Expected Performance of Dijkstra's Shortest Path Algorithm

Andrew V. Goldberg

NEC Research Institute
4 Independence Way
Princeton, NJ 08540
avg@research.nj.nec.com

Robert E. Tarjan*

Computer Science Department
Princeton University
Princeton, NJ 08858
and
NEC Research Institute
ret@cs.princeton.edu

June 1996

Abstract

We show that the expected number of `decrease-key` operations in Dijkstra's shortest path algorithm is $O(n \log(1 + m/n))$ for an n-vertex, m-arc graph. The bound holds for any graph structure; the only assumption we make is that for every vertex, the lengths of its incoming arcs are drawn independently from the same distribution. The same bound holds with high probability. This result explains the small number of `decrease-key` operations observed in practice and helps to explain why Dijkstra codes based on binary heaps perform better than ones based on Fibonacci heaps.

*Research at Princeton University partially supported by the National Science Foundation, Grant No. CCR-8920505. Research during a visit to M.I.T. partially supported by ARPA Contract N00013-95-1-1246.

1 Introduction

The shortest path problem with nonnegative arc lengths is one of the most natural network optimization problems and occurs widely in practice. The problem is well-studied. Dijkstra's algorithm [7] is the best known algorithm for the problem in theory and the most robust in practice. The best currently known bounds for the problem, based on a combination of Dijkstra's algorithm and priority queue data structures, appear in [1, 3, 4, 8, 9, 18].

The running time of an implementation of Dijkstra's algorithm is dominated by arc scans (with each arc scanned at most once) and the priority queue operations `insert`, `extract-min`, and `decrease-key`. For a graph with n vertices and m arcs, the number of `insert` and `extract-min` operations is n for each operation. The number of `decrease-key` operations is m in the worst case. Many data structures, such as Fibonacci heaps [8], reduce the cost of the `decrease-key` operation at the expense of added complexity and an increased constant factor for the `extract-min` operation.

In this paper we show that, for a wide class of input distributions, the expected number of `decrease-key` operations is $O(n \log(1 + m/n))$. This result holds in Bloniarz's *endpoint-independent* probabilty model [2]: for every vertex, the lengths of the incoming arcs are drawn independently from the same probability distribution. We make no other assumptions about the arc length distribution or the input graph structure. We also show that the same bound holds with high probability. Note that the probability model where all arc lengths are drawn independently from the same distribution is a special case of the endpoint-independent model.

We are not aware of any previous results on probabilistic analysis of Dijkstra's algorithm. For probabilistic analysis of other shortest path algorithms, including those for the all-pairs problem, see [2, 10, 11, 12, 13, 15, 16].

Our result is motivated by the empirical observation that the number of `decrease-key` operations is small. This observation can be used to explain why Fibonacci heaps usually perform worse than binary heaps (see *e.g.* [5]). More generally, the observation can be used to predict how an implementation of priority queues will perform when applied to Dijkstra's algorithm.

This paper is organized as follows. In Section 2 we state the necessary definitions and previous results. Section 3 is devoted to our main result. In Section 4 we show how to use the result to explain the relative behavior of priority queue data structures when used in Dijkstra's algorithm.

2 Preliminaries

The input to the shortest path problem is a directed graph $G = (V, E)$, a *source vertex* $s \in V$, and a length function $\ell : E \to \mathbf{R}$. In this paper we consider only nonnegative length functions. The goal is to find shortest path distances from s to all vertices of the graph and a shortest path tree. For more details about the problem and Dijkstra's algorithm, see *e.g.* [6, 17].

We denote the number of vertices in G by n and the number of arcs by m. Without loss of generality, we assume that all vertices of G are reachable from s and $1 \le n - 1 \le m$. These assumptions imply that $\log(1 + m/n) > 0$. We denote the in-degree of a vertex v by $\delta(v)$. Note that $\delta(v) \ge 1$.

We say that a weighted graph is *endpoint-independent* if for every vertex, lengths of its incomming arcs are drawn independently from the same probability distribution over nonnegative real numbers.

We say that a bound expressed in terms of n holds *with high probability* (*w.h.p.*) if the probability that the bound holds goes to one as n goes to infinity.

Let $x_1, \ldots, x_k$ be chosen independently from the same probability distribution. For $1 \le i \le k$, define $X_i = \min_{1 \le j \le i} x_j$. Let $M(k)$ be the number of distinct values of X_i.

The following simple result is part of the folklore; see for example problem 6-2 on page 133 of [6].

Theorem 2.1 *$E(M(k)) = O(\log k)$. $M(k) = O(\log k)$ w.h.p.*

Proof. For completeness, we include a proof that $E(M(k)) = O(\log k)$. Let $\chi_i = 1$ if $x_i < x_j$ for all $1 \le j \le i$, $\chi_i = 0$ otherwise. Then $M(k) = \sum_{i=1}^{k} \chi_i$. Since the x_i's are independent and identically distributed, $\chi(X_i) \le 1/i$. Thus $E(M(k)) = \sum_{i=1}^{k} E(\chi_i) \le \sum_{i=1}^{k} 1/i = O(\log k)$. ■

3 Main Result

In this section we prove the following result.

Theorem 3.1 *For an endpoint-independent graph, the expected number of* `decrease-key` *operations performed by Dijkstra's shortest path algorithm is $O(n \log(1 + m/n))$. The same bound holds w.h.p.*

Proof. Recall that the algorithm maintains distance estimates $d(v)$ for every vertex v. The algorithm scans every arc of the graph at most once. The *decrease-key* operation applies to v if during a scan of an arc (u, v), $d(u) + \ell(u, v) < d(v)$ and this is not the first scan of an arc into v.

Consider the set of all arcs $\{(u_1, v), \ldots, (u_{\delta(v)}, v)\}$ into v. Suppose that all the arcs are scanned and, without loss of generality, assume that they are scanned in the order of the subscripts of u. Also assume that when the arcs are scanned, the values of $d(u_i)$ are the same. (We shall deal with these two assumptions below.) A `decrease-key` operation is applied during a scan of an arc (u_i, v) if and only if $i > 1$ and $\ell(u_j, v) > \ell(u_i, v)$ for all $1 \le j < i$. By Theorem 2.1, the expected number of `decrease-key` operations on v is $O(\log \delta(v))$ and the same bound holds w.h.p.

Since the logarithm function is convex and the average in-degree is m/n, we have

$$\sum_{v \in V} \log(\delta(v)) \le n \log(m/n) < n \log(1 + m/n).$$

Thus the expected number of `decrease-key` operations is $O(n \log(1+m/n))$ and the same bound holds w.h.p.

Next we deal with the assumptions we made above. In general, these assumptions do not hold. We show that the bound does not get worse when the assumptions do not hold. If the first assumption does not hold, *i.e.,* not all arcs are scanned, the expected number of operations can only decrease.

To deal with the second assumption, note that the standard properties of Dijkstra's algorithm imply that $d(u_1) \le d(u_2) \ldots \le d(u_{\delta(v)})$. It follows that

$$\{i | d(u_i) + \ell(u_i, v) < d(v) \text{ when } v \text{ is scanned}\} \subseteq \{i | \ell(u_j, v) > \ell(u_i, v) \; \forall \; 1 \le j \le i\}.$$

The former set of indices corresponds to the insertion of v and the `decrease-key` operations on v during the running of Dijkstra's algorithm. Since the latter set has size $O(\log \delta(v))$ w.h.p., so does the former. ■

4 Applications

Consider the binary heap data structure. For this data structure, all priority queue operations take $O(\log n)$ time. This gives an $O(m + n \log n \log(1 + m/n))$ expected time bound for the binary heap implementation of Dijkstra's algorithm, compared with the $O(m \log n)$ worst-case bound. For Fibonacci heaps, the amortized time bounds for the `insert`, `decrease-key`, and `extract-min` operations are $O(1)$, $O(1)$, and $O(\log n)$, respectively. This gives an $O(m+n \log n)$ expected time bound on the Fibonacci heap implementation of the algorithm, the same as the worst-case bound.

The two expected-time bounds differ only for $\omega(n) = m = o(n \log n \log \log n)$, and the ratio between them is $o(\log \log n)$. Given the current memory limitations, one can assume that $n < 2^{32}$ and $\log \log n < 5$.

On the other hand, the constant factors of implementations of the more complex Fibonacci heaps are greater than those for the binary heaps. If the constants are greater by a factor of 5 or more, Fibonacci heaps are slower than binary heaps on sparse graphs. For dense graphs, the arc scans dominate the priority queue operations, and the two implementations perform similarly. The same holds for k-ary heaps with small values of k. This is consistent with the data in [5].

More precisely, for any heap size, the cost of `extract-min` is significantly higher in the Fibonacci heap implementation than in the binary heap implementation. The costs of `insert` and `decrease-key` are somewhat less for the Fibonacci heap implementation, with the difference increasing with the heap size. However, for practical heap sizes, many `insert` and `decrease-key` operations are needed to make up the difference in the cost of the `extract-min` operations. The number of `decrease-key` operations is not large enough to make Fibonacci heaps perform better than binary heaps.

Theorem 3.1 can be used to explain the relative performance of other implementations of Dijkstra's algorithm. Given approximate heap operation times, we can also use the theorem to predict implementation performance.

The above discussion suggests that Fibonacci heaps might outperform binary heaps in an application in which the number of `decrease-key` operations is much higher than the number of `extract-min` operations. The minimum cut algorithm of Nagamochi and Ibaraki [14] provides such a candidate application. This algorithm works on an undirected graph and uses a max priority queue instead of a min priority queue (*i.e.,* `extract-max` and `increase-key` operations replace `extract-min` and `decrease-key` operations, with no difference in performance). During an iteration for which the current graph has n vertices and m edges, the algorithm performs n `insert`, n `extract-max`, and m `increase-key` operations. On large dense graphs, Fibonacci heaps performed better than binary heaps in a few tests we ran.

References

[1] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. Faster Algorithms for the Shortest Path Problem. *J. Assoc. Comput. Mach.*, 37(2):213–223, April 1990.

[2] P.A. Bloniarz. A Shortest Path Algorithm with Expected Time $O(n^2 \log n \log^* n)$. *SIAM J. Comput.*, 12:588–600, 1983.

[3] P. Van Emde Boas, R. Kaas, and E. Zijlstra. Design and Implementation of an Efficient Priority Queue. *Math. Systems Theory*, 10:99–127, 1977.

[4] B. V. Cherkassky and A. V. Goldberg. Heap-on-Top Priority Queues. Technical Report 96-042, NEC Research Institute, Princeton, NJ, 1996.

[5] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 516–525, 1994. To appear in Math. Prog.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[7] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1:269–271, 1959.

[8] M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. Assoc. Comput. Mach.*, 34:596–615, 1987.

[9] M. L. Fredman and D. E. Willard. Trans-dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. In *Proc. 31st IEEE Annual Symposium on Foundations of Computer Science*, pages 719–725, 1990.

[10] A.M. Frieze and G.R. Grimmet. The Shortest-Path Problem for Graphs with Random Arc-Lengths. *Discrete Applied Mathematics*, 10:57–77, 1985.

[11] Q.P. Gu and T. Takaoka. On the Average Path Length of $O(\log N)$ in the Shortest Path Problem. *Trans. IEICE*, E70:1155–1158, 1987.

[12] S.G. Kolliopoulos and C. Stein. Finding Real-Valued Single-Source Shortest Paths in $o(n^3)$ Expected Time. In *Proc. 5th Int. Programming and Combinatorial Optimization Conf.*, 1996.

[13] A. Moffat and T. Takaoka. An All-Pairs Shortest Path Algorithm with Expected Time $O(n^2 \log n)$. *SIAM J. Comput.*, 16:1023–1031, 1987.

[14] H. Nagamochi and T. Ibaraki. Computing Edge-Connectivity in Multigraphs and Capacitated Graphs. *SIAM J. Disc. Meth.*, 5:54–66, 1992.

[15] P.M. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time $o(n^2 \log^2 n)$. *SIAM J. Comput.*, 2:28–32, 1973.

[16] P. Spirakis and A. Tsakadidis. A Very Fast, Practical Algorithm for Finding a Negative Cycle in a Digraph. In *Proc. 13th ICALP, Lecture Notes in Computer Science 226*, pages 59–67. Springer-Verlag, 1996.

[17] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[18] M. Thorup. On RAM Priority Queues. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 59–67, 1996.