgraph itself.

2.1 The Comparison-Addition Model

In the pure comparison-addition model the only operations which contribute toward the "running time" are comparisons between two real numbers and addition of two real numbers. The input can be neatly divided into *structural* information, in our case the unweighted graph, and *numerical* information, in our case the m edge-weights. An algorithm can be modeled as a kind of decision tree (which depends on the structural information): each node is associated with a sequence of additions followed by a comparison. Depending on the outcome of the comparison, the next operation comes from either the left or right child. We frequently use subtraction in our algorithms; refer to [PR02b] for a simulation of subtraction.

This model is elegant and sufficiently powerful to solve the standard shortest path problems (see, e.g., the textbook algorithms of Dijkstra, Bellman-Ford, Floyd-Warshall, and min-plus matrix multiplication.) There are several lower bounds in the comparison-addition model though they are generally very weak. Spira and Pan [SP73] showed that regardless of additions, $\Omega(n^2)$ comparisons are necessary to solve SSSP on the complete graph. Karger et al. [KKP93] proved that all-pairs shortest paths requires $\Omega(mn)$ comparisons if

all summations correspond to paths in the graph. Kerr [K70] showed that any oblivious APSP algorithm performs $\Omega(n^3)$ comparisons/additions, and Kolliopoulos and Stein [KS98] proved that any fixed sequence[3] of edge relaxations solving SSSP must have length $\Omega(mn)$. Graham et al. [G+80] did not give a lower bound but showed that the standard information-theoretic argument cannot yield a non-trivial ($\omega(n^2)$) lower bound in the APSP problem. Similarly, no information-theoretic argument can lower bound SSSP; however, Pettie and Ramachandran [PR02b] observed that the comparison-addition complexity of SSSP is no less than that of the minimum spanning tree problem. So it seems the manifestly simpler MST problem must be fully resolved (see [KKT95, Chaz00, PR02a]) before the complexity of even undirected SSSP can be resolved.

3 Dijkstra's Algorithm

Dijkstra's SSSP algorithm visits vertices in order of increasing distance from the source s. It maintains a set S of visited vertices, initially empty, and a *tentative* distance $D(v)$ for all $v \in V$ satisfying the following invariant.

Invariant 0 *For $v \in S$, $D(v) = d(s,v)$ and for $v \notin S$, $D(v)$ is the shortest distance from s to v using only intermediate vertices from S.*

Dijkstra's method for growing the set S while maintaining Invariant 0 is to visit vertices *greedily*. In each step, Dijkstra's algorithm identifies the vertex $v \notin S$ with minimum tentative distance, sets $S := S \cup \{v\}$, and updates tentative distances. This involves *relaxing* each outgoing edge (v,w), setting $D(w) := \min\{D(w), D(v) + \ell(v,w)\}$. The algorithm halts when $S = V$, and therefore $D(v) = d(s,v)$ — the tentative distances equal the shortest distances.

Throughout the paper D, S, s mean the same thing as in Dijkstra's algorithm, and the terms "visit" and "relax" are essentially the same.

4 The Component Hierarchy

Dijkstra's algorithm can be thought of as simulating a physical process. Suppose the graph-edges represent water pipes and at time zero we begin releasing water from vertex s. Dijkstra's algorithm simulates the flow of water at unit-speed through the graph. Component hierarchy-based algorithms can also be thought of as simulating this process, though in a much coarser way. Instead of maintaining the same simulated time throughout the whole graph, as Dijkstra's algorithm does, CH-based algorithms decompose the graph into a hierarchy of subgraphs (the component hierarchy), where each subgraph maintains its *own* local simulated time. Progress is made by giving a well-selected subgraph, say at simulated time a, permission to advance its clock to simulated time $b > a$. This scheme will satisfy Invariant 0 provided the subgraphs and intervals $[a,b)$ are chosen properly. The correctness of this scheme is addressed in [Tho99, Hag00, PR02b]; see Appendix B for a sketch of its derivation.

4.1 The CH for Real-weighted Directed Graphs

The following *component hierarchy*, which is similar to the one defined in [Pet02], can be constructed very easily with $O(m \log n)$ comparisons and additions.

First, to ensure that G is strongly connected, we add an n-cycle with infinite-weight edges. As in [PR02b], we first produce the edge-lengths in sorted order: $\ell_1, \ldots, \ell_m$. We then find a set of *normalizing* edge lengths $\{\ell_1\} \cup \{\ell_j : \ell_j > n \cdot \ell_{j-1}\}$. Let r_k be the k^{th} smallest normalizing edge. For each edge j between r_k and $r_{k+1} - 1$ we determine the i s.t. $2^{i-1}\ell_{r_k} \le \ell_j < 2^i \ell_{r_k}$. In other words, we find a factor 2 approximation of every edge length divided by its associated normalizing edge length. For the purpose of first understanding the component hierarchy approach, one can assume there is only one normalizing edge length: ℓ_1.

The CH is composed of layered *strata*, where stratum k, level i nodes correspond to the strongly connected components (SCCs) of the graph restricted to edges with length less than $\ell_{r_k} \cdot 2^i$. Most quantities relating

[3] The sequence must only depend on m and n, not the graph.